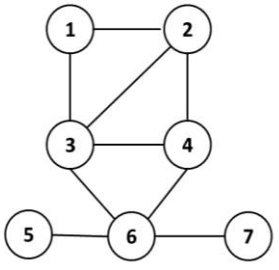
تصميم وتحليل خوارزميات الحاسوب

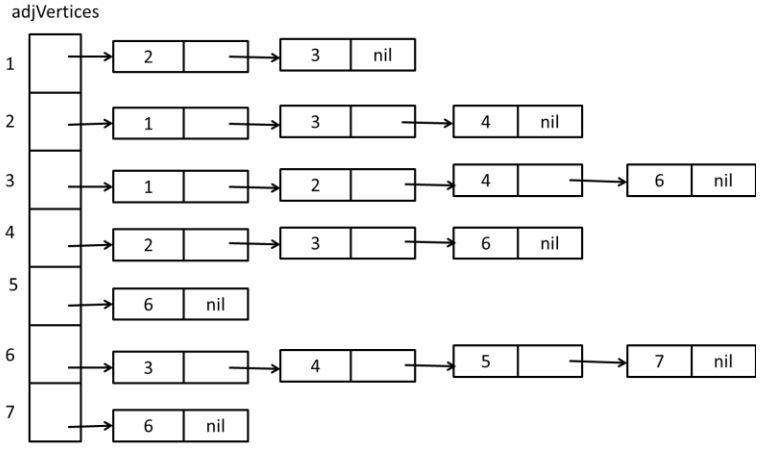
د. أبو بكر أحمد السيد
قسم علوم الحاسوب
جامعة الكويت

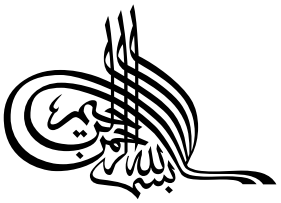
د. حمزة سيد رشوان
قسم الرياضيات
جامعة الأزهر

$$T(n) = n + 2\,T\left(\frac{n}{2}\right), \qquad T(1) = 0$$

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$







الطبعة الأولى
1439 هـ - 2018 م

المحتويات

❊ ❊ ❊

مُقَدِّمَة

الحمد لله رب العالمين، والصلاة والسلام على خاتم المرسلين نبينا مُحَمَّد وعلى آله وصحبه أجمعين.

منذ أكثر من أربعة عشر قرنا، ومنذ أن نزل الروح الأمين جبريل عليه السلام بالوحي مبتدئا بكلمة اقرأ على محمد صلى الله عليه وسلم اكتسبت لغتنا العربية أهمية خاصة، وصفة تميزها عن غيرها من لغات العالم قاطبة، وتحولت من لغة محلية ضيقة تتحدث بها مجموعة من القبائل إلى لغة عالمية إنسانية حضارية، فانتقلت اللغة العربية من مكانها في الجزيرة العربية وفي أجزاء من الشام والعراق لتغزو المشارق والمغارب لتصل إلى الهند والصين شرقا وإلى أوروبا غربا مرورا بإفريقيا .. وأقبل الكثيرون من العرب والأعاجم على اللغة العربية بالدراسة والتحليل حتى استطاعت أن تستوعب بسهولة ويسر النتاج الفكري والعلمي لمختلف الحضارات في العالم. ولم يقتصر عملهم على ترجمة علوم الآخرين فحسب، وإنما بعد ذلك كان المزج بين الحضارات وكان الإبداع وكان السبق والتأليف في مختلف العلوم باللغة العربية.

وقد كانت هناك عوامل عديدة دفعت هؤلاء جميعا إلى تعلم وإتقان واستخدام اللغة العربية في شتى المجالات:

- فطلب العلم فريضة على كل مسلم، وهي علم من العلوم.
- وتعلم العربية كذلك نوع من أنواع العبادة، لأننا بدون هذه اللغة لا نستطيع أن نفهم القرآن الكريم وأحكام الدين فهما جيدا، فتعلم اللغة العربية أكثر فرضية من تعلم غيرها من اللغات أو كثير من العلوم.
- وقد غدت اللغة العربية لغة الدولة والعلوم، والحكومة والسلطان، فأصبح

لزاما على من يريد أن ينهل من علم من العلوم أو أن يصل إلى منصب من المناصب أن يتعلم لغة الدولة التي يظله لواؤها.

ولما كانت اللغة مرتبطة عموما ارتباطا وثيقا بأهلها، يصيبها ما يصيبهم، فلما أصيب العرب بالضعف، ولحقت بهم المآسي، وتخلفوا عن ركب الحضارة، كان طبيعيا أن تتخلف لغتهم عن ركب الحضارة إلى يومنا هذا وإلى أن يهيئ الله لها أقواما كسلفنا الصالح وأجدادنا العظماء ينفضون عنها غبار الدهور، ويبعثونها من جديد حية مشرقة.

وجامعات الدول العربية بأساتذتها وطلابها أول من تقع عليهم مسئولية إحياء اللغة العربية، وجعلها لغة العلوم والثقافة والدرس والتعليم كما كانت سابقا في الحضارة الإسلامية العالمية السامقة، خاصة وأن اللغة العربية هي أوسع اللغات العالمية مدى، وأغزرها مادة، لكثرة أبنيتها، وتعدد صيغها، ومرونتها على الاشتقاق، واستعدادها للتطور، وقدرتها على الاستيعاب، وموادها لا تقل عن ثمانين ألف مادة، وهذا الغنى أعانها على أن تكون لغة خالدة، بالإضافة إلى كونها لغة القرآن الكريم، فهي خالدة بخلوده: {إِنَّا نَحْنُ نزَّلْنَا الذِّكْرَ، و إِنَّا لَهُ لَحَافِظُون} . وفي اللغة العربية من أسباب النمو ما يحفظ لها شبابها، ويجعلها تنجح في أن تكون لغةً لحاضرٍ عظيمٍ مشرق، كما كانت من قبل لماضٍ عريق .. لغة لكتاب الله تعالى، وللدين الخاتم، وللأمة المجاهدة التي نشرت هذا الدين في المشارق والمغارب، وحملت هذه الرسالة وكافة أنواع العلوم والمعرفة إلى الأمم القاصية والدانية.

هذا الكتاب .. والغاية منه

وضمن جهود التعريب في ميدان العلوم الحديثة ظهرت والحمد لله في السنوات الأخيرة بعض الكتب باللغة العربية (مؤلفة أو مترجمة) في مجال علم

الحاسوب، وهذه الكتب وإن كانت قليلة جدا بالنسبة لما هو مطلوب في عملية تعريب العلوم الحديثة للنهوض بأمتنا، إلا أنها جهد مشكور ونواة طيبة لمكتبة علمية عربية ندعو الله عز وجل أن يبارك فيها لتثري وتروي ظمأ الطلاب والدارسين والباحثين. وهذا الكتاب إضافة جديدة لهذه المكتبة في موضوع تصميم وتحليل خوارزميات الحاسوب، ونأمل بذلك أن ينفع الله تعالى بإذنه سبحانه بهذا الكتاب أبناء العربية، وأن يسد الكتاب نقصا في المكتبة العلمية العربية، ويكون بذلك لبنة جديدة في صرح تعريب العلوم الحديثة، وجهدا متواضعاً ندعو الله عز وجل أن يبارك فيه فيثقل ميزان حسناتنا يوم نقف بين يدي المولى عز وجل فيسألنا ماذا قدمنا لغدنا، وماذا عملنا من أجل النهوض بأمتنا، ورفع راية التوحيد بين الخلق أجمعين.

وتُعَدُ دراسة الخوارزميات في صميم قلب علوم الحاسوب، ويهدف الكتاب إلى أن يجمع بعض النتائج الأساسية في هذا المجال، بحيث يسهل بإذن الله تدريس مبادئ ومفاهيم تصميم الخوارزميات وتحليلها. أي أن الكتاب يعرض مقدمه شاملة للدراسة الحديثة لخوارزميات الحاسوب، فيقدم للقارئ الكريم خوارزميات عديدة ويغطيها بعمق إلا أنه يجعل تصميمها وتحليلها في متناول جميع مستويات القراء، حيث حاولنا الحفاظ على بساطة الشرح دون التضحية بعمق التغطية أو جزالة محتوى الرياضيات في الخوارزمية.

والكتاب يصلح لتدريسه في فصل دراسي واحد كمقرر أولى في تصميم وتحليل الخوارزميات، حيث التركيز فيه على الأفكار الأساسية وسهولة الفهم بدلا من تفاصيل التنفيذ أو أساليب البرمجة، وغالبا ما نلجأ إلى الشرح البدائي البسيط البديهي غير الشكلي الرسمي بدلا من البراهين الطويلة المملة. والكتاب يعد متكاملا بذاته، ولا يتطلب خلفية رياضية عميقة في الرياضيات أو لغات

البرمجة، ولجعل بعض الخوارزميات أسهل في القراءة والمتابعة فقد كتبت بلغة حديثة باستخدام خصائص لغة C++، ومعظم الخوارزميات التي عرضناها في الكتاب ليست برامج، بمعنى أن كثيرا من التفاصيل التي تعد غير هامة بالنسبة لطريقة تطبيق الخوارزمية أو لتحليلها قد حذفت، وكثير من المحاضرين قد يفضلون تدريس هذا المقرر باعتباره مقررا نظريا بحتا دون برمجة.

ومن أهداف الكتاب تدريس الخوارزميات لحل مسائل حقيقية عملية تنشأ مرارا في التطبيقات الحاسوبية، وتدريس المبادئ الأساسية وطرق حساب درجة التعقيد الحاسوبية (أسوا حالة، والسلوك المتوسط، والحيز المستخدم، والحدود الدنيا لدرجة تعقيد مسألة ما)، وأيضا لنغرس في القارئ الكريم حس وعادة مواجهة أي خوارزمية جديدة بالأسئلة عن مدى جودة الخوارزمية، وهل هناك طريقة أفضل لتحسينها، وذلك بأن نعطي أحيانا خوارزمية معينة ونقوم بتحليلها وتعديلها أو رفضها إلى أن نصل إلى نتيجة مقبولة، ولذلك أحيانا نتناول في ثنايا الكتاب أسئلة مثل: كيف يمكننا تنفيذ هذه الخطوة بكفاءة أفضل؟، ما هي بنية المعطيات التي تفيدنا في هذا الموضوع؟، أي العمليات يجب أن نركز عليها لتحليل هذه الخوارزمية؟، كيف يجب علينا إعطاء هذا المتغير (أو بنية المعطيات هذه) قيمة ابتدائية؟

كذلك نأمل أن يصل القارئ الكريم إلى فهم السلوك الفعلي لأي خوارزمية المقابل للمدخلات المختلفة، أي ما هي المسارات التي ستسلكها الخوارزمية حين استقبال مدخلات معينه، وكيف يؤثر عرض المدخلات بطرق مختلفة (مثل عرض قائمة رؤوس أو أحرف مخطط بياني معين بترتيب مختلف) على سلوك الخوارزمية، وبعض هذه الأسئلة تثار في تمرينات فصول الكتاب. ومعظم الخوارزميات التي تم عرضها في الكتاب ذات طبيعة عملية.

وعند مناقشتنا كثيرا من الخوارزميات استخدمنا شبه الكود (الكود الزائف) وليس الكود الفعلي لأي لغة برمجة، فعرض الخوارزميات المعقدة باستخدام جميع تفاصيل أي لغة برمجة سيعيق فهم الطلاب للخوارزميات. وبالإضافة إلى ذلك فإن شبه الكود سيكون مفهوما لأي شخص متقن لأي لغة برمجة عالية المستوى، وذلك يعني أنه يجب قدر الاستطاعة تجنب أي تفاصيل خاصة بأي لغة برمجة.

فصول الكتاب:

نفرض ان القارئ الكريم على علم بهياكل البيانات كالمنظومات والقوائم المترابطة والأشجار، وكذلك على علم بموضوع الارتداد، وتحليل الخوارزميات يستخدم بعض الخواص البسيطة للوغاريتمات، وشيئا من حساب التفاضل والتكامل (كالتفاضل لتحديد رتبة دالةٍ ما، والتكامل لتقريب المجاميع). ويعرض **الفصل الأول** في الكتاب مراجعة مختصرة لأهم القوانين والقواعد والعلاقات في الرياضيات التي نحتاجها لتحليل الخوارزميات، وكذلك موضوع الرتبة المقارِبة.

وقد تناولت الفصول التالية طرق تصميم بعض الخوارزميات الهامة وتطبيقاتها في مجالات مختلفة كخوارزميات "قَسِّمْ وتَغَلَّبْ" أو"فَرِّق تَسُدْ" واستخدامها في "البحث الثنائي" وفي معظم طرق الترتيب / الفرز، وفي أشجار البحث الثنائي، والخوارزميات "الجشعة" واستخدامها لإيجاد الأشجار المولدة الصغرى وإيجاد أقصر المسارات، وخوارزميات "البحث بالعمق أولا" وتطبيقاتها في اجتياز المخططات البيانية.

ويوضح **الفصل الثاني** تطبيق خطوات التحليل الأساسي للخوارزميات على أمثلة البحث في منظومة مرتبة أو غير مرتبة، والبحث التتابعي، والبحث الثنائي،

وإيجاد أكبر عنصر في منظومة، وضرب المصفوفات. ويتناول **الفصل الثالث** عن الترتيب أو الفرز تطبيق خوارزميات الترتيب بالإدخال، والترتيب السريع، والترتيب بالدمج، والترتيب بالكومة، وغيرها. أما **الفصل الرابع** عن المخططات البيانية فيعرض تعريفاتها وطرق تمثيلها واجتيازها، ويناقش خوارزميات البحث بالعمق أولا، والبحث بالعرض أولا، وكلا من خوارزمية "بريم" وخوارزمية "كروسكال" لإيجاد شجرة مولدة صغرى، وخوارزمية "ديجسترا" لإيجاد أقصر مسارات من مصدر أحادي.

شكر و تقدير

ونود أن نسجل هنا شكرنا العميق لكل من ساهم بأى جهد فى صدور هذا الكتاب، ونخص بالشكر ((مكتبة دار اقرأ بالكويت)) لجهدها معنا فى تيسير نشر وتوزيع الكتب العلمية باللغة العربية مما يساعد على دفع عملية التعريب خطوات للأمام، وكذلك نتقدم بخالص الشكر للأستاذ/ **مصطفى حسن** بقسم الرياضيات بكلية العلوم جامعة الأزهر الذى لم يدخر جهدا فى طباعة هذا الكتاب. وكذلك نتقدم بخالص الشكر والتقدير لإبنتنا / **علياء حمزة** لقيامها بمراجعة الكثير من الخوارزميات والرسومات في هذا الكتاب.

ونأمل أن يكون الكتاب قد عرض بطريقة مشوقة مقدمة لمجال الخوارزميات وطرق تصميمها وتحليلها.

وآخر دعوانا أن الحمد لله رب العالمين.

المؤلفـان ،

الفصل الأول:

مراجعة بعض القوانين والعلاقات في الرياضيات

أولاً: المجموعات والمتتابعات .

ثانياً: العلاقات والدوال

ثالثاً: الجبر والتفاضل .

رابعاً: نظرية الاحتمالات

خامساً: التجميعات والمتسلسلات

سادسا: الدوال الرتيبة والمحدَّبة

سابعا: التجميعات باستخدام التكامل

ثامنا: المتباينات

تاسعا: المنطق

الفصل الأول :

مراجعة بعض القوانين والعلاقات في الرياضيات

Review of some Mathematical Rules and Relations

يشتمل هذا الفصل على مراجعة سريعة مختصرة لأهم القوانين والقواعد والعلاقات التي تتناولها المفاهيم الرياضية والطرق الرياضية التي سنستخدمها بإذن الله تعالى في هذا الكتاب.

أولا: المجموعات والمتتابعات Sets and Sequences

من أمثلة طرق تعريف **مجموعةٍ (set)** ما:

$S_1 = \{a, b, c\}$, $S_1 = \{b, c, a\}$, $S_2 = \{x \mid x \text{ is an integer power of } 2\}$,

$S_3 = \{1, 2, \ldots, n\}$, $S_3 = \{i \mid 1 \le i \le n\}$.

وعدد عناصر المجموعة S (**cardinality** of) يُرمَز له بالرمز $|S|$.

والمتتابعة (**sequence**) هي مجموعة من العناصر (a group of elements) مرتبة ترتيبا محددا (a specified order). فالمتتابعات الثلاث التالية هي متتابعات مختلفة (distinct sequences): (a, b, c), $(b, c,$ a$)$, $(a, b, c,$ a$)$. ويلاحظ أن المتتابعة – بخلاف المجموعة – يمكن أن تحتوي على عناصر مكررة، مثل المتتابعة (a, b, c, a). وإذا لم تحتو متتابعة محدودة

(a finite sequence) على عناصر مكررة – أي كانت جميع عناصرها مختلفة (distinct) – فيُقال إن المتتابعة "**تبديل**" (a permutation) للمجموعة المحدودة (finite set) المكوَّنة من العناصر نفسها. ويلاحظ أن مجموعة واحدة مكونة من n عنصر لها تبديلات / تباديل مختلفة (distinct permutations) عددها $n!$.

وأي مجموعة محدودة عدد عناصرها n تحتوي على مجموعات جزئية مختلفة (distinct subsets) عددها 2^n. وأما عدد المجموعات الجزئية المختلفة التي تتكون كل منها من عناصر عددها (distinct subsets of cardinality) k مختارة من مجموعةٍ مكوَّنة من n عنصر فهو

$$C(n,k) \equiv \binom{n}{k} = \frac{n(n-1)\ldots(n-k+1)}{k!} = \frac{n!}{(n-k)!\,k!}, \quad n \geq k \geq 0 \qquad (1)$$

ونظراً لأن عدد عناصر أي مجموعة جزئية من هذه المجموعات الجزئية المختلفة هو عدد تتراوح قيمته من 0 إلى n، فلذلك نصل إلى المتطابقة (identity)

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \qquad (2)$$

ويُعـرَّف **حاصـل الضـرب الكـارتيزي / التبـادلي (Cartesian product)** كما يلي

$$S \times T = \{(x, y) \,|\, x \in S, y \in T\} \qquad (3)$$

ولذلك فإن $|S \times T| = |S|\,|T|$.

ثانيا: **العلاقات والدوال** Relations and Functions

تعريف بعض الخواص الهامة للعلاقات

Important properties of relations

تعريف 1–1:

نفرض أن $R \subseteq S \times S$. يقال إن العلاقة $\boldsymbol{R}$ (relation) على المجموعة $\boldsymbol{S}$:

- **علاقة انعكاسية (reflexive)** إذا تحقق الشرط

$$\forall x \in S \qquad (x, x) \in R$$

- **علاقة متماثلة / متناظرة (symmetric)** إذا تحقق الشرط

$$(y, x) \in R\ ] \quad \forall x, y \in R \quad \Rightarrow \quad [\ (x, y) \in R$$

- **علاقة قطرية التناظر (anti-symmetric)** إذا تحقق الشرط

$$x \neq y\ ) \ \Rightarrow \ (y, x) \notin R\ ] \quad \forall x, y \in R \& \ [\ (\ (x, y) \in R$$

- **علاقة متعدية (transitive)** إذا تحقق الشرط

$$(y, z) \in R \ \Rightarrow \ (x, z) \in R] \quad \forall x, y, z \in R \& [(x, y) \in R$$

- **علاقة تكافؤ (equivalence relation)** إذا كانت R علاقة انعكاسية ومتماثلة ومتعدية. وعادةً يُرمز لعلاقة التكافؤ بالرمز "$\equiv$".

وعلاقة التكافؤ تُجَزِّئ (partitions) المجموعة $\boldsymbol{S}$، أي تُقَسِّم S إلى مجموعة (collection) من المجموعات الجزئية المتباعدة (disjoint subsets) $\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots$ يطلق عليها "**طبقات التكافؤ**" **(equivalence classes)**، بحيث أن جميع العناصر في $\boldsymbol{S_1}$ تكون مكافئة (equivalent) لبعضها البعض، وكذلك جميع العناصر في $\boldsymbol{S_2}$ تكون مكافئة لبعضها البعض، وهكذا.

ثالثا: الجبر والتفاضل Algebra and Calculus

دالّتا الأرضية والسقف Floor and Ceiling Functions

تعريف 1–2:

أرضية (floor) أي عدد حقيقي x (real number)، ونرمز لها بالرمز $\lfloor x \rfloor$: **هي أكبر عدد صحيح أقل من أو يساوي** x.

وسقف (ceiling) أي عدد حقيقي x، **و نرمز له بالرمز** $\lceil x \rceil$: **هو أصغر عدد صحيح أكبر من أو يساوي** x.

فمثلا:

$$\lfloor 8.9 \rfloor = 8, \quad \lceil 9.1 \rceil = 10$$

اللوغاريتمات Logarithms

تعريف 1–3:

دالة اللوغاريتم Logarithm Function

نفرض أن $b > 1, x > 0$. **يُعَرَّف** $\log_b x$ **بأنه العدد الحقيقي** L **الذي يحقق العلاقة** $b^L = x$، **أي أن** $\log_b x$ **هي القوة / الأس الذي يجب أن تُرفع إليه** b **حتى نحصل على** x .

فرضية 1–1: خواص اللوغاريتمات

نفرض أن x, y **عددان حقيقيان موجبان اختياريان، وأن** a **أي عدد حقيقي، وأن** $b > 1, c > 1$ **عددان حقيقيان. دالة اللوغاريتم تحقق الخواص التالية:**

1. $x > y \quad \log_b x > \log_b y.$
2. $\log_b x = \log_b y \quad x = y.$
3. $\log_b 1 = 0.$
4. $\log_b b^a = a.$
5. $\log_b (xy) = \log_b x + \log_b y.$
6. $\log_b x^a = a \log_b x.$
7. $x^{\log_b y} = y^{\log_b x}.$
8. $\log_c x = (\log_b x)/(\log_b c).$

اصطلاحات Notations

$\lg x = \log_2 x, \quad \ln x = \log_e x, \quad \lg e \approx 1.443, \quad \lg 10 \approx 3.32$

$\lg \lg(x) = \lg(\lg(x)) = \lg^{(2)}(x) \neq (\lg(x))^2.$

$\lg^{(3)}(65536) = 2, \quad (\lg(65536))^3 = 4096.$

$n = 2^k \Rightarrow \lg n = k.$

$2^k \leq n < n + 1 \leq 2^{k+1} \Rightarrow \lfloor \lg n \rfloor = k \ \& \ \lceil \lg (n + 1) \rceil = k + 1.$

$n \leq 2^{\lceil \lg n \rceil} < 2n.$

$\frac{n}{2} < 2^{\lfloor \lg n \rfloor} \leq n.$

رابعا: نظرية الاحتمالات Probability Theory

نفرض أن لدينا مجموعة أحداث بسيطة (elementary events) $\{s_1, \ldots, s_k\}$ أَيُّ مجموعة جزئية S من هذه المجموعة يطلَق عليها "حَدَث" (event). واحتمال هذا الحدث هو $\mathbf{Pr}(S) = \Sigma_{s_i \in s} \mathbf{Pr}(s_i)$.

تعريف 1–4:

الاحتمال الشَّرْطِي / المشروط **Conditional Probability**

الاحتمال الشَّرْطِي لحدث S إذا أُعطينا حَدَثاً T يُعَرَّف بأنه

$$\Pr(S|T)=\frac{\Pr(S \text{ and } T)}{\Pr(T)}=\frac{\sum_{s_i \in S\cap T}\Pr(s_i)}{\sum_{s_j \in T}\Pr(s_j)} \qquad \textbf{(4)}$$

حيث مَدَى s_i, s_j يقع على الأحداث البسيطة.

تعريف 1–5:

الاستقلال التصادفي **Stochastic independence**

يقال إن الحدثين S, T "مستقلان تصادفيا" (stochastically independent) **أو ببساطة "مستقلان"** (independent)– **إذا كان**

$$Pr(S \text{ and } T) = Pr(S)\, Pr(T)$$

وإذا كانت S مستقلة تصادفيا عن T، فإن $Pr(S \mid T) = Pr(S)$.

والمتغير العشوائي (random variable) **هو متغير ذو قيمة حقيقية** (a real valued variable) **يعتمد على أيٍّ الأحداث البسيطة** (elementary events) **قد حدث** (occurred). **وبأسلوب آخر فالمتغير العشوائي هو دالة مُعَرَّفة للأحداث البسيطة.**

مثلا: **إذا كان عدد العمليات** (number of operations) **التي تقوم بها خوارزميةٌ معينة يعتمد على المدخلات** (input)، **وكان كل مُدْخَل مُحتَمَل** (possible input) **حدثاً بسيطا، فإن عدد العمليات يكون متغيرا عشوائيا.**

تعريف 1–6:

التَّوقُّع والتوقُّع المشروط Expectation and conditional expectation

نفرض أن $f(e)$ متغير عشوائي مُعَرَّف على مجموعة من الأحداث البسيطة $e \in U$ [حيث U هي المجموعة الشاملة لهذه الأحداث البسيطة e].

تَوَقُّع f (expectation) [ونرمز له بـ $E(f)$] يُعَّرف بالصيغة

$$E(f)=\sum_{e\in U} f(e)\Pr(e)$$

وهذه القيمة يُطلَق عليها أيضا **القيمة المتوسطة (average value) للمتغير f . والتوقُّع المشروط للمتغير f (conditional expectation of)** إذا عُلِم / أُعطينا الحدث S يُعَرَّف بالصيغة

$$E(f\,|\,S)=\sum_{e\in U} f(e)\Pr(e\,|\,S)=\sum_{e\in S} f(e)\Pr(e\,|\,S)$$

[" = " لأن الاحتمال المشروط لأي حدث (event) ليس في S يساوي صفرا 0].

تمهيدية 1–2: قوانين التوقعات Laws of expectations

إذا كان $f(e)$, $g(e)$ متغيرين عشوائيين مُعَرَّفين على مجموعة من الأحداث البسيطة $e \in U$، وكان S أي حَدَث (any event)، فإن

$$E(f + g) = E(f) + E(g),$$

$$E(f) = Pr(S)\, E(f \mid S) + Pr(\text{not } S)\, E(f \mid \text{not } S).$$

خامسا: التجميعات والمتسلسلات Summations and Series

هناك عدة تجميعات ومتسلسلات تظهر مرارا عند تحليل الخوارزميات. ونذكر فيما يلي صيغا لبعض هذه التجميعات والمتسلسلات.

(i) **المتسلسلة الحسابية Arithmetic Series**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad (5)$$

(ii) **المتسلسلة الحدودية Polynomial Series**

نذكر أولا مجموع المربعات (sum of squares):

$$\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6} \qquad (6)$$

والحالة العامة (أي مجموع الأعداد مرفوعة لأي أُس / قوة k) هي:

$$\sum_{i=1}^{n} i^k \approx \frac{1}{k+1} n^{k+1} \qquad (7)$$

ومتسلسلة **قُوى 2 (Powers of 2)** [وتُعتبر حالة خاصة من المتسلسلة الهندسية] هي:

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \qquad (8)$$

(iii) **المتسلسلة الهندسية Geometric Series**

$$\sum_{i=o}^{k} a\, r^i = a\left(\frac{r^{k+1}-1}{r-1}\right) \quad , \quad r \neq 1 \qquad (9)$$

وكحالة خاصة من هذه المتسلسلة عندما $a = 1, r = \frac{1}{2}$ نحصل على

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k} \qquad (10)$$

(iv) **المتسلسلة التوافقية Harmonic Series**

$$\sum_{i=1}^{n} \frac{1}{i} \approx 1n(n) + \gamma, \qquad where\ \gamma \approx .577 \qquad (11)$$

والمجموع يُطلَق عليه "العدد التوافقي النوني" (*n*-th Harmonic number)، والثابت γ يُطلَق عليه "ثابت أويلر" (Euler's constant).

(v) **المتسلسلة الحسابية الهندسية Arithmetic-Geometric Series**

$$\sum_{i=1}^{k} i\,2^i = (k-1)2^{k+1} + 2 \qquad (12)$$

(vi) **أعداد "فيبوناتشي" Fibonacci Numbers**

تُعَرَّف متتابعة "فيبوناتشي" ارتداديا كما يلي

$$\text{for } n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0, \quad F_1 = 1. \qquad (13)$$

سادسا: الدوال الرتيبة والمحدَّبة Monotonic and Convex functions

تعريف 1-7:

الدوال الرتيبة وغير الرتيبة

Monotonic and anti-monotonic functions

يقال لدالة $f(x)$ إنها **دالة رتيبة (monotonic)**، أو **غير متناقصة (non-decreasing)** إذا كانت $\forall\, x, y \quad x \leq y \;\Rightarrow\; f(x) \leq f(y)$.

ويقال للدالة $f(x)$ إنها **غير رتيبة (anti-monotonic)** أو **غير متزايدة (non-increasing)** إذا كانت $f(x)$ - دالة رتيبة (monotonic).

ومن أمثلة الدوال الرتيبة: $x,\ x^2$ للقيم $x \geq 0$، وكذلك $\log(x)$ للقيم $x>0$، وأيضا e^x. وليس من الضروري أن تكون الدالة الرتيبة متصلة، فمن الدوال الرتيبة $\lceil x \rceil, \lfloor x \rfloor$. ومن أمثلة الدوال غير الرتيبة $1/x$ للقيم $x > 0$.

تعريف 1-8:

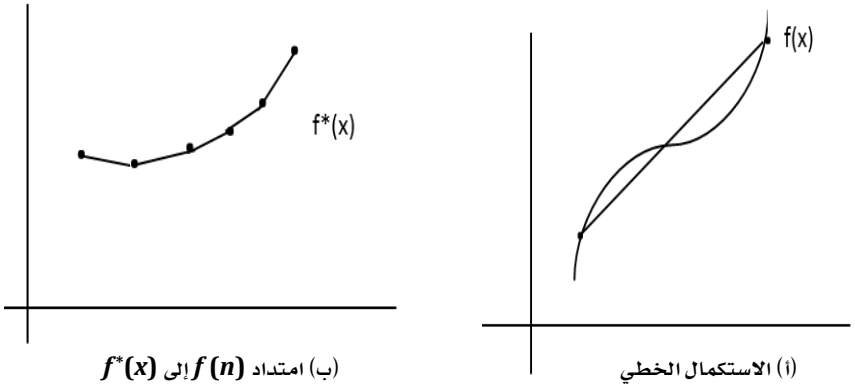
دالة الاستكمال الخطي Linear interpolation function

الاستكمال الخطي (linear interpolation) لدالة معطاة $f(x)$ بين نقطتين u, v، حيث $u < v$ هو الدالة المُعَرَّفة بالصيغة

$$L_{f,u,v}(x) = \frac{(v-x)f(u)+(x-u)f(v)}{(v-u)}$$

$$= f(u)+(x-u)\frac{f(v)-f(u)}{v-u} = f(v)-(v-x)\frac{f(v)-f(u)}{v-u} \quad (14)$$

أي أنه القطعة المستقيمة (straight-line segment) التي تصل بين $f(u)$ و $f(v)$ (انظر شكل 1–1–أ).



(أ) الاستكمال الخطي (ب) امتداد $f(n)$ إلى $f^*(x)$

شكل 1-1

توضيح خاصية التَّحَدُّب (convexity)

الدالة f مختلفة في الجزئين (أ) و (ب). في الجزء (ب) $f^*(x)$ محدبة

تعريف 1-9:

الدوال المُحَدَّبة **Convex functions**

يقال لدالة $f(x)$ إنها **مُحَدَّبة** (**convex**) إذا تحقق الشرط

في الفترة (u, v): $\forall\ u < v, \quad f(x) \le L_{f,\,u,\,v}(x)$

وبصورة غير رسمية نقول إن الدالة $f(x)$ تكون محدبة إذا لم تنحن إطلاقا لأسفل (it never curves downward).

ومن أمثلة الدوال المحدَّبة: $x,\ x^2,\ 1/x,\ e^x$. والدالة التي تظهر في شكل 1- 1- ب دالة محدبة، ولكنها ليست رتيبة، بينما الدالة في شكل 1- 1- أ دالة رتيبة، ولكنها ليست محدبة. وأيضا الدالتان $\sqrt{x}$,$\log(x)$ غير محدبتين. والتمهيدية التالية تعرض بعض الاختبارات العملية للتحدب. ومن السهل أن نرى –ومن الممكن أن نثبت– أن أي دالة غير متصلة لا يمكن أن تكون محدبة. والتمهيدية 1–3 التالية تقرر أنه يكفي لاختبار التحدب أن نأخذ في الاعتبار عدة نقاط متساوية المسافات فيما بينها، مما يُبَسِّط الأمور بدرجة كبيرة.

تمهيدية 1–3:

(1) نفرض أن $f(x)$ دالة متصلة مُعَرَّفة على الأعداد الحقيقية. الدالة $f(x)$ تكون مُحَدَّبة إذا وفقط إذا تحقق الشرط التالي لأي نقطتين x, y:

$$f\left(\frac{1}{2}(x+y)\right) \le \frac{1}{2}\left(f(x)+f(y)\right)$$

(2) أي دالة $f(n)$ مُعَرَّفة على الأعداد الصحيحة تكون مُحَدَّبة إذا وفقط إذا تحقق الشرط التالي لأي قيم $n, n+1, n+2$:

$$f(n+1) \leq \frac{1}{2}\left(f(n) + f(n+2)\right)$$

التمهيدية التالية تلخص عدة خواص مفيدة عن الرتابة (monotonicity) والتحدب (convexity). وهي تنص على أن الدوال المُعَرَّفة فقط على الأعداد الصحيحة يمكن أن تُوَسَّع / تُمَدَّد (extended) إلى الأعداد الحقيقية بالاستكمال الخَطّي (linear interpolation)، مع الاحتفاظ (preserving) بخاصيتي الرتابة والتحدب. كما تشتمل هذه التمهيدية على بعض الخواص المتعلقة بالمشتقات (derivatives).

تمهيدية 1-4:

1) نفرض أن $f(n)$ دالة مُعَرَّفة فقط على الأعداد الصحيحة. ونفرض أن $f^*(x)$ هي امتداد (extension) الدالة f على الأعداد الحقيقية (real) بالاستكمال الخَطِّي بين الأعداد الصحيحة المتعاقبة (consecutive integers) [انظر شكل 1- ب].

أ) الدالة $f(n)$ تكون رتيبة (monotonic) إذا وفقط إذا كانت الدالة $f^*(x)$ رتيبة.

ب) الدالة $f(n)$ تكون مُحَدَّبة (convex) إذا وفقط إذا كانت الدالة $f^*(x)$ مُحَدَّبة.

2) إذا كانت المشتقة الأولى للدالة $f(x)$ موجودة وغير سالبة، فإن الدالة $f(x)$ تكون رتيبة.

3) إذا كانت المشتقة الأولى للدالة $f(x)$ موجودة ورتيبة، فإن الدالة $f(x)$ تكون محدَّبة.

4) إذا كانت المشتقة الثانية للدالة $f(x)$ موجودة وغير سالبة، فإن الدالة $f(x)$ تكون محدَّبة. [هذا ينتج من الجزئين السابقين 2، 3].

سابعا: التجميعات باستخدام التكامل Summations Using Integration

هناك عدة تجميعات تظهر كثيرا عند تحليل الخوارزميات ويمكن تقريبها (approximated) [أو جعلها محدودة من أعلى أو من أسفل(bounded from above or below)] باستخدام التكامل. وسنعرض أولا بعض صيغ التكامل المفيدة:

$$\int_0^n x^k\,dx=\frac{1}{k+1}n^{k+1},\qquad \int_0^n e^{ax}\,dx=\frac{1}{a}\left(e^{an}-1\right),$$

$$\int_1^n x^k\ln(x)dx=\frac{1}{k+1}n^{k+1}\ln(n)-\frac{1}{(k+1)^2}n^{k+1}+\frac{1}{(k+1)^2} \tag{15}$$

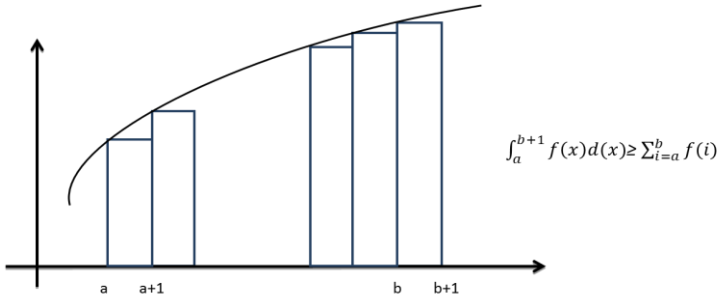
إذا كانت الدالة $f(x)$ رتيبة (أو غير متناقصة) (monotonic/non-decreasing) فإن

$$\int_{a-1}^{b} f(x)dx\le\sum_{i=a}^{b} f(i)\le\int_a^{b+1} f(x)dx \tag{16}$$

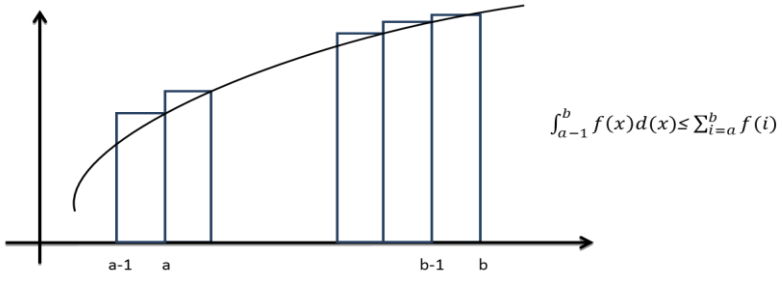
وبالمثل إذا كانت الدالة $f(x)$ غير رتيبة (أو غير متزايدة) (antimonotonic/ non-increasing) فإن

$$\int_a^{b+1} f(x)dx\le\sum_{i=a}^{b} f(i)\le\int_{a-1}^{b} f(x)dx \tag{17}$$

وهذا الوضع بالنسبة للدالة الرتيبة $f(x)$ يوضحه شكل 1–2.



(أ) التقريب العلوي (Over approximation)



(ب) التقريب السفلي (Under approximation)

شكل 1–2 : تقريب مجموع قيم دالة رتيبة
Approximating a sum of values of a monotonic function

وفيما يلي مثالان سنستخدمهما بإذن الله فيما بعد في هذا الكتاب.

مثال 1–1:

تقدير المجموع (An estimate for) $\sum_{i=1}^{n} \frac{1}{i}$

$$\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_1^n \frac{dx}{x} = 1 + \ln x]_1^n = 1 + \ln n - \ln 1 = \ln(n) + 1$$

وذلك باستخدام المعادلة (17). ولاحظ أننا جَزَّأْنا / شققنا / فَصلْنا

(split off) الحد الأول (first term) في المجموع، وطَبَّقنا تقريب التكامل(integral approximation) على الجزء الباقي (rest)، وذلك لنتحاشى القسمة على صفر عند النهاية السفلى (lower limit) للتكامل. وبالمثل

$$\sum_{i=1}^{n} \frac{1}{i} \geq 1n(n+1)$$

[انظر المعادلة (11) لتقريب أدق (a closer approximation)]

مثال 1–2:

حد سفلي للمجموع $\sum_{i=1}^{n} \lg i$ (A lower bound for)

$$\sum_{i=1}^{n} \lg i = 0 + \sum_{i=2}^{n} \lg i \geq \int_{1}^{n} \lg x\, dx$$

وذلك باستخدام المعادلة (16) [انظر شكل 1–2–ب]. والآن

$$\begin{aligned}\int_{1}^{n} \lg x\, dx &= \int_{1}^{n} (\lg e)\ln x\, dx = (\lg e)\int_{1}^{n} \ln x\, dx \\ &= (\lg e)(x\ln x - x)]_{1}^{n} = (\lg e)(n\ln n - n + 1) \\ &= n\lg n - n\lg e + \lg e \geq n\ \lg n - n\ \lg e\end{aligned}$$

ونظرا لأن $\lg e < 1.443$، فلذلك

$$\sum_{i=1}^{n} 1g\, i \geq n\ 1g\, n - 1.443n \qquad (18)$$

وباستخدام أفكار المثال السابق، ولكن برياضيات أكثر دِقَّة، يمكننا استنتاج/ اشتقاق (derivation) قانون / صيغة "سترلنج" (Stirling's formula)، مما يعطينا حدودا للمقدار $n!$:

$$\left(\frac{n}{e}\right)^n \sqrt{2\pi\ n} < n! < \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1+\frac{1}{11\ n}\right) \quad \textbf{for}\ n \geq 1 \qquad (19)$$

ثامنا: **المتباينات** Inequalities

فيما يلي بعض قواعد الجمع بين المتباينات (rules for combining inequalities)، والتي تفيدنا في مواطن كثيرة.

التَّعَدِّي Transitivity	الجَمْع Addition	القياس الموجب Positive Scaling
$A \leq B \ \& \ B \leq C$ $\Rightarrow A \leq C$	$A \leq B \ \& \ C \leq D$ $\Rightarrow A + C \leq B + \mathrm{D}$	$A \leq B \ \& \ \alpha > 0$ $\Rightarrow \alpha A \leq \alpha B$

(20)

تاسعا: **المنطق** Logic

(أ) **متطابقات منطقية** (logical identities)

$$A \Rightarrow B \equiv \neg A \vee B \qquad (21)$$

قانونا "دي مورجان" **(De Morgan's laws)**

$$\neg (A \wedge B) \equiv \neg A \vee \neg B, \qquad (22)$$

$$\neg (A \vee B) \equiv \neg A \wedge \neg B. \qquad (23)$$

$$\forall\ x\ A(x) \equiv \neg \exists\ x\ (\neg A(x)), \qquad (24)$$

$$\exists\ x\ \ A(x) \equiv \neg \forall\ x\ (\neg A(x)). \qquad (25)$$

$$\begin{aligned} \neg (\forall\ x\ (A(x) \Rightarrow B(x))) &\equiv \exists\ x \neg (A(x) \Rightarrow B(x)) \\ &\equiv \exists\ x \neg (\neg A(x) \vee B(x)) \\ &\equiv \exists\ x\ (A(x) \wedge \neg B(x)). \qquad (26) \end{aligned}$$

المكافئ العكسي **(Contrapositive)**

$$A \Rightarrow B \equiv (\neg B) \Rightarrow (\neg A). \qquad (27)$$

البرهان بالتناقض **(Proof by Contradiction)**

$$A \Rightarrow B \equiv (A \wedge \neg B) \Rightarrow B. \qquad (28)$$

(ب) قواعد استدلال (Rules of Inference)

إذا فرضنا أن ***A, B, C*** عبارات منطقية (logical statements)، فإن

(i) **قاعدة الفَصْل (detachment rule)**، ويطلق عليها أيضا قاعدة "**مودَس بونِنز**" (modus ponens):

$$[B \,\&\, (B \Rightarrow C)] \Rightarrow C \qquad (29)$$

(ii) **قاعدة القياس المنطقي** **(syllogism)**

$$[(A \Rightarrow B) \,\&\, (B \Rightarrow C)] \;\Rightarrow\; (A \Rightarrow C) \qquad (30)$$

(iii) **قاعدة الحالات** **(rule of cases)**

$$[(B \Rightarrow C) \,\&\, (\neg B \Rightarrow C)] \;\Rightarrow\; C \qquad (31)$$

❖ ❖ ❖

الفصل الثانى:

التحليل الأساسى للخوارزميات .

1) **صحة الخوارزمية** .

2) **كمية الجُهْد/ العمل المبذول** .

تحليل الحالة المتوسطة وأسوأ حالة

3) **سعة الحيِّز المستخدم**

4) **البساطة والوضوح**

5) **الأمثلية** .

الحدود السُّفْلَى ودرجات تعقيد المسائل .

التنفيذ والبرمجة

تصنيف الدوال بناءً على معدلات نموها المقارِبة .

أهمية الرتبة المقارِبة

خواص المجموعات O, Ω, Θ

البحث في منظومة مرتبة

البحث الثنائى .

تحليل أسوأ حالة لخوارزمية البحث الثنائى

تحليل السلوك المتوسط .

الأمثلية

- **تمرينات الفصل الثانى**

الفصل الثاني

التحليل الأساسي للخوارزميات

Basic Analysis of Algorithms

نقوم عادة بتحليل الخوارزميات بهدف تحسين أدائها إن أمكن، وللاختيار فيما بين عدة خوارزميات متاحة أمامنا لحل مسألة معينة. ونستخدم المعايير (criteria) الأساسية التالية في دراستنا التحليلية للخوارزميات:

1) صحة الخوارزمية (Correctness of an algorithm).
2) كمية الجُهْد / العمل المبذول (Amount of work done).
3) سعة الحَيِّز المستخدَم (Amount of space used).
4) البساطة والوضوح (Simplicity and clarity).
5) الأمثلية (Optimality).

وفيما يلي نتناول هذه المعايير، ونعطِي عدة أمثلة لتطبيقها. وعند مناقشة معيار أمثلية الخوارزميات سنعرض بإذن الله طرقا لإيجاد حدود دنيا /سفلى (lower bounds) لدرجة تعقيد المسائل (complexity of problems).

(1) صحة الخوارزمية (Correctness of an algorithm)

أي خوارزمية يكون لها عادةً مُدْخلات (inputs) لها خصائص

(characteristics) نُطْلِـق عليهـا "الشـروط السـابقة" (preconditions)، ومخرجات (outputs) مقابلة لهذه المدخلات نطلق عليها "الشروط اللاحقة" (postconditions). ولإثبات صحة الخوارزمية علينا أن نثبت أنه إذا تحققت الشروط السابقة للمدخلات المعطاة فإن الخوارزمية ستُنتج (produce) النتائج المطلوبـة، أي أن الشـروط اللاحقـة سـتكون صـحيحة (true) عنـدما تتوقـف (terminates) الخوارزمية.

وهنـاك نقطتـان بالنسـبة لأي خوارزميـة: طريقـة الحـل (solution method)، والتنفيـذ (implementation)، ونقصـد بـه متتابعـة التعليمـات (sequence of instructions) لتنفيـذ / لتطبيـق هـذه الطريقـة. وإثبـات صحة طريقة الحل والعلاقات والصيغ (formulas) التي تستخدمها قد يكون سهلا وبسيطا، وقد يتطلب عدة فرضيات (lemmas) ونظريات (theorems) عـن الأشـياء (objects) الـتي تعمـل عليهـا الخوارزميـة [كالرسـوم البيانيـة (graphs)، والتبـديلات (permutations)، والمصـفوفات (matrices)]. فمـثلا إثبات صحة طريقة "جاوس" للحذف لحل نظم المعادلات الخطية يعتمد على عدد من نظريات الجبر الخطي (linear algebra). وصحة بعض خوارزميات هـذا الكتـاب ليسـت واضـحة (obvious)، وإنمـا تحتـاج إلى برهنتهـا بـبعض النظريات.

و بعـد تحديـد طريقـة الحـل نقـوم بتنفيـذها ببرنـامج. فـإن كانـت الخوارزمية قصيرة ومباشرة (straightforward) فإننا عادةً نستخدم طرقا غير رسميـة / غـير شـكلية (informal) لإقنـاع أنفسـنا بـأن الأجـزاء المختلفـة مـن الخوارزميـة تـؤدي فعلا مـا نطلبـه منهـا. وقد نختـبر (check) بعض التفاصيل بدقة [كالقيم الابتدائية والنهائية لعدادات العُرَى (loop counters) مثلا]،

ونحاكي بأيدينا (hand–simulate) تنفيذ الخوارزمية بعدة أمثلة بسيطة. وكل هذا لا يثبت صحة الخوارزمية، ولكن هذه الطرق غير الرسمية قد تكفي للبرامج الصغيرة. وقد نستخدم طرقا رسمية أكثر - مثل طريقة "لا متغيرات العروة" (loop invariants) - للتحقق من صحة أجزاء من البرامج. والفصل الرابع سيتناول بإذن الله هذه النقطة ببعض التفصيل.

وأما بالنسبة للبرامج الطويلة والمعقدة فلإثبات صحتها يمكننا محاولة تقسيم البرنامج إلى أجزاء / برامج فرعية / وحدات مستقلة مكمِّلة (modules) أصغر، وإثبات أنه إذا كانت جميع هذه الأجزاء / الوحدات المكمِّلة تؤدي فعلا وظائفها المطلوبة منها بصورة مضبوطة (properly) فإن البرنامج كاملا (whole program) يكون صحيحا (correct)، ثم إثبات أن كلا من هذه الأجزاء / الوحدات المكمِّلة صحيح (correct). وهذا الأسلوب لإثبات صحة البرامج الكاملة يكون أسهل وأيسر في التطبيق إن استطعنا كتابة الخوارزميات والبرامج كأجزاء / برامج فرعية / وحدات مكمِّلة مستقلة بدرجة كبيرة (largely independent) عن بعضها البعض، وأمكن التحقق من صحتها منفصلة عن بعضها البعض (verified separately). ويُعَدُّ هذا الأسلوب من أهم مزايا البرمجة المبْنِيَّة / البرمجة المُهَيْكَلَة / البرمجة بالفَصْل المُكَمِّل(structured / modular programming). ومعظم الخوارزميات التي سنعرضها بإذن الله في هذا الكتاب عبارة عن الأجزاء الصغيرة (small segments) التي يمكننا منها بناء (building) / تكوين البرامج الكبيرة. وبالتالي فلن نواجه أو نتعرض لصعوبات إثباتات صحة البرامج أو الخوارزميات الطويلة جدا. وفي الفصل الرابع سنعرض بإذن الله بعض الطرق (techniques) التي تساعد في تيسير برهنة صحة البرامج والخوارزميات.

(2) **كمية الجُهْد / العمل المبذول** (Amount of work done)

نحتاج إلى اختيار مقياس (a measure) لمقدار الجهد / العمل الذي تقوم به أي خوارزمية، وذلك حتى يساعدنا هذا المقياس في المقارنة بين خوارزميتين لحل مسألة معينة - أي لحل المسألة نفسها - كي نستطيع تحديد أي الخوارزميتين أعلى كفاءة(more efficient) من الأخرى. وقد يفكِّر البعض في اتخاذ الوقت الفعلي لتنفيذ الخوارزمية (actual execution time) مقياسا، ولكننا نرى أنه لا يصلح مقياسا لأنه يعتمد على الحاسوب المستخدَم، وتتغير قيمته بتغير الحاسوب، ولا نريد أن نُنشئ نظرية خاصة بحاسوب معيَّن. وكذلك قد يفكِّر البعض الآخر في جَعْل المقياس هو عدد (count) جميع التعليمات (instructions) أو العبارات (statements) التي يقوم البرنامج بتنفيذها. ولكن هذا المقياس عليه كذلك العديد من المآخذ التي نأخذها على مقياس وقت التنفيذ (execution time)، إذ أنه يعتمد بدرجة كبيرة على لغة البرمجة المستخدَمة وعلى أسلوب الشخص المبرمج، ويتطلب كذلك بذل وقت وجهد لكتابة وتصحيح برامج لكل خوارزمية من الخوارزميات قيد الدراسة. فنحن نريد مقياسا للجهد المبذول يخبرنا شيئا عن **كفاءة الطريقة** (efficiency of the method) التي تستخدمها الخوارزمية مستقلة عن كل من (independent of) [أي غير معتمدة على أي من] الحاسوب المستخدَم، ولغة البرمجة المستخدَمة، والشخص المبرمِج، وكذلك لا تعتمد على تفاصيل التنفيذ الكثيرة (many implementation details)، وعمليات الإضافيات أو التَّهْيئَة [overhead or "bookkeeping" operations] كزيادة مؤشرات / دليل / فهارس العُرَى (incrementing loop indexes)، وحساب مؤشرات / فهارس المنظومات (computing array

(setting indexes)، وضبط المؤشرات في بنى المعطيات / هياكل البيانات (measure of pointers in data structures). فمقياس الشُّغل المبذول (work) يجب أن يكون دقيقا بدرجة كافية (precise enough)، وعامًّا بدرجة كافية (general enough) للوصول إلى نظرية تتسم بالثراء (a rich theory) لإمكانية تطبيقها على الكثير من الخوارزميات والتطبيقات.

و قد تتكون خوارزمية بسيطة من بعض تعليمات إبداء (أي إعطاء قيم ابتدائية) (initialization instructions) وعروة (a loop). وعدد الدورات / المسارات / الاجتيازات / التمريرات (number of passes) عبر جسم العروة (body of the loop) يُعَد دلالة معقولة ومقبولة إلى حد كبير على مقدار الجهد المبذول بمثل هذه الخوارزمية. وبالطبع قد يكون الشُّغل المبذول في دورةٍ / تمريرةٍ ما (a pass) في إحدى العُرَى أكبر بكثير من ذلك المبذول في دورةٍ / تمريرةٍ أخرى، وكذلك قد تحتوي خوارزميةٌ ما على أجسام / بنيات عُرَى (loop bodies) أطول من تلك التي تحتوي عليها خوارزمية أخرى، ولكننا سنصل أخيرا إلى مقياس جيد للشغل المبذول. فرغم أن بعض العُرَى قد تحتوي مثلا على خمس خطوات، بينما تحتوي أخرى على تسع خطوات، إلا أنه بالنسبة للمدخلات الكبيرة (large inputs) فإن عدد الدورات / التمريرات (passes) عَبْر العُرَى سيكون عموما كبيرا مقارنةً بأحجام العُرَى (loop sizes). وبالتالي فإن حساب عدد الدورات / التمريرات(counting the passes) عَبْر جميع العُرَى (through all the loops) في الخوارزمية يُعَدُّ فكرةً جيدة.

وفي حالات كثيرة يمكننا لتحليل أي خوارزمية أن نحدد عملية معينة (a particular operation) أساسية بالنسبة للمسألة قيد الدراسة (أو

بالنسبة لأنواع الخوارزميات قيد الدراسة)، ونهمل الإبداء (initialization) والتحكم في العروة (loop control)، والإضافيات الأخرى، ونحسب فقط عدد العمليات المختارة أو الأساسية التي تقوم بها الخوارزمية. و في كثير من الخوارزميات تكون هناك عملية واحدة بالضبط من هذه العمليات يتم تنفيذها في كل تمريرة عبر العُرَى الرئيسية في الخوارزمية، وبالتالي فهذا المقياس مشابه للمقياس المذكور في الفقرة السابقة.

و فيما يلي بعض الأمثلة لاختيارات مناسبة لعمليات أساسية في عدة مسائل.

المسألة Problem	العملية Operation
• إيجاد x في منظومة أسماء	• مقارنة x بعنصر في المنظومة
• ضرب مصفوفتي أعداد حقيقية	• ضرب عددين حقيقيين (أو ضرب وجمع أعداد حقيقية)
• ترتيب / فرز (sorting) عناصر منظومة أعداد	• المقارنة بين عنصري منظومة
• اجتياز (Traversing) شجرة ثنائية	• اجتياز حرف (Traversing an edge)
• أي إجراء غير تكرارى (any noniterative procedure) بما في ذلك الارتدادي (recursive)	• استدعاء إجراء (procedure invocation)

و طالما أنه قد تم اختيار العملية الأساسية (أو العمليات الأساسية) بطريقة جيدة، وكان العدد الإجمالي للعمليات التي يتم تنفيذها متناسبا تقريبا (roughly proportional) مع عدد العمليات الأساسية، فإنه يكون لدينا مقياس جيد (a good measure) للشغل المبذول بالخوارزمية، ومعيار

(criterion) جيد للمقارنة بين عدة خوارزميات. وهذا هو المقياس الذي سنستخدمه عادة بإذن الله في هذا الكتاب. وفيما يلي بعض الملاحظات:

1) في بعض الحالات قد يكون اهتمامنا مركزًا على العملية الأساسية والتي قد تكون عالية التكلفة مقارنة مع العمليات الأخرى، أو قد تكون ذات أهمية نظرية (Theoretical interest).

2) عادة نهتم بمعدل زيادة الوقت (rate of growth of time) اللازم لتنفيذ الخوارزمية مع زيادة / كبر حجم المدخلات. وطالما أن العدد الإجمالي للعمليات متناسب تقريبًا مع عدد العمليات الأساسية، فإن حساب عدد العمليات الأساسية يمكن أن يعطينا فكرة واضحة عن إمكانية استخدام الخوارزمية للمدخلات الكبيرة.

3) هذا الاختيار لمقياس الشغل المبذول يسمح بقدر كبير من المرونة (flexibility). فرغم أننا سنحاول غالبًا اختيار عملية معينة - أو على الأكثر عمليتين معينتين - للعد، فإنه يمكننا أن نشمل بعض العمليات الإضافية(include some overhead operations)، وفي ناحية قصوى (in the extreme) يمكننا اختيار مجموعة تعليمات الآلة (set of machine instructions) لحاسوب معيَّن باعتبارها العمليات الأساسية (basic operations). و في الناحية القصوى المقابلة يمكننا اعتبار "تمريرة واحدة عبر عروة" (one pass through a loop) العملية الأساسية. وهكذا بتغيير اختيار العمليات الأساسية يمكننا تغيير درجة الدِّقة (precision) والتجريد (abstraction) في تحليلنا لتناسب احتياجاتنا.

وهنا تُثار بعض التساؤلات: ماذا إذا اخترنا عملية أساسية لمسألةٍ ما، ثم وجدنا أن العدد الإجمالي للعمليات التي تقوم بها الخوارزمية ليس متناسبا مع

عدد العمليات الأساسية؟ وماذا إذا كان أكبر ؟ وفي الحالة المتطرفة قد نختار عملية أساسية لمسألة معينة ثم نكتشف أن بعض الخوارزميات لهذه المسألة تستخدم طرقا مختلفة لا تقوم بتنفيذ أيٍّ من العمليات التي نقوم بعَدِّها. في مثل هذه الحالات أمامنا خياران: إما أن نلغي تركيزنا على العملية المعينة ونلجأ إلى عَدِّ التمريرات (counting passes) عبر العروات، أو - إن كان اهتمامنا فعليا بهذه العملية المعينة المختارة - يمكننا أن نحصر دراستنا في طبقة معينة من الخوارزميات (a particular class of algorithms) تكون هذه العملية المختارة مناسبة لها. وأما الخوارزميات التي تستخدم طرقا أخرى يناسبها اختيار مختلف للعملية الأساسية فيمكن دراستها بصورة منفصلة. وعادةً تُعَرَّف أي طبقة من الخوارزميات لمسألةٍ ما بتحديد العمليات التي يمكن أن تُطَبَّق على البيانات (data).

و سنستخدم بإذن الله المصطلح **"درجة تعقيد الخوارزمية" (complexity of the algorithm)** لتعنى: مقدار العمل أو الشُّغل أو الجهد المبذول بالخوارزمية (amount of work done by the algorithm) مقاساً **بمقياسِ تعقيدٍ** محددٍ ما (some specified **complexity measure**)، والذي سيكون في كثير من أمثلتنا هو عدد العمليات الأساسية المعينة التي تم إنجازها. ويلاحَظ أن مصطلح "التعقيد" (complexity) هنا - بناءً على المعنى المذكور - لا علاقة له بمفهوم أن الخوارزمية معقَّدة (complicated) أو صعبة وشائكة (tricky)، فمن الممكن أن تكون لدينا خوارزمية معقدة جدا (very complicated algorithm) بينما درجة تعقيدها منخفضة (low complexity). وفي هذا الكتاب سنستخدم تبادليا (interchangeably) بإذن الله التعابير: "درجة التعقيد" (complexity)، و "مقدار الشُّغل أو الجهد المبذول" (amount of work done)، و "عدد العمليات الأساسية التي تم إجراؤها" (number of basic operations done) لتعنى الشئ نفسه.

تحليل الحالة المتوسطة و أسوأ حالة و أحسن حالة

Average, Worst, and Best-Case Analysis

نلاحظ أولا أن قدر الشُّغل المبذول يعتمد عادة على حجم المدخلات. فمثلا الترتيب الأبجدي لأسماء منظومة مكونة من 1000 اسم يتطلب عادةً عمليات أكثر من الترتيب الأبجدي لأسماء منظومة مكونة من 10 أسماء فقط باستخدام الخوارزمية نفسها. وحل نظام خطي مكون من 12 معادلة في 12 مجهول يستغرق جهدا أكبر من حل نظام خطي مكون من معادلتين في مجهولين. ونلاحظ ثانيا أنه حتى لو اعتبرنا أن جميع المدخلات لها الحجم نفسه، فإن عدد العمليات التي تقوم بإجرائها خوارزميةٌ ما قد يعتمد على المدخلات نفسها. فقد تقوم خوارزميةٌ ما لترتيب أسماء منظومة ترتيبا أبجديا بجهد بسيط جدا إذا كان هناك عدد قليل من الأسماء غير المتفقة مع الترتيب الأبجدي، بينما قد تقوم بجهد أكبر بكثير مع منظومةٍ أسماؤها مبعثرة بدرجة كبيرة. وكذلك قد لا يتطلب حل نظام خطي مكوَّن من 12 معادلة جهدا كبيرا إذا كانت معظم المعاملات أصفارا.

الملاحظة الأولى تشير إلى أننا نحتاج إلى مقياس لحجم المدخلات لمسألةٍ ما. وعادةً يكون من السهل اختيار مقياس معقول للحجم / للسعة. وفيما يلي بعض الأمثلة.

المسألة Problem	حجم المدخلات Size of input
• إيجاد x في منظومة أسماء	• عدد الأسماء في المنظومة
• ضرب مصفوفتين	• أبعاد (dimensions) المصفوفتين
• ترتيب / فرز (sorting) منظومة أعداد	• عدد عناصر المنظومة
• اجتياز شجرة ثنائية	• عدد العُقَد (nodes) في الشجرة
• حل نظام معادلات خطية	• عدد المعادلات أو عدد المجاهيل أو كلاهما
• حل مسألة تتعلق بمخطط بياني (a graph)	• عدد العُقد أو عدد الأحرف (edges) في المخطط البياني، أو كلاهما

وحتـى لـو كـان حجـم المـدخلات ثابتـا - يسـاوي n مـثلاً - فـإن عـدد العمليات التي يتم تنفيذها قد يعتمد على المدخلات بعينها. فكيف يمكننا إذن التعـبير عـن نتـائج تحليـل أي خوارزميـة؟ في معظـم الأحـوال نَصِـف سـلوك أي خوارزمية بإعطاء درجة تعقيدها في أسوأ حالة (worst case complexity)، كما نوضح ذلك فيما يلي:

تعريف 2-1 : درجة التعقيد في أسوأ حالة Worst-case complexity

نفـرض أن D_n هـي مجموعـة المـدخلات ذوات السـعة n (inputs of size) للمسألة قيد الاعتبار، وأن I عنصر في المجموعة D_n. ونفرض أن $t(I)$ هي عدد العمليـات الأساسـية الـتي تجريهـا الخوارزميـة علـى المـدخلات I. نُعَـرِّف الدالـة $W(n)$ كما يلي:

$$W(n) = \max \{t(I) \mid I \in D_n\}$$

الدالة $W(n)$ يُطلق عليها "**درجة تعقيد الخوارزمية في أسوأ حالة**" (worst-case complexity of the algorithm). $W(n)$ هي أكبر عدد من العمليات الأساسية التي تجريها الخوارزمية على أي مدخلات حجمها n.

ودرجة التعقيد في أسوأ حالة تعطينا حدًّا أعلى (an upper bound) للجهد المبذول بالخوارزمية. وتحليل أسوأ حالة (worst-case analysis) يمكن أن يُستخدَم ليساعدنا في الوصول إلى تقديرٍ (an estimate) لحدٍّ زمنيٍّ (a time limit) لتنفيذٍ معيَّن لخوارزميةٍ ما. وسنقوم بإذن الله بإجراء تحليل أسوأ حالة بالنسبة لمعظم خوارزميات هذا الكتاب. وحينما نشير إلى قدر الجهد المبذول بأي خوارزمية، فإننا سنعنى بذلك قدر الجهد المبذول في أسوأ حالة، ما لم يُنص على غير ذلك.

تعريف 2-2: درجة التعقيد في أحسن حالة Best-case complexity

نُعَرِّف الدالة ***B(n)*** كما يلي:

$$B(n) = \min \{t(I) \mid I \in D_n\}$$

الدالة ***B(n)*** يُطلق عليها "**درجة تعقيد الخوارزمية في أحسن حالة**" (best-case complexity of the algorithm). ***B(n)*** هي أصغر عدد من العمليات الأساسية التي تجريها الخوارزمية على أي مدخلات حجمها ***n***. و هذه الحالة ليست لها أهمية كبيرة من الناحية العملية.

وقد يبدو من الطبيعي والأكثر فائدة أن نَصِفَ سلوك أي خوارزمية بذكر مقدار الجهد المبذول في المتوسط (on the average)، أي بحساب عدد العمليات التي يتم إجراؤها على كل من المدخلات التي سعتها n، ثم أَخْذ المتوسط. ومن الناحية العملية قد تحدث بعض المدخلات مرات عديدة أكثر بكثير من مدخلات أخرى، ولذلك يكون من الأنسب حساب المتوسط الموزون (weighted average) لا المتوسط الحسابي (arithmetic average).

تعريف 2–3: درجة التعقيد المتوسطة Average complexity

نفرض أن ***Pr(I)*** هي احتمال حدوث المدخلات ***I***. يُعَرَّف السلوك المتوسط للخوارزمية (average behavior of the algorithm) بالعلاقة

$$A(n) = \sum_{I \in D_n} Pr(I)\, t(I)$$

ونقوم بحساب قيمة t(I) بتحليل الخوارزمية، ولكن الاحتمال Pr(I) لا يمكن حسابه تحليليا (analytically). وهذه الدالة Pr(I) تُحْسَب قيمتها بالخبرة (experience) أو / و المعلومات الخاصة عن التطبيق (application) الذي ستُستخدم له الخوارزمية، أو بفرض بعض الفروض التبسيطية

(simplifying assumptions) [مثلا كفرض أن جميع المدخلات ذوات السعة n متساوية الاحتمالات فيما بينها (equally likely to occur)]. وإذا كانت الدالة Pr(I) دالة معقدة (complicated)، فإن حساب السلوك المتوسط (average behavior) يكون صعبا. وبالطبع أيضا إذا كانت الدالة Pr(I) تعتمد على تطبيق معيَّن / خاص للخوارزمية (a particular application of the algorithm) فإن دالة السلوك المتوسط A ستَصِف السلوك المتوسط للخوارزمية لهذا التطبيق فقط.

وفيما يلي بعض الأمثلة التي توضح كيفية إيجاد كل من درجة التعقيد المتوسطة ودرجة التعقيد في أسوأ حالة.

مثال2-1 : البحث في منظومة غير مرتبة

Search in an unordered array

المسألة (Problem): نفرض أن E منظومة تحتوي على n عنصر [يطلَق عليها "مفاتيح" (keys)]: E[0], …, E[n - 1] ليست بترتيب معين. المطلوب إيجاد مؤشر (index) قيمة معينة / مفتاح معين (a specified key) K إن كان K موجودًا في المنظومة، أو إعادة 1- إن لم يكن K في المنظومة [ملاحظة: سندرس لاحقاً بإذن الله الحالة التي تكون فيها عناصر المنظومة مُرَتَّبَة].

طريقة الحل: نقارن القيمة K بكل عنصر على الترتيب حتى نجد توافقا (a match) / تطابقًا ، أو حتى نستنفذ جميع عناصر المنظومة (array is exhausted). وإن لم تكن K بالمنظومة، فإن الخوارزمية تعيد القيمة / الإجابة 1- .

وهناك طائفة كبيرة من الإجراءات (large class of procedures)

المشابهة لهذا الإجراء المذكور، ونطلق عليها "برامج البحث الروتينية المُعَمَّمَة" (generalized searching routines). وغالبًا ما تكون في صورة برامج فرعية (subroutines) لإجراءات أكثر تعقيدًا.

تعريف 2-4: إجراء البحث الروتيني المُعَمَّم

Generalized searching routine

إجراء البحث الروتيني المعمم هو إجراء يقوم بتشغيل / بمعالجة (processing) كمية غير محددة من البيانات (indefinite amount of data) إلى أن يستنفذ هذه البيانات (exhausts the data) أو إلى أن يحقق هدفه (achieves its goal). وفيما يلي المخطط العام عالي المستوى (high level outline) لهذا الإجراء:

إذا لم تكن هناك بيانات أخرى للنظر فيها:

if there is no more data to examine:
    fail. — يفشل الإجراء
else — وإلا
    examine one datum. — انظر في وحدة بيانات أخرى
    if this datum is what we want: — **فإن** كانت هي ما نريد
        succeed. — ينجح الإجراء
    else — **وما عدا ذلك**
        keep searching in remaining data. — استمر في البحث في البيانات المتبقية

وهذا الإجراء يوصف بأنه بحث مُعَمَّم (generalized searching) لأن الإجراء يقوم غالبا بتنفيذ بعض العمليات البسيطة الأخرى أثناء قيامه بالبحث، مثل تحريك عناصر بيانات، وإضافة عناصر إلى (أو حذف عناصر من) بنية معطيات (a data structure)، وغيرها.

خوارزمية 2-1 : البحث التتابعي (غير المُرتَّب)

Sequential Search (unordered)

المدخلات (Input): ***E, n, K***، حيث:

E: منظومة مُكَوَّنة من ***n*** عنصر، وهذه العناصر مؤشَّرة (indexed): **0**, …, ***n*-1**.

K: القيمة التي نبحث عنها (the item sought). وللتبسيط نفرض أن ***K*** وعناصر ***E*** جميعها أعداد صحيحة.

المخرجات (Output) : **ans** ، وهي موقع (location) القيمة ***K*** في المنظومة E [1– إذا لم نجد ***K***].

```
int seqSearch ( int [ ] E, int n, int K )
1.  int ans, index;
2.  ans = –1;   // Assume failure.
3.  for (index = 0; index < n; index ++)
4.      if (K = = E[index])
5.          ans = index;   // Success!
6.          break;
    // continue loop.
7.  return ans;
```

العملية الأساسية (Basic Operation):

مقارنة ***K*** مع عنصر منظومة

تحليل أسوأ حالة (Worst–Case Analysis):

من الواضح أن $W(n) = n$. وأسوأ حالات تحدث عندما تظهر K فقط في آخر موضع في المنظومة، وعندما لا تكون K موجودة في المنظومة. وفي كل من هاتين الحالتين تتم مقارنة K مع جميع عناصر المنظومة والتي عددها n.

تحليل السلوك المتوسط (Average-Behavior Analysis):

سنقوم أولا بذكر بعض الافتراضات التبسيطية (simplifying assumptions) لتقديم مثال سهل، ثم نذكر بعد ذلك افتراضات مختلفة لتقديم تحليل أعقد قليلا (a slightly more complicated analysis). ونفرض أن عناصر المنظومة جميعها متباينة (distinct)، وأنه إذا كانت K في المنظومة فإن احتمالات وجودها في المواقع المختلفة في المنظومة جميعها متساوية.

i) في الحالة الأولى نفرض أن K موجودة في المنظومة، ونرمز لهذا الحدث بـ "$succ$". ويمكن تصنيف المدخلات بناءً على الموقع الذي ستظهر فيه K في المنظومة، ولهذا نأخذ في الاعتبار مدخلات عددها n. ونفرض أن I_i تمثل الحدث أن K تظهر في الموضع i في المنظومة، وذلك للقيم $0 \le i < n$. ثم نفرض أن $t(I)$ هي عدد المقارنات التي يتم إجراؤها [أي عدد المرات التي نختبر فيها الشرط المذكور في السطر رقم 4 في الخوارزمية] على المدخلات I. من الواضح أنه للقيم $0 \le i < n$، فإن $t(I_i) = i + 1$. و نفرض أيضا تساوي احتمال (equally likely) وجود K في موضع ما: $Pr(I_i \mid succ) = \frac{1}{n}$. وهكذا نجد أن

$$A_{succ}(n) = \sum_{i=0}^{n-1} Pr(I_i \mid succ)\, t(I_i)$$

$$= \sum_{i=0}^{n-1} \left(\frac{1}{n}\right) (i+1) = \left(\frac{1}{n}\right) \frac{n(n+1)}{2} = \frac{n+1}{2}$$

المؤشر السفلي "***succ***" (subscript) يشير إلى أننا نفترض أن البحث ناجح (successful) في هذه العملية الحسابية. والنتيجة التي وصلنا إليها تحقق توقُّعَنا البديهي أنه في المتوسط سيتم البحث في حوالي نصف المنظومة.

ii) والآن نفرض أن K غير موجودة في المنظومة، ونطلق على هذا الحدث "***fail***"(حالة فشل). هناك مُدْخَل واحد فقط لهذه الحالة، وسنطلق عليه I_{fail}. عدد المقارنات في هذه الحالة هو $t(I_{fail}) = n$، وبالتالي فإن $A_{fail} = n$.

وأخيرا نجمع بين حالتي وجود K في المنظومة وعدم وجودها. نفرض أن q هي احتمال وجود K في المنظومة. وبتطبيق قانون التوقعات المشروطة (law of conditional expectations) [التمهيدية 1-2 بالفصل الأول - رابعًا (نظرية الاحتمالات)] نحصل على:

$$A(n) = Pr(succ)\, A_{succ}(n) + Pr(fail)\, A_{fail}(n)$$

$$= q\left(\frac{1}{2}(n+1)\right) + (1-q)\, n = n\,(1 - \tfrac{1}{2}q) + \tfrac{1}{2}q$$

إذا كانت $q = 1$، أي أن K دائما موجودة في المنظومة، فإن $A(n) = (n+1)/2$ كما سبق. وإذا كانت $q = 1/2$، أي أن هناك احتمالا 50% لعدم وجود K في المنظومة، فإن $A(n) = 3n/4 + 1/4$، أي أن ثلاثة أرباع العناصر تقريبا يتم اختبارها. وهذا ينهي مثال 2-1.

❖ ❖ ❖

مثـــال 2-1 يوضح كيف يجب علينا أن نفسِّر D_n: مجموعة المدخلات ذوات السعة n (set of inputs of size). فبدلاً من اعتبار جميع المنظومات الممكنة - من أسماء أو أعداد أو غير ذلك - التي يمكن أن تتواجد كمدخلات، نقوم بتحديد خواص المدخلات (properties of the inputs) التي تؤثر على سلوك (behavior)

الخوارزمية. وفي هذه الحالة (أي في مثال 2-1) الخواص هي: ما إذا كانت K موجودة في المنظومة، وإن كانت كذلك فما موضع ظهورها. وأي عنصر I في D_n قد ننظر إليه على أنه مجموعة (set) [أو طبقة تكافؤ (equivalence class)] من جميع المنظومات وجميع القيم K بحيث أن K تظهر في الموضع المحدَّد (specified place) في المنظومة (أو لا تظهر إطلاقا). وبالتالي فإن $t(I)$ هي عدد العمليات التي يتم إجراؤها لأي واحد من المدخلات في I.

لاحظ أيضًا أن المُدْخَل (input) الذي يكون سلوك الخوارزمية بالنسبة له أسوأ ما يمكن يعتمد على الخوارزمية المعينة، وليس على المسألة التي نقوم بحلها. فمثلا بالنسبة للخوارزمية 2-1 تحدث أسوأ حالة عندما يكون الموضع الوحيد في المنظومة الذي يحتوي على K هو الموضع الأخير. ولكن بالنسبة لخوارزمية تبحث في منظومة في الاتجاه العكسي من الخلف إلى الأمام (backwards) [أي مبتدئةً من آخر عنصر والذي مؤشره index $= n - 1$ إلى أول عنصر والذي مؤشره index $= 0$]، تحدث أسوأ حالة إذا ظهرت K فقط في الموضع 0. [ومرة أخرى نصل إلى أسوأ حالة أخرى عندما لا تكون K موجودة في المنظومة إطلاقا].

وأخيرًا يوضح مثال 2-1 فرضاً غالبًا ما نستخدمه عند القيام بتحليل الحالة المتوسطة (average–case analysis) - أو اختصارًا بالتحليل المتوسط (average analysis) - بالنسبة لخوارزميات البحث والترتيب /الفرز (sorting and searching algorithms): وهو أن جميع العناصر متباينة / مختلفة (distinct). والتحليل المتوسط لحالة العناصر المتباينة يعطي تقريبًا مقبولاً للسلوك المتوسط (average behavior) في الحالات التي تحتوي على قيم قليلة مكررة (few duplicates) و هذا ما يحدث في كثير من التطبيقات العملية. وأما في الحالات التي قد تظهر فيها قيم كثيرة مكررة (many duplicates)، فيكون من الأصعب لنا استخدام افتراضات معقولة حول احتمال حدوث أول ظهور للقيمة K في المنظومة عند موضع معيَّن.

مثال 2-2 : ضرب المصفوفات Matrix multiplication

المسألة Problem: نفرض أن $A = (a_{ij})$ مصفوفة $m \times n$، و $B = (b_{ij})$ مصفوفة $n \times p$، وأن عناصر المصفوفتين أعداد حقيقية. المطلوب حساب حاصل ضرب المصفوفتين $C = AB$. [**ملاحظة: في حالات كثيرة نفترض أن المصفوفات مربعة، أي أن:** $m = n = p$].

طريقة الحل: نستخدم الخوارزمية التي يقتضيها تعريف حاصل ضرب مصفوفتين:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik}\, b_{kj}\ ; \qquad 0 \le i < m,\ 0 \le j < p$$

خوارزمية 2-2 : ضرب المصفوفات Matrix Multiplication

المدخلات (input): مصفوفتان ***A, B***، وثلاثة أعداد صحيحة ***m, n, p*** تفيدنا أن ***A*** مصفوفة $m \times n$ و ***B*** مصفوفة $n \times p$.

المخرجات (output): مصفوفة ***C***، وهي مصفوفة $m \times p$، حيث تُمرَّر ***C*** (passed in)، وتقوم الخوارزمية بملئها (filling in).

```
void matMult ( A, B, C, m, n, p )
  for ( i = 0; i < m; i ++ )
    for ( j = 0; j < p; j ++ )
      [ c_ij = 0;
        for (k = 0; k < n; k ++)
          c_ij + = a_ik * b_kj
```

العملية الأساسية (Basic operation) :

ضرب عناصر مصفوفية (multiplication of matrix entries).

التحليل (Analysis): لحساب كل عنصر من عناصر C، يتم إجراء n عملية ضرب. وحيث أن C تتكون من $m\ p$ عنصر، فلذلك

$$A(m, n, p) = W(m, n, p) = mnp$$

وبالنسبة للحالة الخاصة $m = n = p$، فإن $A(n) = W(n) = n^3$. وهذا ينهي مثال 2-2.

❖ ❖ ❖

مثـــال 2-2 يوضح أنه بالنسبة لبعض الخوارزميات لا تعتمد التعليمات (instructions) التي يتم تنفيذها - وبالتالي كمية الشُّغل أو الجهد المبذول- على تفاصيل المدخلات، وإنما تعتمد فقط على حجم (size) المدخلات. وفي مثل هذه الحالات تتساوى أسوأ حالة والحالة المتوسطة وفي خوارزميات أخرى للمسألة نفسها قد لا يكون هذا صحيحًا.

ويفيدنا مفهوما تحليل أسوأ حالة وتحليل السلوك المتوسط (average–behavior analysis) حتى لو اخترنا مقياسًا مختلفًا للشُّغل المبذول [مثل "وقت التنفيذ" (execution time)] . وملاحظة أن كمية الشغل المبذول تعتمد غالبا على حجم وخواص المدخلات تؤدي إلى دراسة السلوك المتوسط وسلوك أسوأ حالة، بغض النظر عن المقاييس (measures) المستخدَمة.

(3) **سعة الحيِّز المستخدَم** (Amount of space used)

يعتمد عدد خلايا الذاكرة (memory cells) التي يستخدمها برنامج ما، كما يعتمد عدد الثواني التي يستغرقها تنفيذ برنامج ما، على

التطبيـق المعـيَّن / الخـاص (particular implementation). إلا أن بعـض الاستنتاجات حول الحيِّز المستخدَم يمكن الوصول إليها بمجرد فحص واختبار الخوارزمية. وأي برنامج يحتاج إلى حيِّز لتخزين (storage space) التعليمات والثوابت والمتغيرات التي يستخدمها البرنامج والبيانات المدخلة (input data). وقـد يَسـتخدم البرنـامج أيضـا حيِّـزا إضـافيا للشُّـغل (workspace) لمعالجـة البيانات (manipulating the data) وتخزين المعلومات التي يحتاجها لإجراء عملياتـه الحسـابية. ومعلـوم أن بيانـات الإدخـال نفسـها يمكـن تمثيلـها بصـيغ مختلفة يحتاج بعضها إلى حيِّز أكبر من البعض الآخر.

وإذا كان لبيانات الإدخال صيغة طبيعية (natural form) واحدة [كمنظومة أعداد أو مصفوفة]، فإننا نقوم بتحليل (analyzing) سعة الحيِّز **الإضافي** (amount of extra space) المستخدَم، بعيدا عن البرنامج والمدخلات. فإذا كانت سعة هذا الحيِّز الإضافي ثابتة بالنسبة لحجم المدخلات (input size)، فيُقال إن الخوارزمية **"تعمل في مكانها الصحيح / في موضعها الملائم"** (work in place). وهذا المصطلح يُستخدم بصورة خاصة مع خوارزميات الترتيب / الفَرْز (sorting algorithms). [كثيرًا ما نستخدم تعريفا مخفَّفا لهذا المصطلح "في مكانها الصحيح" (in place) لا يشترط أن تكون سعة الحيز الإضافي ثابتة، ولكن يكفي فقط أن تكون دالةً لوغاريتمية في حجم المدخلات (logarithmic function of the input size)، وذلك لأن دالة اللوغاريتم **log** تنمو نموا بطيئا (grows so slowly).

وإذا كان من الممكن تمثيل المدخلات بعدة صيغ مختلفة، فإننا حينئذ نأخذ في الاعتبار الحيِّز (space) المطلوب للمدخلات نفسها وكذلك أي حيِّز إضافي (extra space) مستخدَم. وعمومًا سنشير إلى عدد "الخلايا" (cells)

المستخدَمة دون أن نُعَرِّف "الخلايا" بالضبط. ويمكننا اعتبار أن الخلية حيِّز كبير يكفي لأن يَسَع / يحتفظ (hold) بعدد واحد (one number) أو بشئ واحد (one object). وإذا كانت سعة الحيِّز المستخدم تعتمد على المدخلات المعيَّنة / الخاصة (particular input)، فيمكننا إجراء تحليل أسوأ حالة والحالة المتوسطة.

(4) **البساطة والوضوح** (Simplicity and Clarity)

من الملاحظ أن أبسط (simplest) الطرق وأقصرها للوصول مباشرة (most straightforward) إلى حل مسألة معينة لا تكون غالباً –ولكن ليس دائماً - أكفأ (most efficient) الطرق للوصول إلى هذا الحل. إلا أن بساطة الخوارزمية تُعَدُّ من الخصائص الهامة المرغوبة، فهي قد تجعل التحقق من صحة الخوارزمية أسهل، كما أنها تُسَهِّل كتابة البرنامج وتعديله (modifying) وتصحيح أخطائه (debugging). فعند اختيار خوارزمية يجب أن نأخذ في الاعتبار الوقت المطلوب لكتابة برنامج مُصَحَّح الأخطاء (debugged program)، ولكن إذا كان البرنامج سيُسْتَخْدَم كثيرا، فكفاءته قد تكون العامل الحاسم في اختيار الخوارزمية.

(5) **الأمثلية** (Optimality)

بالنسبة لأي خوارزمية لحل أي مسألة معينة لا يمكن أن تتجاوز جهودُنا لتحسين الخوارزمية نقطةً معينة. فلكل مسألةٍ درجة تعقيد ذاتية ملازمة لها ومتضمنة في صلبها ومتأصلة في طبيعتها الأساسية (inherent complexity). أي أنه يوجد حد أدنى (some minimum) لمقدار الشُّغل المبذول المطلوب لحل المسألة. ولتحليل درجة تعقيد مسألةٍ

ما - مقابل درجة تعقيد خوارزمية معينة - نختار طائفة من الخوارزميات (a class of algorithms) [غالبًا بتحديد أنواع العمليات التي سيُسمح للخوارزميات بإجرائها] ومقياس لدرجة التعقيد (a measure of complexity)، مثل العملية (أو العمليات) الأساسية التي نقوم بعَدِّها (to be counted). ثم قد نسأل بعد هذا: كم عدد العمليات التي نحتاجها فعلاً (actually needed) لحل المسألة؟ ونقول إن خوارزميةً ما هي خوارزمية **مُثلَى** (**optimal**) [في أسوأ حالة (in the worst case)] إذا لم توجد أي خوارزمية أخرى في هذه الطائفة من الخوارزميات قيد الدراسة تُجرى عددا أقل من العمليات الأساسية [في أسوأ حالة] . ولاحظ أننا عندما نتكلم عن الخوارزميات في طائفة الخوارزميات قيد الدراسة لا نعني فقط تلك الخوارزميات التي فَكَّرْنا فيها واخترناها، وإنما نعني جميع الخوارزميات الممكنة والمحتملة (all possible algorithms) بما في ذلك الخوارزميات التي لم يتم اكتشافها بعد (not yet discovered). فكلمة **مُثْلّى** (**optimal**) لا تعني "أفضل خوارزمية معلومة" (the best known)، وإنما تعني "أفضل خوارزمية ممكنة" (the best possible).

الحدود السُفْلَى ودرجات تعقيد المسائل

Lower Bounds and the Complexity of Problems

والآن نحاول الإجابة على السؤال: كيف يمكننا بيان أن خوارزميةً ما هي خوارزمية مُثْلَى؟ هل يجب علينا أن نقوم بتحليل (analyzing) كل خوارزمية ممكنة أخرى على حدة (individually) [بما في ذلك حتى الخوارزميات التي لم نفكر فيها بعد!!]؟ بالطبع: لا. ولكن يمكننا إثبات نظريات تضع حدا أدنى لعدد العمليات التي نحتاجها لحل مسألةٍ ما. وبعد ذلك إذا وجدنا أي خوارزمية تقوم بإجراء هذا العدد من العمليات لحل تلك المسألة، فهي

خوارزمية مُثْلَى. وهكذا فأمامنا مهمتان علينا القيام بهما كي نجد خوارزمية جيدة، أو - من وجهة نظر أخرى - كي نجيب على التساؤل: كم هو الشُّغل / الجهد (how much work?) الضروري والكافي (necessary and sufficient) لحل المسألة؟

1) ابتكر / صمم / استنبط (devise) ما قد تبدو أنها خوارزمية ذات كفاءة عالية (an efficient algorithm)، ونطلق عليها الخوارزمية A. قم بتحليل A وأوجد دالة W_A بحيث أنه بالنسبة للمدخلات التي حجمها n، تقوم الخوارزمية A بإجراء عدد من الخطوات (steps) لا يتجاوز (at most) $W_A(n)$ في أسوأ حالة (in the worst case).

2) لدالةٍ ما F (for some function) اثبت نظريةً تقرر أنه لأي خوارزمية في طائفة الخوارزميات قيد الدراسة توجد مدخلاتٌ ما (some input) سعتها n (size) بحيث أنه يجب أن تقوم الخوارزمية بإجراء عدد من الخطوات (steps) لا يقل عن $F(n)$ (at least) لهذه المدخلات.

إذا تساوت الدالتان W_A, F فالخوارزمية A خوارزمية مُثْلَى (لأسوأ حالة). وإذا لم تتساوا فقد تكون هناك خوارزمية أفضل (a better algorithm)، أو قد يكون هناك حد أدنى أفضل (a better lower bound)، ولاحظ أن تحليل خوارزمية معينة يعطي **حدّاً أعلى (upper bound)** لعدد الخطوات الضرورية لحل مسألة، بينما نظرية من النوع المذكور في النقطة 2) في الفقرة السابقة تعطي **حدّاً أدنى (lower bound)** لعدد الخطوات الضرورية (في أسوأ حالة). وفي هذا الكتاب سنرى بإذن الله مسائل نعلم لها خوارزميات مُثْلى، ومسائل أخرى لا تزال فيها فجوة بين أفضل حدٍّ أدنى معلوم وأفضل خوارزمية معلومة. وفيما يلي نعرض أمثلة لكل من الحالتين.

يُعَدُّ مفهوم الحد الأدنى لسلوك الخوارزميات في أسوأ حالة من المفاهيم الهامة جدا في موضوع درجة التعقيد الحوسبية (computational complexity). والمثال التالي (مثال 2-3) والمسائل التي سندرسها في نهاية هذا الفصل (عن البحث في منظومة مرتبة) وفي الفصلين الثالث والرابع ستساعد بإذن الله في توضيح معنى الحدود السُّفلى / الدنيا (lower bounds) وكذلك في عرض بعض الطرق (techniques) للحصول على هذه الحدود. ونؤكد هنا أن التعريف "F هو حدٌّ أدنى لطائفة من الخوارزميات" يعني أنه **لأي** خوارزمية في هذه الطائفة، ولأي حجم مدخلات n (input size) توجد مدخلات (some input) حجمها n بحيث يجب أن تقوم الخوارزمية بإجراء عدد من العمليات الأساسية لا يقل عن $F(n)$ (at least).

مثال 2-3: إيجاد أكبر عنصر في منظومة

Finding the largest entry in an array

- **المسألة (problem)**: أوجد أكبر عنصر في منظومة مكونة من n عدد. (نفرض أن نوع الأعداد **float**، ويمكن أن يكون أي نوع عددي).
- **طائفة الخوارزميات (Class of Algorithms)**: الخوارزمات التي يمكنها أن تقارن (compare) وتنسخ (copy) أعدادا من النوع **float**، ولكن لا تقوم بأي عمليات أخرى عليها.
- **العملية الأساسية (Basic Operation)**: المقارنة بين عنصر منظومة وأي شئ من النوع **float**: قد يكون عنصر منظومة أخرى أو متغيرا مخزونا (a stored variable).

الخوارزمية 2-3: findMax

- **المدخلات (Input)**: E: منظومة أعداد، مُعَرَّفة للمؤشرات $0, \ldots, n-1$; $n \geq 1$ حيث n هي عدد العناصر.
- **المخرجات (Output)**: الخوارزمية تعيد **max** وهو أكبر عنصر في المنظومة E.

```
float findMax ( float [ ] E, int n )
1. max = E[0];
2. for (index = 1; index < n; index ++)
3.     if ( max < E[index] )
4.         max = E[index];
5. return max;
```

الحد الأعلى (Upper Bound):

مقارنات عناصر المنظومة تتم في السطر 3 الذي يُنَفَّذ عدد $n - 1$ من المرات بالضبط. وبالتالي فإن $n - 1$ هو حد أعلى (an upper bound) لعدد المقارنات اللازمة / الضرورية (necessary) لإيجاد القيمة العظمى (maximum) في أسوأ حالة. فهل توجد خوارزمية تقوم بإجراء عدد أقل من المقارنات؟

الحد الأدنى (Lower Bound)

لإيجاد حد أدنى يمكننا أن نفرض أن عناصر المنظومة جميعها متباينة (distinct). وهذا الفرض مسموح به نظرا لأنه إذا أمكننا إيجاد حد أدنى لسلوك أسوأ حالة لمجموعةٍ جزئيةٍ من المدخلات (منظومات عناصرها متباينة)،

فستكون حدا أدنى لسلوك أسوأ حالة عندما نأخذ في الاعتبار جميع المدخلات الصالحة (all valid inputs).

وفي منظومةٍ عناصرها متباينة و عددها n، نرى أن $n-1$ عنصراً **ليست** هي القيمة العظمى (maximum). فيمكننا أن نستنتج أن عنصرا معينا (a particular entry) ليس هو القيمة العظمى فقط إذا كان أصغر من عنصر واحد آخر - من عناصر المنظومة - على الأقل. وبالتالي فإن $n-1$ عنصرا يجب أن يكونوا "خاسرين" (losers) في المقارنات التي تقوم الخوارزمية بإجرائها. وكل مقارنة يكون فيها خاسر واحد فقط. وبالتالي فيجب إجراء $n-1$ مقارنة على الأقل. أي أنه إذا بَقى لدينا عنصران غير خاسرين (nonlosers) أو أكثر عندما تنتهي / تتوقف (terminates) الخوارزمية، فلا يمكن للخوارزمية حينئذ أن تكون قد تأكدت من أنها قد حددت القيمة العظمى. وبناءً عليه فإن $F(n) = n-1$ هو حد أدنى لعدد المقارنات التي نحتاجها.

الاستنتاج (Conclusion):

الخوارزمية 2-3 (findMax) خوارزمية مُثْلَى. وهذا ينهي مثال 2-3.

ملاحظة: كان يمكننا تَبَنِّي وجهة نظر مختلفة قليلا لإيجاد الحد الأدنى في مثال 2–3. إذا أُعطينا خوارزمية ومنظومة مكونة من n عدد بحيث أن الخوارزمية تتوقف وتعطِي إجابةً بعد إجراء أقل من $n-1$ مقارنة، فإنه يمكننا إثبات أن الخوارزمية تعطِي الإجابةَ الخاطئة لمجموعةٍ ما من بيانات المدخلات. وإذا لم يتم إجراء أكثر من $n-2$ مقارنة، فهناك عنصران لن يكونا خاسرَيْن (losers) إطلاقا ، أي أنه ليس معلومًا أنهما أصغر من أي عناصر أخرى. والخوارزمية يمكنها تحديد عنصر واحد على الأكثر من هذين العنصرين باعتباره القيمة العظمى. ويمكننا ببساطة أن نستبدل بالعنصر الآخر عددا أكبر (إن كان ذلك ضروريا). ونظرا لأن نتائج جميع المقارنات التي يتم

إجراؤها ستكون هي نفسها كما سبق، فإن الخوارزمية ستعطِي الإجابة نفسها كما سبق وستكون خاطئة.

هذه الحجة (argument) تُعَدُّ برهانا بالمكافئ العكسي (contrapositive) [انظر الفصل الأول]. وهكذا أثبتنا أنه "إذا قامت الخوارزمية A بإجراء عدد من المقارنات أقل من $n-1$ في أي حالة، فإن A تكون غير صحيحة". وبالمكافئ العكسي يمكننا استنتاج أنه "إذا كانت A صحيحة، فإنها تُجرِي $n-1$ مقارنة على الأقل وذلك في جميع الحالات". وهذا يوضح تقنية مفيدة لإيجاد حدود سفلى، أي لبيان أنه إذا لم تقم خوارزمية ما بجهد كاف (enough work)، فإنه يمكننا ترتيب المدخلات (arranging the input) كي تعطِي الخوارزميةُ الإجابةَ الخاطئة.

مثال 2-4: ضرب المصفوفات Matrix Multiplication

المسألة : نفرض أن $A = (a_{ij})$, $B = (b_{ij})$ مصفوفتان $n \times n$ عناصرهما أعداد حقيقية (real). احسب مصفوفة حاصل الضرب $C = AB$.

طائفة الخوارزميات: الخوارزميات التي تستطيع أن تجري عمليات الضرب والقسمة والجمع والطرح على عناصر المصفوفات وعلى النتائج المرحلية / الوسطية (intermediate results) التي نحصل عليها بإجراء هذه العمليات على تلك العناصر.

العملية الأساسية: الضرب

- **الحد العلوي:** الخوارزمية المعتادة (انظر مثال 2–2) تُجرى n^3 عملية ضرب.
  وبالتالي فيلزمنا (necessary) على الأكثر n^3 عملية ضرب
  (at most n^3 multiplications).
- **الحد السفلي:** تم إثبات أنه يلزمنا على الأقل n^2 عملية ضرب.

الاستنتاجات: من المعلومات المتوفرة لدينا ليس أمامنا سبيل لتحديد ما إذا كانت الخوارزمية المعتادة مُثْلَى (optimal) أم لا. وقد حاول بعض الباحثين تحسين الحد السفلي، أي إثبات أنه يلزمنا أكثر من n^2 عملية ضرب، بينما بحث آخرون عن خوارزميات أفضل. وإلى اليوم فقد تم بيان أن الخوارزمية المعتادة **ليست** مُثْلى. وهناك طريقة تُنَفِّذ تقريبا $n^{2.376}$ عملية ضرب. هل هذه الطريقة مُثْلى؟ والحد الأدنى لم يتم تحسينه إلى اليوم، وبالتالي فإننا لا نعلم ما إذا كانت هناك خوارزميات تُجري عددا أقل بدرجة ملموسة من عمليات الضرب.

❖ ❖ ❖

حتى الآن ما زلنا نناقش الحدود السفلى والأمثلية لسلوك أسوأ حالة (worst– case behavior). فماذا عن سلوك الحالة المتوسطة (average behavior)؟ يمكننا استخدام الطريقة نفسها التي اتبعناها مع سلوك أسوأ حالة. اختر ما قد تبدو أنها خوارزمية جيدة وحاول معرفة الدالة $A(n)$ حيث تقوم الخوارزمية بإجراء عدد $A(n)$ من العمليات - في المتوسط - لمدخلات حجمها n. ثم اثبت نظرية تقرر أن أي خوارزمية في طائفة الخوارزميات قيد الدراسة يجب أن تُجرى على الأقل $G(n)$ عملية - في المتوسط - لمدخلات حجمها n. فإذا كانت $A = G$، فإنه يمكننا أن نقول إن السلوك المتوسط (average behavior) للخوارزمية هو سلوك أمثل (optimal). وإن لم تكن $A = G$ فنبحث عن خوارزمية أفضل، او حد سفلي أفضل (أو كليهما).

يُعَدُّ تحليل عدد العمليات بالضبط في كثير من المسائل من المهام الشاقة جدا. ومن المعتاد اعتبار أي خوارزمية أنها مُثْلَى إذا كان عدد العمليات - التي تقوم بتنفيذها - في مدى عامل ثابت (within a constant factor)

للعدد المثالي بالضبط (of the exact optimum) [والذي يكون هو نفسه معلوما غالبا في مدى عامل ثابت]. وسنقوم بإذن الله بعد قليل بعرض طريقة لتحليل مسائل عديدة في مدى عامل ثابت، إلا أننا سنكون غير قادرين على إجراء تحليل دقيق / مضبوط (exact analysis).

ويمكننا استخدام الطريقة نفسها التي اتبعناها لتحليل الزمن (time analysis) وذلك لتدارس وفحص (investigation) الحيِّز المستخدم (space usage)، أي لبحث وتحليل المكان. وذلك بتحليل خوارزمية معينة للحصول على حد أعلى (upper bound) لِقَدْر الحيِّز الذي نحتاجه، وبرهنة نظرية تعطِي حدًّا أسفل (a lower bound). والآن هل يمكننا إيجاد خوارزمية واحدة لمسألةٍ معطاة بحيث تكون خوارزمية مُثلى بالنسبة لكل من قدر الجهد المبذول وقدر الحيِّز المستخدم ؟ الإجابة على هذا السؤال هي: أحيانا. وفي بعض المسائل تكون هناك مقايضة / مفاضلة (trade–off) بين الزمن والمكان (time and space).

التنفيذ والبرمجة Implementation and Programming

التنفيذ هو مهمة تحويل خوارزمية إلى برنامج حاسوبي (a computer program). والخوارزميات يمكن وصفها بتعليمات (instructions) مُفَصَّلة شبيهة بلغة الحاسوب (computer–language–like) لمعالجة (manipulating) والتعامل مع المتغيرات وبنى المعطيات / هياكل البيانات (data structures)، أو وصفها بتوضيحات وشروحات (explanations) - باللغة العربية أو الإنجليزية - بحتة / تجريدية جدا (very abstract) وعالية المستوى (high–level) لطرق حل مسائل مجردة (abstract problems)، دون أي ذِكْر لتمثيلات الحاسوب (computer representations) للأشياء التي تشملها (objects involved) طرق الحل هذه. وبناءً على هذا فقد يكون تنفيذ

أي خوارزمية هو مجرد مهمة ترجمة مباشرة، أو قد يكون مهمة شاقة وطويلة جدا تتطلب عددا من القرارات (decisions) الهامة من طرف المبرمج بخصوص اختيار هياكل البيانات بصفة خاصة. وسنقوم بإذن الله بدراسة ومناقشة التنفيذ بالمفهوم العام (in the general sense) لاختيار هياكل البيانات ووصف طرق لتنفيذ (carrying out) تعليمات معطاة في وصف (description) خوارزمية باللغة العربية / الإنجليزية. وهناك سببان لهذه الدراسة / المناقشة. الأول: أن هذه الدراسة هي جزء طبيعي وهام من عملية كتابة برنامج عملي جيد (a good working program). والثانى: غالبًا ما يكون أخذ تفاصيل التنفيذ في الاعتبار ضروريا لتحليل أي خوارزمية، فمقدار الزمن (amount of time) المطلوب لإنجاز عمليات متعددة على أشياء مجردة (abstract objects) كمجموعات (sets) ورسوم بيانية (graphs) يعتمد على كيفية تمثيل (representation) هذه الأشياء. فمثلا تكوين (forming) اتحاد (union) مجموعتين قد يتطلب عملية واحدة أو عمليتين فقط إذا كانت المجموعتان ممثلتين كقائمتين مترابطتين (linked lists)، بينما يتطلب عددا كبيرا من العمليات يتناسب مع عدد العناصر في إحدى المجموعتين إن كانت المجموعتان ممثلتين كمنظومتين (arrays) ومطلوب نسخ إحداهما في الأخرى.

وبالمفهوم الضيق فإن التنفيذ أو ببساطة البرمجة تعني تحويل وَصْفٍ مُفَصَّلٍ لخوارزمية وبنيات المعطيات التي تستخدمها إلى برنامج لحاسوب معيَّن. وسيكون تحليلنا مستقلا عن التنفيذ (implementation–independent) بهذا المفهوم. وبأسلوب آخر سوف لا يعتمد على الحاسوب المستخدم ولغة البرمجة المستخدمة والتفاصيل الثانوية (minor details) للخوارزمية أو البرنامج.

ويستطيع المبرمج تحسين (refining) تحليل الخوارزميات قيد الدراسة باستخدام معلومات عن الحاسوب المعيَّن المستخدم. فمثلا: إذا كانت هناك أكثر من عملية واحدة يتم حسابها / عدّها، فإن العمليات يمكن وزنها (weighted) بناءً على أوقات تنفيذها (their execution times)، أو إن العدد الفعلي للثواني التي يستغرقها الحاسوب (في أسوأ حالة أو في الحالة المتوسطة) يمكن تقديره. وأحيانًا تؤدي معرفة الحاسوب المستخدم إلى تحليل جديد. فمثلاً إذا كان للحاسوب أيُّ تعليمات قوية عالية القدرة غير معتادة (unusual powerful instructions) يمكن استخدامها بكفاءة عالية في المسألة قيد الدراسة، فإنه يمكننا دراسة طائفة الخوارزميات التي تستفيد من هذه التعليمات، ونعدُّها باعتبارها العمليات الأساسية. وإذا كان للحاسوب مجموعة تعليمات (instruction set) محدودة جدا (very limited) تجعل تنفيذ العملية الأساسية حرجا / غير ملائم (awkward)، فيمكننا النظر في طائفة أخرى من الخوارزميات. إلاَّ أنه عموماً إذا تم إجراء التحليل غير المعتمد على التنفيذ (implementation–independent analysis) بصورة جيدة، فإن التحليل المعتمد على البرنامج (program–dependent analysis) يجب أن يقوم أساسا بإضافة تفاصيل.

وأي تحليل مُفَصَّل لسعة الحيِّز المستخدم بالخوارزمية قيد الدراسة يُعَدُّ بالطبع مناسبًا أيضًا حين النظر في طرق تنفيذ معيَّنة (particular implementations).

وأي معلومات خاصة عن مدخلات (inputs) المسألة التي نبحث لها عن خوارزمية يمكن أن تُستخدم لتحسين (refining) التحليل. فمثلاً إذا عَلِمْنَا أن

المدخلات ستكون محصورة في مجموعة جزئية (subset) معيَّنة من مجموعة جميع المدخلات المحتملة ، فيمكننا حينئذ إجراء تحليل أسوأ حالة لهذه المجموعة الجزئية. وكما لاحظنا فإن تحليلا جيدا للسلوك المتوسط (a good average–behavior analysis) سوف يعتمد على معرفة احتمال حدوث المدخلات المختلفة.

تصنيف الدوال بناءً على معدلات نموها المُقارِب

Classifying Functions by their Asymptotic Growth Rates

ذكرنا سابقا عند الحديث عن قياس مقدار العمل المبذول بخوارزميةٍ ما، والمقارنة بين خوارزميتين أننا عادةً لا نَعُدُّ كل خطوة تنفذها الخوارزمية، ولذلك فإن التحليل يشوبه عدم الدِّقة، إلا أننا سوف نرضى به إذا كان العدد الإجمالي للخطوات متناسبًا تقريبًا (roughly proportional) مع عدد العمليات الأساسية التي نحسبها. وهذا يُعَدُّ كافيا للفصل بين الخوارزميات التي تقوم بتنفيذ مقادير عمل مختلفة اختلافًا جوهريًّا كبيرًا (drastically different amounts of work) لمدخلات كبيرة (large inputs).

نفرض أن خوارزميةً ما لمسألةٍ معينة تُجري $2n$ عمليةً أساسية، وبالتالي فالعدد الإجمالي للعمليات يساوي تقريبا $2cn$، حيث c ثابتٌ ما. ونفرض أن خوارزميةً أخرى للمسألة نفسها تُجري $4.5n$ عملية أساسية، وبالتالي فالعدد الإجمالي للعمليات يساوي تقريبا $4.5c'n$، حيث c' ثابتٌ ما. أيُّ الخوارزميتين تُعَدُّ أسرع تنفيذا (runs faster)؟ في الحقيقة لا نعلم. فالخوارزمية الأولى قد تجري عمليات إضافية (overhead operations) أكثر، أي أن ثابت التناسب c فيها قد يكون أكبر بكثير. وهكذا إذا اختلفت دالتان تصفان سلوكَي خوارزميتين بعامل ثابت (a constant factor)، فقد لا يكون هناك

معنى لمحاولة التمييز بينهما [إلاَّ إذا قمنا بإجراء تحليل أكثر تحسينا (a more refined analysis)] . ونحن نعتبر أن مثل هاتين الخوارزميتين تقعان في طبقة التعقيد نفسها (same complexity class).

نفرض الآن أن خوارزمية مسألةٍ ما تجري عمليات ضرب عددها $n^3/2$، وأخرى تجري $5n^2$ عملية ضرب. أيُّ الخوارزميتين أسرع تنفيذا؟ بالنسبة لقيم n الصغيرة تقوم الخوارزمية الأولى بتنفيذ عدد أقل من عمليات الضرب، بينما لقيم n الكبيرة فإن الخوارزمية الثانية أفضل، حتى لو قامت بتنفيذ عدد أكبر من العمليات الإضافية (overhead operations). فمُعَدَّل نمو دالة تكعيبية يُعَدُّ أكبر بكثير من معدل نمو دالة تربيعية لدرجة أن ثابت التناسب (constant of proportionality) لا يصبح له تأثير يُذكر عندما تصبح قيمة n كبيرة.

فكما يشير المثالان السابقان نريد طريقة للمقارنة بين الدوال أو لتصنيفها بحيث تتجاهل العوامل الثابتة (constant factors) والمدخلات الصغيرة (small inputs). ونستطيع الحصول على مثل هذا التصنيف بدراسة ما يُعْرَف بـ "**معدل النمو المُقارِب**" **(asymptotic growth rate)**، أو "**الرتبة المُقارِبة**" **(asymptotic order)**، أو ببساطة "**رتبة الدالة**" **(order of the function)**.

وعند المقارنة بين خوارزميتين لِوَقْتَي تنفيذهما (running times) الرتبة نفسها (of the same order) يمكننا النظر إلى ثابت التناسب، أو حساب عدد جميع العمليات بما فيها الإضافيات (overhead) للحصول على تقدير أدق لوقت التنفيذ، أو يمكننا النظر إلى معيار آخر كمقدار الحيِّز الإضافي المستخدم (amount of extra space used).

تعريفات والاصطلاحات المُقارِبة Definitions and Asymptotic Notation

تعريف 2-5: اصطلاحات الأعداد الطبيعية والأعداد الحقيقية

Notations for natural numbers and real numbers

1) مجموعة الأعداد الطبيعية (natural numbers) يُرمز لها هكذا:

$\mathbf{N} = \{0, 1, 2, 3, \ldots\}$.

2) مجموعة الأعداد الصحيحة الموجبة (positive integers) يُرمز لها هكذا:

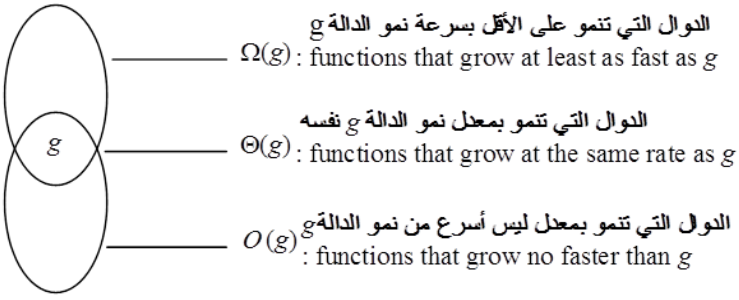
$\mathbf{N}^+ = \{1, 2, 3, \ldots\}$.

3) مجموعة الأعداد الحقيقية (real numbers) يُرمز لها بالرمز: $\mathbf{R}$.

4) مجموعة الأعداد الحقيقية الموجبة (positive reals) يُرمز لها بالرمز: $\mathbf{R}^+$.

5) مجموعة الأعداد الحقيقية غير السالبة (nonnegative reals) يُرمز لها بالرمز: $\mathbf{R}^*$.

نفرض أن f, g دالتان من N إلى R^*. الشكل التالي (شكل 2-1) يصف بصورة غير رسمية / غير شكلية (informally) المجموعات التي نستخدمها لبيان العلاقات بين رُتَب الدوال (orders of functions). وبالاحتفاظ في أذهاننا بهذه الصورة وبالتعريفات غير الرسمية فإن ذلك يساعدنا على توضيح التعريفات الرسمية / الشكلية والخواص التالية.



شكل 2-1

Big omega (Ω), big theta (Θ), & big oh (O)

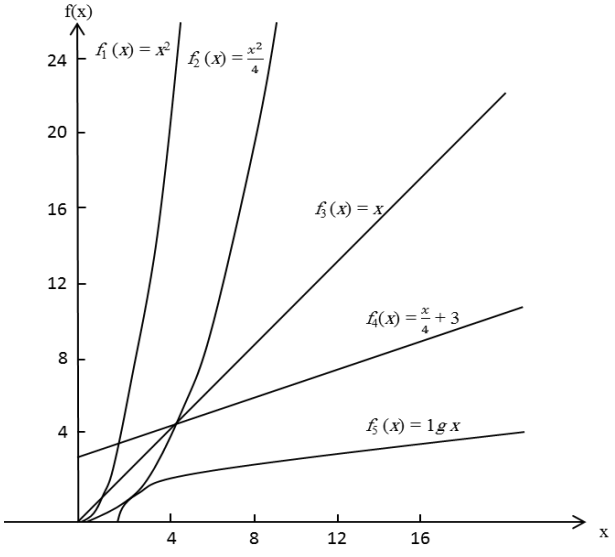
تعريف 2-6: المجموعة $O(g)$

نفرض أن g دالة من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة. المجموعة $O(g)$ هي مجموعة الدوال f، أيضا من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة، بحيث أنه يوجد ثابت حقيقي $c > 0$، وثابت صحيح غير سالب n_0، بحيث أن

$$f(n) \leq c\, g(n) \qquad \forall\ n \geq n_0$$

يمكننا أن ننظر إلى الدالة g على أنها دالة معطاة (given function)، وإلى الدالة f على أنها الدالة التي نقوم بتحليلها. ولاحظ أن أي دالة f قد تكون في المجموعة $O(g)$ حتى إذا كان $f(n) > g(n)\ \forall n$. والنقطة الهامة هي أن الدالة f محدودة من أعلى (bounded above) بأحد المضاعفات الثابتة للدالة g (some constant multiple of). وأيضا لا نأخذ في الاعتبار العلاقة بين الدالة f والدالة g لقيم n الصغيرة. والشكل التالي (شكل 2-2) يبيِّن علاقات الرتبة (order relations) لبعض الدوال. [1]ولاحظ أن الدوال التي تظهر في

شكل 2-2 قد رُسمت كدوال متصلة (continuous) مُعَرَّفة على $\mathbf{R}^+$ أو $\mathbf{R}^*$.
والدوال التي تصف سلوك معظم الخوارزميات التي سندرسها بإذن الله لها مثل هذه الامتدادات الطبيعية (natural extensions).



شكل 2-2

رُتَب الدوال orders of functions

المجموعة $O(g)$ يطلَق عليها عادةً "Oh الكبيرة للدالة g" (big oh of "g)
(g). ورغم أننا عَرَّفنا $O(g)$ على أنها مجموعة، إلا أنه من الشائع عمليا أن نقول:
"f هي oh للدالة g" (f is oh of g) بدلا من: "f هي عنصر من عناصر مجموعة oh للدالة g" (f is a member of oh of g).

- لاحظ أن $f_3 \in O(f_4)$ رغم أن $f_3(x) > f_4(x)$ للقيم $x > 4$ نظرا لأن الدالتين خطيتان.

- الدالتان f_1, f_2 لهما الرتبة نفسها. وهما تنموان بمعدل أسرع من الدوال الثلاث الأخرى.
- الدالة f_5 لها أقل رتبة من بين الدوال التي تظهر بالشكل.

وهناك طريقة بديلة لإثبات أن f موجودة في $O(g)$:

تمهيدية 2-1: أي دالة f تكون $f \in O(g)$ إذا كان $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$

بما في ذلك الحالة التي تكون فيها النهاية تساوي صفرا. أي أنه إذا كانت نهاية (lim) نسبة (ratio) f إلى g موجودة وليست ∞، فإن f تنمو بمعدل ليس أسرع من g.

مثال 2-5: دوال ذوات رُتَب مُقارِبة مختلفة

(Functions of different asymptotic orders)

نفرض أن

$$f(n) = n^3/2, \quad g(n) = 37\ n^2 + 120\ n + 17.$$

سنثبت أن $g \in O(f)$، ولكن $f \notin O(g)$.

نظراً لأنه لجميع القيم $n \geq 78$ تكون $g(n) < 1.f(n)$ فلذلك تكون $g \in O(f)$. ويمكننا أيضا الوصول إلى النتيجة نفسها كما يلي:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{37n^2 + 120n + 17}{n^{3/2}} = \lim_{n\to\infty} \left(74/n + 240/n^2 + 34/n^3\right) = 0.$$

ويمكننا إثبات أن $f \notin O(g)$ بملاحظة أن $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. وفيما يلي طريقة بديلة، وهي البرهنة بالتناقض، حيث نفرض أن $f \in O(g)$ ثم نصل إلى

تناقض. إذا كانت $f \in O(g)$ فإنه يوجد ثابتان c, n_0 بحيث أنه لجميع القيم n تكون $\geq n_0$

$$\frac{n^3}{2} \leq 37cn^2 + 120cn + 17c$$

وبالتالي فإن

$$\frac{n}{2} \leq 37c + \frac{120c}{n} + \frac{17c}{n^2} \leq 174c$$

وحيث أن c ثابت و n قد تكون كبيرة كبرًا اختياريًّا (arbitrarily large)، فمن المستحيل أن تتحقق لدينا المتباينة $n/2 \leq 174c$ لجميع القيم $n \geq n_0$.

❖ ❖ ❖

النظرية التالية تفيدنا في حساب النهايات عندما تمتد f, g إلى دالتين متصلتين قابلتين للمفاضلة (continuous differentiable functions) على الأعداد الحقيقية.

تمهيدية 2-2: قاعدة "هوبيتال" **(L'Hopital's Rule)**

نفرض أن f, g دالتان قابلتان للمفاضلة، وأن مشتقتيهما على الترتيب هما f', g' ، إذا كان

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$$

فإن

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

مثال 2-6:

استخدام قاعدة "هوبيتال"

نفرض أن $f(n) = n^2, \quad g(n) = n \lg n$.

اثبت أن $f \notin O(g), \quad g \in O(f)$.

الحل :

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n \lg n} = \lim_{n\to\infty} \frac{n}{\lg n} = \lim_{n\to\infty} \frac{n \ln(2)}{\ln n} = \lim_{n\to\infty} \frac{\ln(2)}{1/n} = \lim_{n\to\infty} n \ln(2) = \infty$$

ولذلك فإن $f \notin O(g)$. وحيث أن النسبة العكسية (inverse ratio) تؤول إلى الصفر، فلذلك $g \in O(f)$.

❖ ❖ ❖

وأما تعريف المجموعة $\Omega(g)$ – وهي مجموعة الدوال التي تنمو بسرعة لا تقل عن سرعة نمو الدالة g– فهو ثَنَوِيٌّ (dual) تعريف المجموعة O(g) [*]

تعريف 2-7: المجموعة $\Omega(g)$

نفرض أن g دالة من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة. $\Omega(g)$ هي مجموعة الدوال f –أ يضًا من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة– بحيث أنه يوجد ثابت حقيقي $c > 0$، وثابت صحيح غير سالب n_0، بحيث أن $f(n) \geq c\, g(n) \quad \forall\, n \geq n_0$

وأما الطريقة البديلة لإثبات أن f موجودة في $\Omega(g)$ فهي كما يلي:

---

[*] بعض الكتب تستخدم تعريفًا مختلفًا قليلاً للمجموعة Ω، حيث تُخَفِّف الشرط / العبارة: "لكل ∀" إلى: "لعدد كبير كبرا لا نهائيا" (for infinitely many). وبناءً عليه يتم تعديل تعريف Θ.

تمهيديـة 2-3: الدالـة f تنتمـي إلى المجموعـة $\Omega(g)$ [أي أن $f \in \Omega(g)$] إذا كان

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

ويشمل ذلك الحالة التي تكون فيها النهاية ∞.

تعريف 2-8: المجموعة $\Theta(g)$: الرتبة المُقارِبة للدالة g

The set $\Theta(g)$, asymptotic order of g

نفرض أن g دالة من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة. المجموعة $\Theta(g)$ هي مجموعة الدوال التي تنتمي إلى كل من $O(g)$ و $\Omega(g)$، أي أن $\Theta(g) = O(g) \cap \Omega(g)$. وعادةً العبارة "$f \in \Theta(g)$" تقرأ هكذا: "$f$ من رتبة g" (f is order g) وغالبا ما نستخدم العبارة / الاصطلاح "رتبة مُقارِبة" (asymptotic order) للتحديد (definiteness)، وكذلك نرى الاصطلاح "درجة التعقيد المُقارِبة" (asymptotic complexity).

تمهيدية 2-4: الدالة f تنتمي إلى المجموعة $\Theta(g)$ أي أن $f \in \Theta(g)$ إذا كان

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

حيث c ثابتٌ ما بحيث أن $0 < c < \infty$.

مثال 2-7: الرتبة المُقارِبة لبعض الخوارزميات

Asymptotic order of some algorithms

درجتا تعقيد أسوأ حالة (worst–case complexities) للخوارزمية

2-1 (البحث التتابعي غير المرتب) والخوارزمية 2-3 (إيجاد أكبر عنصر) تنتميان إلى $\Theta(n)$. ودرجة تعقيد [أسوأ حالة أو المتوسطة] للخوارزمية 2-2 (ضرب المصفوفات في الحالة $m = n = p$) تنتمي إلى $\Theta(n^3)$.

❖ ❖ ❖

والاصطلاح الشائع استخدامه عند الحديث عن مجموعات الرتب /المراتب / المرتبات (order sets) يتسم بعدم الدقة. فمثلا حين نقول "هذه خوارزمية من المرتبة n^2" (an order n^2 algorithm)، فإن ذلك يعني في الحقيقة أن الدالة التي تصف سلوك الخوارزمية توجد في $\Theta(n^2)$. والتدريبات تؤكد على عدة حقائق حول مجموعات المراتب الشائعة، والعلاقات فيما بينها، كحقيقة أن $n(n-1)/2 \in \Theta(n^2)$.

وأحيانا نرغب في الإشارة إلى أن دالةً ما لها مرتبة / رتبة مقارِبة (asymptotic order) أكبر فعلاً أو أصغر فعلاً (strictly greater / smaller) من مرتبة أخرى. يمكننا حينئذ أن نستخدم التعريف التالي.

تعريف 2-9: المجموعتان $o(g)$, $\omega(g)$

نفرض أن g دالة من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة.

1) المجموعة $o(g)$ هي مجموعة الدوال f – أيضًا من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة– بحيث أن $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

2) المجموعة $\omega(g)$ هي مجموعة الدوال f – أيضا من الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية الموجبة– بحيث أن $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

أي أن الدوال في $o(g)$ هي الدوال الصغرى "smaller" لدوال $O(g)$، بينما $\omega(g)$ هي الدوال الكبرى "larger" لدوال $\Omega(g)$. ومع ذلك فإننا نقرأ $\omega(g)$: "أومجا الصغيرة للدالة g" (small omega of g)!! كما نقرأ $o(g)$: "o الصغيرة للدالة g" (small oh of g).

أهمية الرتبة المُقارِبة Importance of Asymptotic Order

يبـيِّن جـدول 2-1 فـترات تنفيـذ (running times) عـدة خوارزميـات فعلية للمسألة نفسها. [العمود الأخير أقصى يمين الجدول لا يقابل خوارزمية معينـة للمسـألة، وإنمـا أُضـيف فقـط ليبـيِّن مـدى سـرعة نمـو الـدوال الأُسِّـيَّة (exponential functions)، وبالتـالي يظهـر عيـب الخوارزميـات الأسِّـيَّة]. وبنظرة عامة على الجدول يتبيَّن لنا مدى سرعة زيادة وقت التنفيذ (running time) مـع زيـادة حجـم المدخلات بالنسـبة للخوارزميـات الأكثـر تعقيـدًا (of higher complexties). وأحد الدروس الهامة التي نستخلصها من الجدول أن المعـاملين الثـابتين الكـبيرين (large constant factors) في الخـوارزميتين $\Theta(n)$, $\Theta(n \log n)$ لا يجعلـهما أبطـأ مـن بقيـة الخوارزميـات إلاَّ في حالـة المدخلات الصغيرة جدًا.

الخوارزمية	1	2	3	4	5
دالة الزمن (مايكرو ثانية)	$33n$	$46n1g\ n$	$13n^2$	$3.4n^3$	2^n
حجم المدخلات (n)	زمن الحل: الفترة الزمنية اللازمة لتنفيذ الخوارزمية				
10	.00033 ثانية	.0015 ثانية	.0013 ثانية	.0034 ثانية	.001 ثانية
100	.003 ثانية	.03 ثانية	.13 ثانية	3.4 ثانية	4.10^{16} سنة
1,000	.033 ثانية	.45 ثانية	13 ثانية	.94 ساعة	.
10,000	.33 ثانية	6.1 ثانية	22 دقيقة	39 يوما	.
100,000	3.3 ثانية	1.3 دقيقة	1.5 يوما	108 سنة	.
الوقت المسموح به	أكبر حجم مدخلات (تقريبا) قابل للحل				
ثانية واحدة	30,000	2,000	280	67	20
دقيقة واحدة	1,800,000	82,000	2,200	260	26

جدول 2-1

كيفية نمو الدوال How functions grow

وأما الجزء الأسفل من الجدول (جدول 2-1) فينظر في تأثير معدل النمو المقارِب على الزيادة في حجم المدخلات التي يمكن معالجتها (handled) بإعطاء فترة زمنية أكبر للحاسوب (more computer time) [أو باستخدام حاسوب أسرع (a faster computer)]. فليس من الصحيح عموما أننا إذا ضربنا الفترة الزمنية (أو السرعة) في 60 مثلا، فإننا نستطيع معالجة مدخلات أكثر تُقَدَّر ب 60 مرة قدر المدخلات الحالية. وهذا يُعَدُّ صحيحا فقط بالنسبة للخوارزميات التي درجة تعقيدها في $O(n)$ (complexity in). وأما الخوارزمية $\Theta(n^2)$ مثلاً فيمكنها معالجة مدخلات أكثر تُقَدر بـ $\sqrt{60}$ مرة فقط قدر المدخلات الحالية.

ولبيان أن الرتبة المقارِبة لوقت تنفيذ خوارزمية تُعَدُّ أكثر أهمية من معامل ثابت (a constant factor) [لمدخلات كبيرة (for large inputs)] انظر جدول 2-2. وقد كُتِب برنامج للخوارزمية التكعيبية من جدول 2–1 للحاسوب "Cray–1 super computer"، وتم تشغيله لفترة زمنية $3n^3$ نانو ثانية (it ran in $3n^3$ nanoseconds) لمدخلات حجمها n (of size). كما تمت برمجة الخوارزمية الخطية (linear algorithm) على حاسوب "80-TRS" [وهو حاسوب شخصي متواضع السعر من الثمانينات] وتم تشغيل البرنامج لفترة زمنية $19.5n$ ميللي ثانية (milliseconds) [أي n 19,500,000 نانو ثانية]. ورغم أن الثابت في حالة الخوارزمية الخطية أكبر من ثابت الخوارزمية التكعيبية (cubic algorithm) بقدر 6.5 مليون مرة إلا أن الخوارزمية الخطية أسرع بالنسبة للمدخلات التي حجمها for input sizes) $n \geq 2500$). [ما إذا كنا سنعتبر حجم المدخلات هذا كبيرا أم صغيرا يعتمد على سياق المسألة].

N	Cray-1 Fortran $3n^3$ نانو ثانية	TRS-80 Basic 19,500,000 n نانو ثانية
10	3 ميكرو ثانية	.2 ثانية
100	3 ميللي ثانية	2.0 ثانية
1,000	3 ثانية	20.0 ثانية
2,500	50 ثانية	50.0 ثانية
10,000	49 دقيقة	3.2 دقيقة
1,000,000	95 سنة	5.4 ساعة

جدول 2-2

الرتبة المقارِبة أكثر أهمية Asymptotic order wins out

وإذا ركّزنا اهتمامنا على الرتبة المقارِبة للدوال [وبالتالي نشمل n (include) و 10^6 n مثلاً في الطائفة / الطبقة نفسها (in the same class)]، فإننا حين يمكننا بيان أن دالتين ليستا من الرتبة نفسها فإننا نقرر حينئذ عبارة هامة حول الفارق بين الخوارزميتين الموصوفتين بهاتين الدالتين. وإذا كانت هناك دالتان من الرتبة نفسها (same order)، فقد تختلفان بمعامل ثابت كبير. إلا أن قيمة الثابت لا علاقة لها بتحديد تأثير حاسوب أسرع على أقصى حجم مدخلات (maximum input size) يمكن لخوارزمية أن تعالجه / تنفذه / تتعامل معه (can handle) في فترة زمنية معطاة (a given amount of time). أي أن قيمة الثابت لا علاقة لها بالزيادة بين آخر صفين في جدول 2-1. والآن ننظر نظرة أعمق في دلالة هذه الأرقام.

نفرض أننا سنختار فترة زمنية محددة ونُثَبِّتَها (fix it) (مثلاً ثانية واحدة، أو دقيقة واحدة، . . . الاختيار لا يهم). ونفرض أن s ترمز إلى أكبر

حجم مدخلات يمكن لخوارزمية معينة أن تعالجه خلال الفترة الزمنية المحددة المختارة. والآن نفرض أننا قد سمحنا بفترة زمنية أطول تساوي أضعاف الفترة المختارة t مرة (أو أن سرعة حاسوبنا قد زادت إلى أضعاف قيمتها t مرة، إما بسبب تقدم التكنولوجيا أو ببساطة لأننا قد اشترينا جهازا أغلى ثمنا). جدول 2–3 يبيِّن تأثير التسريع (زيادة السرعة) (speedup) لعدة درجات تعقيد.

عدد الخطوات التي تم إنجازها على مدخلات حجمها n	أكبر حجم مدخلات يمكن تنفيذه	أكبر حجم مدخلات يمكن تنفيذه بعد زيادة الفترة الزمنية إلى أضعاف قيمتها t مرة
$f(n)$	S	S_{new}
$lg\ n$	s_1	s_1^t
n	s_2	$t\ s_2$
n^2	s_3	$\sqrt{t}\ \ s_3$
2^n	s_4	$s_4 + 1g\ t$

جدول 2-3

تأثير زيادة سرعة الحاسوب على أكبر حجم مدخلات يمكن تنفيذه

Effect of increased computer speed on maximum feasible input size

القيم الموجودة في العمود أقصى اليمين يتم حسابها بملاحظة أن عدد الخطوات بعد التسريع يساوي:

$$f(s_{new}) = \text{number of steps after speedup}$$
$$= t\ .\ (\text{number of steps before the speedup}) = t\ f(s)$$

وحل المعادلة

$$f(s_{new}) = t\ f(s)$$

للحصول على s_{new}.

والآن إذا ضربنا الدوال الموجودة في العمود أقصى اليسار بثابتٍ ما c، فإن القيم الموجودة في العمود أقصى اليمين سوف لا تتغير! وهذا هو ما قصدناه حين قلنا إن الثابت لا علاقة له بتأثير زيادة الفترة الزمنية أو سرعة الحاسوب على أكبر حجم مدخلات يمكن تنفيذه بخوارزميةٍ ما.

خواص المجموعات Properties of O, Ω, Θ

مجموعات الرتب (order sets) لها عدة خواص مفيدة نذكر بعضها فيما يلي، وهي تنتج بسهولة من تعريفات المجموعات، ولذلك سنترك معظم براهين هذه الخواص كتدريب للقارئ الكريم. ولكل تلك الخواص نفرض أن $f, g, h: \mathbf{N} \to \mathbf{R}^*$. أي أن الدوال تسقط (map) الأعداد الصحيحة غير السالبة إلى الأعداد الحقيقية غير السالبة.

تمهيدية 2-5:

إذا كانت $f \in O(g), \quad g \in O(h)$ فإن $f \in O(h)$. أي أن O متعدية (transitive). وكذلك فإن $\Omega, \Theta, o, \omega$ جميعها متعدية.

البرهان: نفرض أن لدينا c_1, n_1 بحيث أن

$$f(n) \le c_1 g(n) \quad \forall\ n \ge n_1$$

وكذلك نفرض أن لدينا c_2, n_2 بحيث أن

$$g(n) \le c_2 h(n) \quad \forall\ n \ge n_2$$

وبناءً عليه فإنه

$$\forall n \ge \max(n_1, n_2),\ f(n) \le c_1 c_2 h(n)$$

وهكذا فإن $f \in O(h)$.

وأما البراهين بالنسبة للمجموعات $\Omega, \Theta, o, \omega$ فهي شبيهة بهذا البرهان.

تمهيدية 2-6:

1) $f \in O(g)$ إذا وفقط إذا كان $g \in \Omega(f)$.

2) إذا كانت $f \in \Theta(g)$ فإن $g \in \Theta(f)$.

3) Θ تُعَرِّف علاقة تكافؤ (an equivalence relation) على الدوال. وكل مجموعة $\Theta(f)$ هي طبقة تكافؤ (an equivalence class) نطْلِق عليها طبقة تعقيد (a complexity class).

4) $O(f + g) = O(\max(f, g))$. و هناك معادلتان مماثلتان متحققتان بالنسبة للمجموعتين Θ ,Ω. [هذه المعادلات تفيدنا حين تحليل خوارزميات معقدة (complex)، حيث f, g قد تصفان الشغل المبذول بأجزاء مختلفة من الخوارزمية].

ونظرا لأن Θ تُعَرِّف علاقة تكافؤ، فيمكننا الإشارة إلى طبقة تعقيد خوارزمية ما بتحديد أي دالة في الطبقة. وعادة نختار أبسط دالة [كممثِّلة (representative) للطبقة]. فمثلا إذا كانت الدالة $f(n) = n^3/6 + n^2 + 1g\ n + 12\ 2$ تصف عدد الخطوات التي تنفذها خوارزمية ما، فإننا نقول ببساطة إن درجة تعقيد الخوارزمية (complexity of the algorithm) تقع / توجد في $\Theta(n^3)$. وإذا كانت $f \in \Theta(n)$ فإننا نقول إن f خطية (linear). وإذا كانت $f \in \Theta(n^2)$ قلنا إن f تربيعية (quadratic). وإذا كانت $f \in \Theta(n^3)$ قلنا إن f تكعيبية (cubic)[*]. والاصطلاح $O(1)$ يرمز إلى مجموعة الدوال المحدودة (bounded) بثابت (لقيم n الكبيرة).

---

[*] لاحظ أن المصطلحات "خطية وتربيعية وتكعيبية" تستخدم هنا بطريقة فضفاضة أوسع من استخدامها في الرياضيات.

وفيما يلي نظريتان مفيدتان، ويمكن برهنتهما باستخدام قاعدة "هوبيتال"

نظرية 2-1:

$\lg n$ تنتمي إلى $o(n^{\alpha})$ وذلك لأي $\alpha > 0$. أي أن دالة log تنمو أبطأ من أي قوة موجبة (أُس موجب) (positive power) من قوى n [بما في ذلك القوى الكسرية (fractional powers)].

نظرية 2-2:

n^k تنتمي إلى $o(2^n)$ وذلك لأي $k>0$. أي أن قوى n (powers of) تنمو أبطأ من الدالة الأسِّيَّة 2^n. [في الحقيقة قوى n تنمو أبطأ من أي دالة أسِّيَّة c^n حيث $c>1$].

الرتبة المقارِبة للمجاميع الشائعة

The Asymptotic Order of Commonly Occurring Sums

اصطلاح الرتبة يُسَهِّل علينا اشتقاق وتذكُّر الرتبة المقارِبة لكثير من المجاميع التي تظهر مرارا عند تحليل الخوارزميات. وقد تم تعريف بعض هذه المجاميع في الفصل الأول.

نظرية 2-3:

نفرض أن d ثابت غير سالب، وأن r ثابت موجب لا يساوي 1.

1) مجموع أي "متسلسلة حدودية" (a polynomial series) يزيد الأس(the exponent) بواحد 1. [تذكر أن المتسلسلة

الحدودية من الدرجة d هي مجموع صيغته $\sum_{i=1}^{n} i^d$]. القاعدة هي أن هذا النوع من المجاميع ينتمي إلى $\Theta(n^{d+1})$.

2) مجموع أي "متسلسلة هندسية" (a geometric series) ينتمي إلى Θ الخاصة بأكبر حدودها (in Θ of its largest term). [تذكَّر أن المتسلسلة الهندسية هي مجموع صيغته $\sum_{i=a}^{b} r^i$].

وتنطبق هذه القاعدة سواء كان $0 < r < 1$ أو $r > 1$، ولكن - كما هو واضح - ليس عندما $r = 1$. والنهايتان a, b (limits) ليستا ثابتين معاً (are not both constants). وعادةً تكون النهاية العليا b (upper limit) دالةً ما في n، والنهاية السفلى a (lower limit) ثابتا.

3) مجموع أي متسلسلة لوغاريتمية (a logarithmic series) ينتمي إلى Θ الخاصة بحاصل ضرب عدد الحدود ولوغاريتم أكبر حد.

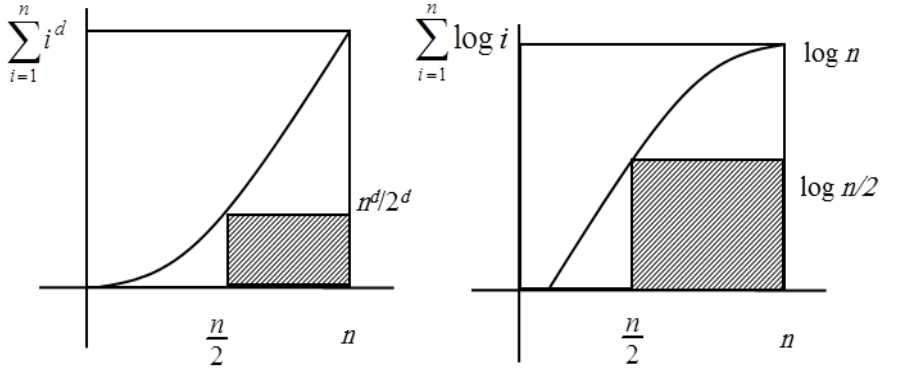
[in Θ (the number of terms times the log of the largest term)].

[المتسلسلة اللوغاريتمية هي مجموع صيغته $\sum_{i=1}^{n} \log i$].

وتنص القاعدة على أن هذا النوع من المجاميع ينتمي إلى $\Theta(n \log(n))$. وتذكر أنه بالنسبة للعبارات التي تتناول الرتبة المقارِبة فإن أساس اللوغاريتم لا يهم.

4) مجموع أي متسلسلة لوغاريتمية حدودية (a polynomial–logarithmic series) [والتي هي عبارة عن مجموع صيغته $\sum_{i=1}^{n} i^d \lg i$] ينتمي إلى $\Theta(n^{d+1} \log n)$.

البرهان: انظر إلى شكل 2-3.



شكل 2-3

المستطيلات تعطي الحدود العليا والحدود السفلى لأنواع كثيرة من المجاميع.
عندما يكون لمساحتي المستطيلين الرتبة المقارِبة نفسها، هذه الرتبة يجب أن تكون رتبة المجموع

نظراً لأن جميع المتسلسلات في النظرية صيغتها $\sum_{i=1}^{n} f(i)$ **، حيث** $f(i)$ **دالة رتيبة** (monotonic)**، فمن الواضح أن المستطيل الأكبر الذي ارتفاعه** (height) $f(n)$ **وعرضه** n (width) **هو حد علوي** (upper bound) **للمجموع . وأيضًا - كما رأينا سابقا في شكل 1-2- ب (في الفصل الأول) - فإن المساحة الواقعة تحت منحنى الدالة** $f(i)$ **بين** $i = 0$ **و** $i = n$ **هو حد سفلى** (a lower bound) **للمجموع. وبالنسبة لحالتي المتسلسلة الحدودية والمتسلسلة اللوغاريتمية فإن هذه المساحة يمكن أن تحدَّها بسهولة من أسفل مساحة المستطيل الأصغر داكن / غامق الظل** (darkly shaded)**. وفي الصورة اليسرى في شكل 2–3 نجد أن مساحة المستطيل الأكبر تساوي** n^{d+1}**، بينما مساحة المستطيل الأصغر تساوي** $n^{d+1}/2^{d+1}$**. ونظرا لأن مساحتي المستطيلين**

لهما الرتبة المقارِبة نفسها (same asymptotic order)، فلذلك يجب أن يكون للمجموع هذه الرتبة أيضا . وفي الصورة اليمنى في شكل 2-3 نجد أن المساحتين هما:

$n \log n$, $(n/2)(\log n - \log 2)$. وحالة المتسلسلات اللوغاريتمية الحدودية شبيهة بما سبق. ولكن هذه الطريقة (technique) لا تصلح للمتسلسلات الهندسية. وأما القاعدة بالنسبة لهذه المتسلسلات الهندسية فتنتج مباشرة من المعادلة (9) في الفصل الأول.

البحث في منظومة مرتبة Searching an Ordered Array

لتوضيح الأفكار التي سبق ذكرها في هذا الفصل ندرس المسألة التالية، وهي مسألة مشهورة ومعتادة.

المسألة 2-1 البحث في منظومة مرتبة Ordered array search

نفرض أننا قد أُعطينا منظومة E مكونة من عناصر عددها n مُرَتَّبَة ترتيبا غير تناقصي (n entries sorted in non-decreasing order). ونفرض كذلك أننا قد أُعطينا قيمة K.

المطلوب: إيجاد قيمة مؤشر index تتحقق عنده العلاقة $K = E[\text{index}]$، أو إذا لم تكن القيمة K موجودة بالمنظومة نعيد القيمة -1 باعتبارها الإجابة المطلوبة.

عمليا تكون K عادةً المفتاح **key** لعنصرٍ ما (an entry)، وتكون العناصر في طبقةٍ ما (in some class) مع مجالات أخرى (with other instance fields) بجانب المفتاح، وبالتالي فقد يكون المتطلب الأكثر دقة هو: $K=E[\text{index}].\text{key}$. ولتبسيط المناقشة نفترض أن عنصر المنظومة كاملا هو المفتاح، وأنه من النوع العددي (numeric type).

ونفرض / نَدَّعي الآن أننا لا نعرف خوارزمية البحث الثنائي، ونحاول حل المسألة كأننا نراها لأول مرة. وسندرس عدة خوارزميات مختلفة، ونقوم بتحليل سلوك أسوأ حالة، والسلوك المتوسط، وأخيرا ندرس خوارزمية البحث الثنائي، ونثبت أنها خوارزمية مُثْلَى وذلك بتعيين / بتحديد حد أسفل (establishing a lower bound) للعدد المطلوب من المقارنات مع المفتاح (number of key comparisons needed).

بعض الحلول Some Solutions

لاحظنا أن خوارزمية البحث التتابعي (الخوارزمية 2–1) تحل المسألة، ولكنها لا تستفيد من حقيقة أن المنظومة الآن تحتوي على عناصر مرتبة. فهل يمكننا تعديل هذه الخوارزمية بحيث تستفيد من هذه المعلومة الإضافية وتقلل الجهد المبذول؟

لا شك أن أول تحسين للخوارزمية يمكن أن يخطر على بالنا هو أنه: نظرًا لأن المنظومة مُرَتَّبَة ترتيبا غير تناقصي، فبمجرد أن يقابلنا عنصر أكبر من K، فيمكن للخوارزمية حينئذ أن تتوقف (terminate) وتعطي الإجابة 1-. [كيف يجب تغيير الاختبار / الشرط الموجود في سطر 4 في خوارزمية 2-1 لتجنب (avoiding) إجراء مقارنتين في كل مرور (on each pass) خلال العروة؟]. كيف يؤثِّر هذا التغيير على التحليل (analysis)؟ من الواضح أن الخوارزمية المُعَدَّلة ستكون أفضل في بعض الحالات، وسوف تتوقف أسرع بالنسبة لبعض المدخلات. إلاَّ أن درجة التعقيد في أسوأ حالة ستظل كما هي دون تغيير لأنه إذا كانت K هي آخر عنصر في المنظومة، أو إذا كانت K أكبر من جميع العناصر فإن الخوارزمية ستُجرِي عدد n من المقارنات.

وبالنسبة للتحليل المتوسط (average analysis) للخوارزمية المعدَّلة يجب أن نعلم مدى احتمال وجود K بين أي عنصرين في المنظومة. نفرض أننا سنُعَرِّف **فجوة (a gap)** g_i على أنها مجموعة القيم y التي تحقق الشرط:

$$E[i-1] < y < E[i],\ i = 1, \ldots, n-1.$$

ونفرض أيضا أن g_0 تمثل جميع القيم الأصغر من $E[0]$، وأن g_n تمثل جميع القيم الأكبر من $E[n-1]$. ونفرض (كما فعلنا في مثال 2-1) أن هناك احتمالاً يساوي q لوجود K في المنظومة. وإذا كانت K في المنظومة E فنفرض أن جميع المواضع في المنظومة متساوية الاحتمالات [أي أن احتمالها الشَّرطِي (conditional probability) يساوي $1/n$]. وإذا لم تكن K في المنظومة فنفرض أن جميع الفجوات متساوية الاحتمالات [أي أن احتمالها الشَّرطِي يساوي $1/(n+1)$]. وبالنسبة لقيم i حيث $0 \le i < n$ نحتاج إلى مقارنات (comparisons) عددها $i+1$ لتحديد أن $K=E[i]$ أو أن K موجودة في g_i. ونحتاج إلى مقارنات عددها n لتحديد أن K موجودة في g_n. وبناءً عليه نحسب متوسط عدد المقارنات المشروطة بالنجاح (conditioned on success) "A_{succ}"، ومتوسط عدد المقارنات المشروطة بالفشل (conditioned on failure) "A_{fail}" كما يلي:

$$A_{succ}(n) = \sum_{i=0}^{n-1}\left(\frac{1}{n}\right)(i+1) = \frac{n+1}{2},$$

$$A_{fail}(n) = \sum_{i=0}^{n-1}\left(\frac{1}{n+1}\right)(i+1) + \left(\frac{1}{n+1}\right)n.$$

المعادلة الأولى تقابل حالات تكون فيها K موجودة في المنظومة، وهي نفسها المعادلة الموجودة في مثال 2-1. والمعادلة الثانية تقابل حالات تكون فيها K غير موجودة في المنظومة. وإيجاد قيمة المجموع أمر سهل بإذن الله ونتركه

كتدريب للقارئ الكريم. وكما عملنا في مثال 2-1 فإننا نجمع (combine) هنا بين النتائج والمعادلة $A(n) = q\ A_{succ}(n) + (1 - q)\ A_{fail}\ (n)$. والنتيجة هي أن $A(n)$ تساوي تقريبا $n/2$ بغض النظر عن قيمة q. ونلاحظ أن خوارزمية 2-1 (البحث التتابعي غير المُرَتَّب) كانت قد أَجْرَت في المتوسط $3n/4$ مقارنة عندما كانت $q = \frac{1}{2}$، وبناءً عليه فإن الخوارزمية المُعَدَّلة (modified algorithm) تُعَدُّ تحسينا (an improvement)، رغم أن سلوكها المتوسط (average behavior) لا يزال خَطِّيًّا (linear).

دعنا نحاول مرة أخرى. هل يمكننا أن نجد خوارزمية تقوم فعليا بمقارنات عددها أقل من n مقارنة في أسوأ حالة؟ نفرض أننا سنقارن K مع كل رابع عنصر - مثلاً - في المنظومة. وإذا حدث توافق (a match) عند أي مقارنة فقد انتهى عمل الخوارزمية ووصلنا إلى الموضع / المؤشر المطلوب، وهو الإجابة المطلوبة. وإذا كانت قيمة K أكبر من قيمة العنصر المقارَن وليكن $\mathrm{E}[i]$، فمعنى هذا أننا لا نحتاج لاختبار العناصر الثلاثة السابقة للعنصر $\mathrm{E}[i]$ لأنها جميعها ليست أكبر من $\mathrm{E}[i]$. وإذا كانت $K < \mathrm{E}[i]$، فمعنى ذلك أن K تقع بين آخر عنصرين تمت مقارنتهما مع K وتكفينا مقارنات قليلة (كم؟) أكثر لتحديد موضع K إن كانت في المنظومة أو لتقرير أنها ليست في المنظومة. ونترك تفاصيل الخوارزمية وتحليلها كتدريب للقارئ الكريم، ولكن من السهل أن نرى أننا نحتاج لاختبار رُبْع عناصر المنظومة فقط تقريبا. وبالتالي ففي أسوأ حالة يتم تنفيذ $n/4$ مقارنة تقريبًا.

ويمكننا تنفيذ هذه الطريقة [وهي مقارنة K مع كل j-th عنصر، حيث $j = 4$]، ولكن باختيار قيمة كبيرة للعدد الصحيح j (أكبر من 4)، وبالتالي نسمح بحذف عناصر / مفاتيح (keys) عددها $j-1$ مِنْ أَخْذِها في الاعتبار عند كل مقارنة ونحن مستمرون في التقدم عَبْر المنظومة. وهكذا نقوم بإجراء

مقارنات عددها يساوي تقريبا n / j لتحديد جزء صغير من المنظومة (a small section of) E قد يحتوي على K. ثم نُتْبِع هذا بحوالي j مقارنة لاجتياز وفحص هذا الجزء الصغير. ولأي قيمة ثابتة محدَّدة j (fixed) فإن الخوارزمية ستظل خطية (linear) ، ولكننا إذا اخترنا j بحيث أنها تجعل $(n/j + j)$ أقل ما يمكن (minimum)، فإننا نجد بالاشتقاق (بالتفاضل) أن j تساوي $\sqrt{n}$. وبالتالي فإن تكاليف البحث الكلية (total search cost) تساوي $2\sqrt{n}$ فقط. وهكذا فقد كسرنا حاجز الزمن الخَطِّي (linear – time barrier).

ولكن هل يمكننا تحسين الخوارزمية للوصول إلى نتائج أفضل؟ لاحظ أننا قد غَيَّرنا طريقة البحث بعد تحديد الجزء الصغير. فهذا الجزء يحتوي على نحو j عنصر، وقد تكلفنا j لاستكشافه (explore it)، وهي تكاليف خطية (linear cost). ولكننا الآن نعلم أن التكاليف الخطية تكاليف باهظة. ولذلك يجب علينا أن نطبق ارتداديا (recursively) طريقتنا الأساسية على الجزء الصغير بدلا من تغيير الطريقة.

وفكرة خوارزمية البحث الثنائي (Binary Search algorithm) هي أن تأخذ التعبير "كل j-th عنصر" إلى أقصاه المنطقي بالقفز عبر نصف العناصر في خطوة واحدة. وبدلا من اختيار عدد صحيح معيَّن j ومقارنة K مع كل j-th عنصر، فإننا نقارن K أولا مع العنصر الموجود في منتصف المنظومة. وهذا يلغي نصف المفاتيح بمقارنة واحدة.

وبمجرد أن حَدَّدْنا أي نصفي المنظومة قد يحتوي على K، فإننا نطبق الطريقة نفسها ارتداديا (recursively) ونستمر في مقارنة K مع العنصر الموجود في وسط جزء المنظومة (section of the array) قيد البحث والاعتبار، إلى أن تنكمش (shrink) سعة الجزء الذي قد يحتوي على K إلى الصفر (zero size)، أو إلى أن نجد K في المنظومة. وبعد كل مقارنة فإن سعة / حجم (size) جزء المنظومة الذي قد يحتوي على K تقل إلى النصف. ولاحظ

كذلك أن هذا هو مثال آخر لطريقة بحث مُعَمَّمَة (a generalized searching routine) [انظر تعريف 2–4]، حيث يفشل الإجراء (procedure *fails*) إذا انكمشت سعة جزء المنظومة الذي قد يحتوي على K إلى الصفر، وينجح (*succeeds*) إذا حَدَّد موضع K، ويستمر في البحث (*keeps searching*) إذا لم يتحقق أيٌّ من هذين الحدثين.

وهذا الإجراء هو مثال أَوَّلي (a prime example) **لنموذج / لطريقة "قَسِّمْ وتَغَلَّبْ / فَرِّق تَسُدْ" (divide and conquer paradigm / method)**، والتي سنناقشها بتفصيل أكثر بإذن الله في الفصل القادم. فمسألة إيجاد K من بين عناصر مرتبة عددها n تُقَسَّم إلى مسألتين فرعيتين بمقارنة K مع العنصر الأوسط(middle element) [بفرض أن العنصر الأوسط ليس K]. وسنرى بإذن الله بالتحليل أن حل المسألتين الفرعيتين منفصلتين (separately) أسهل (في أسوأ حالة وفي الحالة المتوسطة) من حل المسألة الأصلية دون تقسيمها (dividing it up). وفعليا إحدى المسألتين الفرعيتين تُحَلُّ بِجُهْد يساوي صفرا لأننا نعلم أن K لا يمكن أن تكون في هذا الجزء من المنظومة.

أگوارزميث 2-4 : البحث الثنائي Binary Search

المدخلات (Input): E, **first**, **last**, K

حيث: E : منظومة مرتبة في المدى (range) **first**, …, **last**

K : المفتاح (key) الذي نبحث عنه .

وللتبسيط نفرض أن K, **first**, **last** وعناصر المنظومة E جميعها أعداد صحيحة.

المخرجات (Output): **index** بحيث أن:

E[index] = ***K*** إذا كانت ***K*** موجودة في المنظومة ***E*** في المدى first, …, last، index = –1 إذا لم تكن ***K*** موجودة في ***E*** في هذا المدى.

```
int binarySearch ( int[ ] E, int first, int last, int K )
1.  if ( last < first )
2.      index = –1;
3.  else
4.      int mid = (first + last) / 2;
5.      if ( K == E[mid] )
6.          index = mid;
7.      else if (K < E[mid])
8.          index = binarySearch (E, first, mid–1, K);
9.      else
10.         index = binarySearch (E, mid + 1, last, K);
11. return index;
```

تحليل أسوأ حالة لخوارزمية البحث الثنائي

Worst–Case Analysis of Binary Search Algorithm

سنُعَرِّف حجم مسألة (problem size) البحث الثنائي بأنها n = last– first+1، وهي عدد العناصر في مدى المنظومة ***E*** الذي نبحث فيه. والمقارنة بين K وعنصر في المنظومة (an array entry) تُعَدُّ اختيارا مناسبا للعملية الأساسية لخوارزمية البحث الثنائي. [وكلمة "مقارنة" (a comparison) في

المناقشة الحالية ستعنى دائما مقارنة مع عنصر في المنظومة E، وليس مقارنة مؤشِّر (an index comparison) كتلك المقارنة التي تظهر في السطر line 1 في الخوارزمية]. ونفرض أن $W(n)$ هي عدد المقارنات التي تنفذها الخوارزمية في أسوأ حالة على المنظومات –التي تحتوي كل منها على n عنصر– في المدى الذي نبحث فيه.

ومن المعتاد أن نفرض أننا نُنَفِّذ مقارنة واحدة ثلاثية التفرع (one comparison with a three–way branch) للاختبارين (tests) على K في السطرين 5 و 7. [وحتى بدون المقارنات ثلاثية التفرع (three–way comparisons) يمكننا الوصول إلى الحد نفسه (same bound) بالمقارنات الثنائية (binary comparisons) {أي المقارنات التي تعيد نتيجة بُولية a Boolean result}]. وهكذا فإن $W(n)$ هي أيضا عدد مرات استدعاء الدالة **binarySearch** عدا المرة التي تصل فيها السطر line 2 وتخرج (exits) دون مقارنة.

نفرض أن $n > 0$. مهمة الخوارزمية هي أن تجد K في مدى عناصر عددها n ومُرَقَّمَة / مرتَّبة / مؤشَّرة (indexed) من **first** إلى **last**. تتجه الخوارزمية أولا إلى سطر line 5 وتقارن K مع $E[\text{mid}]$، حيث $\text{mid} = \lfloor (\text{first} + \text{last})/2 \rfloor$. في أسوأ حالة يكون هذان المفتاحان غير متساويين، ونصل إما إلى سطر 8 أو إلى سطر 10 حسب احتمال احتواء الجزء الأيسر أو الجزء الأيمن (بالنسبة للقيمة mid) على K. كم عدد العناصر الموجودة في كل من هذين الجزئين؟ إذا كانت n زوجية فهناك $n/2$ عنصر في الجزء الأيمن من المنظومة، و $(n/2) - 1$ عنصر في الجزء الأيسر. وإذا كانت n فردية فهناك $(n - 1)/2$ عنصر في كل من الجزئين. وبالتالي

فهناك على الأكثر $\lfloor n/2 \rfloor$ عنصر في جزء المنظومة الذي يُعَيَّن للاستدعاء الارتدادي. ولذلك فيمكننا اعتبار أن حجم المدى (size of the range) يُقسم على 2 مع كل استدعاء ارتدادي تقديرًا حَذِرا / معتدلاً (a conservative estimate).

كم مرة يمكننا تقسيم n على 2 دون أن نحصل على نتيجة أقل من 1؟ و بكلمات أخرى ما هي أكبر قيمة للعدد الصحيح d بحيث أن $n/2^d \geq 1$؟ إذا قمنا بحل هذه المتباينة للحصول على d نحصل على $2^d \leq n$، و بالتالى $d \leq \lg (n)$. و هكذا يمكننا إجراء $\lfloor \lg(n) \rfloor$ مقارنة عقب الاستدعاءات الارتدادية، و مقارنة واحدة قبل أي استدعاء ارتدادي. وبالتالي فمجموع المقارنات على الأكثر يساوي $\lfloor \lg(n) \rfloor + 1$ (at most) $W(n)$. وباستخدام المتطابقة $\lfloor \lg n \rfloor + 1 = \lceil \lg (n+1) \rceil$ للأعداد الصحيحة $n \geq 1$، نحصل على صيغة أكثر ملاءمة قليلا للعدد الإجمالي للمقارنات على الأكثر، وهي الصيغة $W(n) = \lceil \lg (n+1) \rceil$، وهي مُعَرَّفة جيدا في حالة $n = 0$. وهكذا أثبتنا النظرية التالية:

نظرية 2-4:

خوارزمية البحث الثنائي تُنَفِّذ $W(n) = \lceil \lg (n+1) \rceil$ مقارنةً بين K وعناصر المنظومة في أسوأ حالة (حيث $n \geq 0$ هي عدد عناصر المنظومة). ونظرا لأنه يتم تنفيذ مقارنة واحدة مع كل استدعاء للدالة، فلذلك يُعَدُّ وقت التنفيذ / التشغيل (running time) في $\Theta(\log n)$.

وتجدر الإشارة إلى أن خوارزمية البحث الثنائي تُنَفِّذ مقارنات أقل في أسوأ حالة مما تنفذه خوارزمية البحث التتابعي (sequential search) في المتوسط.

تحليل السلوك المتوسط Average–Behavior Analysis

لتبسيط التحليل قليلا سنفرض أن K تظهر على الأكثر في موضع واحد في المنظومة. وكما لاحظنا سابقا فهناك $2n+1$ موضعاً محتملاً يمكن أن تشغله K: n موضعا في المنظومة E –وهذه سنطلق عليها مواضع النجاح (**success positions**)- و $n+1$ فجوة (gap) - وهذه نطلق عليها مواضع الفشل (**failure positions**) - . وبالنسبة للمدى $0 \le i < n$ نفرض أن I_i تمثل جميع المدخلات التي تحقق العلاقة $K=E[i]$. وبالنسبة للمدى $1 \le i < n$ نفرض أن I_{n+i} تمثل المدخلات التي تحقق العلاقة $E[i-1] < x < E[i]$. I_n تمثل المدخلات التي تحقق $K < E[0]$، و I_{2n} تمثل المدخلات التي تحقق $K > E[n-1]$. نفرض أن $t(I_i)$ هي عدد مقارنات K مع عناصر المنظومة E والتي تجريها الخوارزمية 2–4 على المدخلات I_i. الجدول التالي (جدول 2–4) يبيِّن قيم t وذلك لعدد العناصر $n = 25$.

موضع K i	عدد المقارنات $t(I_i)$		i	$t(I_i)$
0	4		13	4
1	5		14	5
2	3		15	3
3	4		16	4
4	5		17	5
5	2		18	2
6	4		19	4
7	5		20	5
8	3		21	3
9	5		22	5
10	4		23	4
11	5		24	5
12	1		gaps 25, 28, 31, 38, 41, 44 الفجوات	4
			جميع الفجوات الأخرى	5

جدول 2- 4

عدد المقارنات التي تنفذها خوارزمية البحث الثنائي

المقابلة لمواضع K المختلفة ، وذلك لعدد العناصر $n = 25$

The number of comparisons done by Binary Search,
depending on the location of K, for $n = 25$

لاحظ أن معظم النجاحات (most successes) (مواضع النجاح)، وجميع الفجوات (all gaps) تقع في مدى واحدة من أسوأ الحالات، أي أنها تستغرق / تتطلب 4 إلى 5 مقارنات لتجد K معظم الوقت. [في الحالة $n = 31$ سنجد أن معظم النجاحات وجميع الفجوات هي بالضبط أسوأ حالة]. وبالتالي فإذا فرضنا أن جميع مواضع النجاح متساوية الاحتمالات (equally likely)، فمن المعقول أن نتوقع أن عدد المقارنات التي يتم تنفيذها في المتوسط تكون قريبة من $\lg n$. وحساب المتوسط بفرض أن احتمال أي موضع هو 1/51 يُعطِي 223/51، أي تقريبا 4.37 و $4.65 \approx \lg 25$.

ونظرا لأن متوسط عدد المقارنات قد يعتمد على احتمال أن يكون البحث ناجحا، دعنا نرمز إلى هذا الاحتمال بالرمز q، ونُعرِّف $A_q(n)$ بأنها متوسط عدد المقارنات عندما يكون احتمال النجاح هو q. ونحصل بقانون التوقعات المشروطة (conditional expectations) [التمهيدية 1–2] على:

$$A_q(n) = q\, A_1(n) + (1-q)\, A_0(n)$$

ولذلك يمكننا حل الحالتين الخاصتين: $A_1(n)$ [النجاح أكيد (success is certain)] و $A_0(n)$ [الفشل أكيد (failure is certain)]، ونضيفهما لنحصل على حل لأي q. ولاحظ أن A_1 هي نفسها A_{succ} و A_0 هي نفسها A_{fail} في المصطلحات التي استخدمناها في البحث التتابعي.

وفيما يلي سنشتق صيغتين تقريبيتين للمتوسطين $A_0(n), A_1(n)$ على أساس الفرضين التاليين:

1) جميع مواضع النجاح متساوية الاحتمالات، أي أن

$$Pr(I_i \mid succ) = 1/n; \quad 1 \le i \le n$$

2) لعددٍ صحيحٍ ما $n = 2^k - 1 \qquad ; k \ge 0$

والهدف من الفرض الأخير هو تبسيط التحليل. والنتيجة بالنسبة لجميع قيم n قريبة جدا من النتيجة التي سنحصل عليها بإذن الله.

نظرا لأن $n=2^k-1$، فمن السهل أن نرى أن أي بحث فاشل (failing search) سوف يستخدم بالضبط k مقارنة، بغض النظر عن الفجوة (gap) التي تقع فيها K. ولذلك فإن

$$A_0(n) = \lg(n + 1)$$

المفتاح لتحليل السلوك المتوسط (average behavior) لحالات البحث الناجح (successful searches) هو أن نتحول من التفكير في عدد المقارنات التي تُنَفَّذ على مُدْخَل معيَّن I_i (a particular input) إلى التفكير في عدد المدخلات التي تُنَفِّذ عدداً معيَّناً (a specific number) من المقارنات، وليكن t مقارنة مثلا. لقيم t حيث $1 \le t \le k$ نفرض أن s_t هي عدد المدخلات التي تقوم عندها الخوارزمية بتنفيذ t مقارنة.

مثلا عندما $n = 25$ فإن $s_3 = 4$ لأنه سيتم تنفيذ ثلاث مقارنات لكل من المدخلات الأربعة I_2, I_8, I_{15}, I_{21}.

ومن السهل أن نرى أن $s_1 = 1 = 2^0,\ s_2 = 2 = 2^1,\ s_3 = 4 = 2^2$ وعموما $s_t = 2^{t-1}$. ونظرا لأن احتمال كل مُدْخَل يساوي $1/n$، فإن احتمال أن تنفِّذ الخوارزمية t مقارنة هو s_t/n، والمتوسط (average) هو

$$A_1(n) = \sum_{t=1}^{k} t\left(\frac{s_t}{n}\right) = \frac{1}{n}\sum_{t=1}^{k} t\,2^{t-1} = \frac{(k-1)2^k + 1}{n}$$

وذلك باستخدام المعادلة (12) بالفصل الأول. [ملاحظة : إذا لم نفرض أن $n = 2^k - 1$ فإن قيمة s_t سوف لا تَتْبَع النموذج / النمط (the pattern)، وبعض حالات الفشل (some failures) سوف تستخدم $k - 1$ مقارنة فقط كما هو مبين في جدول 2–4 في الحالة $n = 25$]. والآن نظرا لأن $n + 1 = 2^k$، فإن

$$A_1(n) = \frac{(k-1)(n+1)+1}{n} = lg(n+1) - 1 + O\left(\frac{log\, n}{n}\right)$$

وكما ذكرنا سابقا فإن العلاقة $A_0(n)=\lg(n+1)$ تتحقق بفرض أن K ليست في المنظومة. وهكذا نكون قد أثبتنا النظرية التالية.

نظرية 2-5:

البحث الثنائي (الخوارزمية 2-4) يُنفِّذ تقريبا $\lg(n + 1) - q$ مقارنةً في المتوسط بالنسبة للمنظومات التي تحتوي على n عنصر، حيث q هو احتمال أن يكون البحث ناجحا، وجميع مواضع النجاح تكون متساوية الاحتمالات.

الأمثلية Optimality

رأينا فيما سبق عند البحث في منظومة مرتبة أننا قد بدأنا بخوارزمية $\Theta(n)$، ثم قمنا بتحسينها (improving it) إلى $\Theta(\sqrt{n})$، ومن ثَمَّ إلى $\Theta(\log n)$. فهل يمكننا القيام بتحسينات أكثر؟ وحتى إذا لم نستطع تحسين الرتبة المقارِبة (asymptotic order)، فهل يمكننا تحسين المعامل الثابت (constant factor)؟ دور تحليل الحدود السفلى / الدنيا (lower bounds analysis) أن يخبرنا متى يمكن الإجابة سلباً (negatively) على أحد أو على كلى هذين السؤالين. وأي حد سفلي "مُحْكَم"

(a **tight** lower bound) - وهو حد سفلي يوافق/ يوائم (matches) الحد العلوي (upper bound) لخوارزميتنا - يؤكد لنا أنه لا يمكننا الوصول إلى تحسينات أكثر.

سنُبَيِّن فيما يلي بإذن الله أن خوارزمية البحث الثنائي خوارزمية مُثْلَى (optimal) في طائفة الخوارزميات التي لا تستطيع إجراء أي عمليات أخرى سوى المقارنات على عناصر المنظومات. وسوف نصل إلى حد أدنى (a lower bound) لعدد المقارنات المطلوبة بدراسة / بفحص "أشجار الحسم / القرار" (by examining decision trees) لخوارزميات البحث في هذه الطائفة. نفرض أن A هي إحدى هذه الخوارزميات. أي شجرة حسم لهذه الخوارزمية A وحجم مدخلات مُعْطَى n هي شجرة ثنائية عناصرها / عقدها (nodes) مُعَنْوَنة / مُرقَّمة (labeled) بأرقام بين 0 و $n - 1$، ومرتَّبَة (arranged) بناءً على القاعدتين التاليتين:

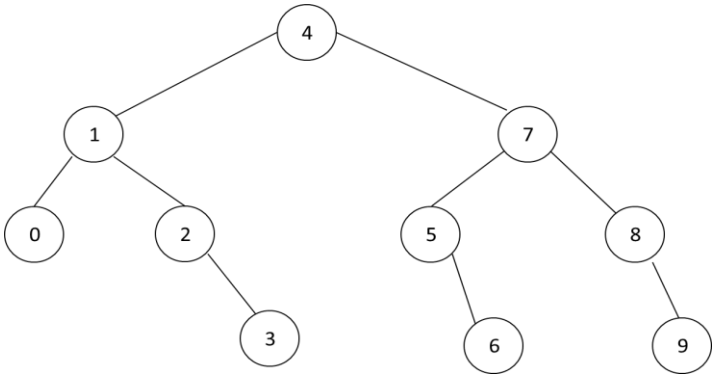
1) جذر (root) الشجرة يُعَنْوَن / يُرَقَّم برقم / بمؤشر (index) أول عنصر في المنظومة التي تقارن الخوارزمية A عناصرها بالقيمة K.

2) نفرض أن الاسم / العنوان (label) على عقدة / عنصر معين (a particular node) هو i. الاسم / العنوان على الابن الأيسر (left child) لهذا العنصر سيكون هو مؤشر (index) العنصر الذي ستقارنه الخوارزمية A بالقيمة K في الخطوة التالية إذا كانت $K < E[i]$. والاسم / العنوان على الابن الأيمن (right child) سيكون هو مؤشر العنصر الذي ستقارنه الخوارزمية بالقيمة K في الخطوة التالية إذا كانت $K > E[i]$. والعقدة / العنصر (node) سوف لا يكون له ابن أيسر (أو أيمن) إذا توقفت الخوارزمية (the algorithm halts) بعد مقارنة K مع $E[i]$ واكتشاف

أن $K < E[i]$ (أو أن $K > E[i]$). ولا يوجد فرع (branch) في الحالة $K = E[i]$. وأي خوارزمية مقبولة سوف لا تقوم بإجراء أي مقارنات أخرى.

وطائفة الخوارزميات التي يمكن عمل نماذج لها (can be modeled) بهكذا أشجار حسم كبيرة جدا (very broad)، وتشمل خوارزميات البحث التتابعي والتعديلات/ التغييرات (variations) التي أخذناها في الاعتبار سابقا. [لاحظ أن الخوارزمية مسموح لها أن تقارن بين مفتاحين (two keys) في المنظومة، ولكن هذا لا يعطينا أي معلومات، لأن المنظومة مُرَتَّبة فعلا (already sorted) ، وبالتالي لا نضيف أي عقدة / عنصر (node) في شجرة الحسم لهذه الحالة]. والشكل التالي (شكل 2-4) يبيِّن شجرة الحسم لخوارزمية البحث الثنائي حيث n = 10.

إذا أَعطينا الخوارزمية A مُدْخَلاً معينا (a particular input) فإنها ستقوم بإجراء المقارنات المشار إليها عبر مسار واحد (one path) يبدأ عند جذر شجرة الحسم الخاصة بها.



شكل 2- 4

شجرة الحسم لخوارزمية البحث الثنائي حيث $n = 10$

Decision tree for the Binary Search algorithm with $n = 10$

وعدد المقارنات بين المفاتيح (key comparisons) التي سيتم إجراؤها (performed) هي عدد العقد / العناصر الموجودة على المسار. وعدد المقارنات في أسوأ حالة هو عدد العقد على أحد أطول المسارات (number of nodes on a longest path) من الجذر إلى ورقة (a leaf). وسنرمز إلى هذا العدد بالرمز ***p***. نفرض أن شجرة الحسم فيها ***N*** عقدة. وحيث أن كل عقدة لها على الأكثر ابنان (two children)، فلذلك يكون عدد العقد عند مسافة معينة (a particular distance) [حيث نعتبر أي حرف يمثل مسافة واحدة (counting each edge as one)] من الجذر هو على الأكثر ضِعف العدد عند المسافة السابقة. ونظرا لأن أكبر مسافة بين أي عقدة والجذر هي $p - 1$، فلذلك

$$N \leq 1 + 2 + 4 + \ldots + 2^{p-1}$$

وبتطبيق المعادلة (8) من الفصل السابق نرى أن الطرف الأيمن في هذه المتباينة ***يساوي*** 2^p-1، وبالتالي نحصل على $2^p \geq (N + 1)$.

لدينا الآن علاقة بين p و N، ولكننا نريد علاقة بين p و n (حيث n هي عدد عناصر المنظومة التي نبحث فيها). الادعاء المفتاح (key claim) في ذلك هو أن $N \geq n$ إذا كانت الخوارزمية ***A*** تعمل بطريقة صحيحة في جميع الحالات. وبصورة ***خاصة*** فإننا نزعم أنه تُوجد عقدةٌ ما (some node) في شجرة الحسم معنونةٌ i (labeled) لأي i من 0 إلى $n - 1$ [أي أن شجرة الحسم تحتوي على الأقل على n عقدة].

ولإثبات ذلك بالتناقض نفرض أنه توجد قيمةٌ ما i في المدى من 0 إلى $n - 1$ ولكن لا توجد أي عقدة مُعَنْونة i. يمكننا أن نكوِّن منظومتين مُدْخَلَتين (make *E1*, *E2* up two input arrays) بحيث أن $E1[i] = K$ ولكن $E2[i] = K' > K$. ***لجميع*** المؤشرات j (all indexes) الأقل من i نجعل $E1[j] = E2[j]$ باستخدام بعض القيم المفاتيح (key values) الأقل من K

بالترتيب المفروز (in sorted order). وبالنسبة لجميع المؤشرات j الأكبر من i نجعل $E1[j] = E2[j]$ باستخدام بعض القيم المفاتيح الأكبر من K بالترتيب المفروز. ونظرا لأنه لا توجد أي عقدة في شجرة الحسم مُعَنْوَنَة i، فإن الخوارزمية A سوف لا تقارن K إطلاقا بأيٍّ من $E1[i]$ أو $E2[i]$. وتسلك الخوارزمية السلوك نفسه مع المنظومتين المُدْخَلتين نظرا لأن عناصرهما الأخرى متطابقة (identical)، ويجب أن تُعْطِي الخوارزمية المخرجات نفسها (same output) للمُدخلين (both inputs). وهكذا فإن الخوارزمية A ستعطِي المخرجات الخاطئة لإحدى المنظومتين وبالتالي فهي ليست خوارزمية صحيحة (correct algorithm). وهكذا نستنتج أن شجرة الحسم تحتوي على n عقدة على الأقل.

بناء على ما سبق نحصل على $2^p \geq (N + 1) \geq (n + 1)$، حيث p هي عدد المقارنات على أطول مسار في شجرة الحسم. وبأخذ اللوغاريتمات نحصل على $p \geq \lg(n+1)$ ونظرا لأن الخوارزمية A كانت خوارزمية اختيارية من طائفة الخوارزميات قيد الدراسة، فإننا نكون قد أثبتنا النظرية التالية :

نظرية 2-6:

أي خوارزمية لإيجاد K في منظومة مكوَّنة من n عنصر (عن طريق مقارنة K مع عناصر المنظومة) يجب أن تقوم بتنفيذ $\lceil \lg (n+1) \rceil$ مقارنة على الأقل لأي مدخلات.

نتيجة:

نظرا لأن خوارزمية البحث الثنائي (الخوارزمية 2-4) تقوم بتنفيذ $\lceil \lg (n+1) \rceil$ مقارنة في أسوأ حالة، فهي خوارزمية مُثْلَى.

تمرينات الفصل الثاني

تمرينات الفصل الثاني

2-1) نفرض أن لدينا الخوارزمية التالية:

خوارزمية البحث التتابعى – غير المرتب

Sequential Search Algorithm, unordered

المدخلات: E, n, K، حيث

E: منظومة من عناصر عددها n، ومؤشراتها $0, 1, \ldots, n\text{-}1$.

K: القيمة / المفتاح (key) الذى نبحث عنه (item sought). وللبساطة (simplicity) نفرض أن K وعناصر المنظومة E جميعها أعداد صحيحة مثل n.

المخرجات: الخوارزمية تعيد (returns) القيمة ans وهى موضع K (location of) فى المنظومة E، بينما تعيد 1- إن لم تجد K.

```
int seqSearch ( int [ ] E, int n, int K )
1. int ans, index;
2. ans = −1;                    // Assume failure
3. for ( index = 0; index < n; index + +)
4.     if ( K == E[index] )
5.         ans = index;         //success!
6.         break;               // to leave the loop
7. return ans;
```

أوجد صيغة (formula) للعدد الإجمالى للعمليات (total number of operations) التى تجريها هذه الخوارزمية للبحث التتابعى فى أسوأ

حالة (in the worst case) بالنسبة لمنظومة عدد عناصرها n، وذلك عن طريق حساب:

- عدد مقارنات K مع عناصر المنظومة E.
- عدد المقارنات مع المتغير index.
- عدد عمليات الجمع (additions).
- عدد عمليات الإسناد (assignments) للمتغير index.

2-2) يُعرَّف أوسط / وسيط مجموعة (median of a set) بأنه عنصر فى المجموعة يحقق الشرط أن: عدد العناصر الأصغر منه يساوى (أو أقل بواحد من) عدد العناصر الأكبر منه [number of elements less than the median is within one of the number of elements that are greater] بفرض أنه لا يوجد عنصران متساويان (assuming no ties). [مثلا: أوسط المجموعة $\{8, 5, 20\}$ هو 8 وأوسط المجموعة $\{30, 5, 7, 20\}$ هو 7].

أ) اكتب خوارزمية لإيجاد أوسط ثلاثة أعداد صحيحة مختلفة a, b, c .

ب) صف D (describe) مجموعة مدخلات خوارزميتك (set of inputs for your algorithm).

جـ) ارسم شجرة قرار الخوارزمية.

د) كم عدد المقارنات التى تجريها خوارزميتك فى أسوأ حالة؟

هـ) كم عدد المقارنات الضرورية / اللازمة (necessary) فى أسوأ حالة

لإيجاد أوسط ثلاثة أعداد؟ عَلِّل إجابتك (justify your answer).

و) كم عدد المقارنات التى تجريها خوارزميتك فى المتوسط؟

3-2) اكتب خوارزمية لإيجاد كل من أصغر عنصر وأكبر عنصر فى مجموعة مكونة من n عنصر. حاول أن تجد طريقة تجرى على الأكثر عدد $1.5n$ تقريباً من المقارنات بين العناصر.

4-2) نفرض أن

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

حدودية (polynomial) فى n من الدرجة k (of degree)، حيث $a_k > 0$. اثبت أن $p(n)$ تقع فى $\Theta(n^k)$.

5-2) رتب قائمة الدوال التالية من أقل رتبة مقاربة (lowest asymptotic order) إلى أعلى (highest) رتبة مقاربة، وإذا تساوت دالتان (أو أكثر) فى الرتبة المقاربة فبَيّن ذلك.

أ) ابدأ بالدوال الأساسية التالية

n	2^n	$n \lg n$	n^3
n^2	$\lg n$	n-$n^3 + 7n^5$	$n^2 + \lg n$

ب) اجمع (combine) الدوال التالية ضمن إجابتك على الجزء أ) من السؤال، وافرض أن $0 < \epsilon < 1$.

e^n	$\sqrt{n}$	2^{n-1}	$\lg \lg n$
$\ln n$	$(\lg n)^2$	$n!$	$n^{1+\epsilon}$

2-6) اثبت النظرية التالية:

نظرية: n^k تقـــع فـى $o(2^n)$ ، وذلـك لأى $k > 0$. أى أن قـوى n (powers of) تنمو ببطء أكبر من نمو الدالة الأسية 2^n (the exponential function). [فى الحقيقة قوى n تنمو ببطء أكبر من نمو أى دالة أسية c^n، حيث $c > 1$].

2-7) أعـط مثـالاً لدالتـين $f, g : N \to R^*$ بحيـث أن $f \notin O(g)$ & $g \notin O(f)$.

2-8) صمم تعديلاً لخوارزمية البحث الثنائى (Binary Search) (الخوارزمية 2- 4) بحيث تقوم الخوارزمية المعدلة بإجراء مقارنة ثنائية (binary comparison) [أى أن المقارنة تعيد نتيجة منطقية / بُولية (a Boolean result)] واحدة فقط للمفتاح K (الذى نبحث عنه) مع عنصر فى المنظومة (an array entry) فى كل استدعاء للدالة (per function invocation). يمكنك إضافة مقارنات إضافية على متغيرات المدى (range variables). حلِّل (analyze) صحة (correctness) إجرائك. **إرشاد:** متى يجب ان تكون مقارنتك الواحدة هى التساوى (= =) (equality)؟

2-9) نفرض أننا نقوم بالمقارنة بين تنفيذ خوارزمية الترتيب بالإدخال وتنفيذ خوارزمية الترتيب بالدمج على الآلة نفسها (on the same machine). بالنسبة لمدخلات سعتها n: يتم تنفيذ الترتيب بالإدخال فى $8\,n^2$ خطوة (steps)، بينما يتم تنفيذ الترتيب بالدمج فى $64\,n \lg n$ خطوة. ما هى قيم n التى يتفوق عندها الترتيب بالإدخال على الترتيب بالدمج؟

2-10) ما هى أصغر قيمة للمتغير n بحيث يتم تنفيذ خوارزمية زمن تشغيلها $100\,n^2$ أسرع من تنفيذ خوارزمية زمن تشغيلها 2^n على الآلة نفسها؟

2-11) مقارنة بين أزمنة التشغيل Comparison of running times

لكل دالة $f(n)$ وزمن t فى الجدول التالى، حدد أكبر سعة n (largest size) لمسألة يمكن حلها فى زمن t، بفرض أن الخوارزمية المستخدمة لحل المسألة تستغرق $f(n)$ ميكرو ثانية.

t / f(n)	ثانية	دقيقة	ساعة	شهر	سنة
$\lg n$					
$\sqrt{n}$					
n					
$n \lg n$					
n^2					
n^3					
2^n					
$n!$					

2-12) عَبّر عن الدالة $n^3/1000 - 100\,n^2 - 100\,n + 3$ بدلالة الاصطلاح $(notation) - \Theta$.

2-13) نفرض أننا سنقوم بترتيب (sorting) أعداد مخزونة فى منظومة A وعددها n بالطريقة التالية: نبحث أولاً عن أصغر عنصر (smallest element) فى المنظومة A، ونبدله (exchange it) مع العنصر الموجود فى $A[1]$. ثم نبحث عن ثانى أصغر عنصر

(second smallest element) فى A، ونبدله مع $A[2]$. ونستمر بهذه الكيفية للعناصر الـ n-1 الأولى فى A (first n-1 elements).

- اكتـــب شـــبه كـــود (pseudo code) لهـذه الخوارزميـة التـى تعـرف باسـم **"خوارزميـة الترتيـب بالاختـيار" (selection sort algorithm)**.

- اذكر زمن التشغيل فى أحسن حالة، وزمن التشغيل فى أسوأ حالة، وكذلك زمن التشغيل في المتوسط بالنسبة لخوارزمية الترتيب بالاختيار بدلالة الاصطلاح $(notation) - \Theta$.

14-2) بالنسبة لطريقة البحث الخطى (linear / sequential search):

- كم عدد عناصر متتابعة الإدخال (input sequence) التى نحتاج لاختبارها (need to be checked) فى المتوسط (on the average) بفرض أن العنصر الذى نبحث عنه (key) تتساوى احتمالاته (equally likely) أن يكون أى عنصر فى المنظومة؟

- وماذا عن إجابة السؤال فى أسوأ حالة؟

- ما هو زمن التشغيل (running time) فى كل من المتوسط وأسوأ حالة (average-case and worst-case) لخوارزمية البحث الخطى بدلالة الاصطلاح Θ؟ علل إجابتك.

15-2) كيف يمكننا تعديل (modifying) أى خوارزمية تقريبًا (almost any algorithm) حتى يصبح زمن تشغيلها جيداً فى أحسن حالة (to have a good best-case running time)؟

16-2) بالإشارة مرة أخرى إلى مسألة البحث الخطى عن عنصر K لاحظ أنه إذا تم ترتيب / فرز المتتابعة A (sequence A is sorted)، فإنه يمكننا حينئذ اختبار / مقارنة النقطة الوسطى فى المتتابعة مع القيمة K واستبعاد / حذف (eliminating) نصف المتتابعة من أى اعتبار تالى فى عملية البحث. **"البحث الثنائى" (Binary search)** هو خوارزمية تكرر هذا الإجراء، بتنصيف (halving) سعة الجزء المتبقى من المتتابعة فى كل مرة.

اكتب شبه كود (pseudo code) تكرارياً (iterative) أو ارتداديا (recursive) لخوارزمية البحث الثنائى. وبَيّن أن زمن التشغيل فى أسوأ حالة لهذه الخوارزمية هو $\Theta(\lg n)$.

17-2- أ) هل $2^{n+1} = O(2^n)$؟

ب) وهل $2^{2n} = O(2^n)$؟

18-2) **خواص اصطلاح المُقارَبة Asymptotic notation properties**

نفرض أن $f(n),\ g(n)$ دالتان موجبتان تقاربيا (asymptotically positive functions). اثبت صحة أو خطأ كل من العبارات التالية:

أ) $f(n) = O(g(n)) \quad \Rightarrow \quad g(n) = O(f(n))$

ب) $f(n) + g(n) = \Theta\ (\ \min\ (f(n), g(n)\ ))$

جـ) $f(n) = O\big(g(n)\big) \Rightarrow lg(f(n)) = O\left(lg\big(g(n)\big)\right)$

حيث [لجميع قيم n الكبيرة كبراً كافياً

$[lg(g(n)) \geq 1\ \&\ f(n) \geq 1$

د) $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O\left(2^{g(n)}\right)$

هـ) $f(n) = O((f(n))^2)$

و) $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$

ز) $f(\mathrm{n}) = \Theta(f(\mathrm{n}/2))$

19-2) أوجد صيغة (formula) للعدد الإجمالى للعمليات التى تجريها الخوارزمية التالية findMax فى أسوأ حالة بالنسبة لمنظومة عدد عناصرها n. احسب عدد مقارنات max مع عناصر المنظومة، وعدد المقارنات مع المتغير index، وعدد عمليات الجمع، وعدد عمليات الإسناد (assignments) للمتغير index.

خوارزمية findMax

المدخلات: E, n، حيث

E: منظومة من أعداد (array of numbers) معرفة مؤشراتها $0, 1, \ldots, n\text{-}1$.

$n \geq 1$: عدد عناصر المنظومة.

المخرجات: الخوارزمية تعيد القيمة max، وهى أكبر عنصر فى المنظومة E.

```
int findMax ( E, n )
1.    max = E[0];
2.    for (index = 1; index < n; index + +)
3.          if (max < E[index])
4.                max = E[index]);
5.    return max;
```

2-20) طَبِّق طريقة البحث الثنائى للبحث عن العدد 18 فى القائمة التالية، موضحاً خطوات الطريقة خطوة خطوة.

10	12	13	14	18	20	25	27	30	35	40	45	47

2-21) نفرض أن لدينا منظومة بها عدد كبير جدا من العناصر وليكن ∞. العناصر الـ n الأولى من المنظومة مرتبة ترتيبا تصاعديا (sorted in ascending order)، وباقى المنظومة يحتوى على عناصر كبيرة جدا ولتكن ∞. قيمة n نفسها غير معلومة. المطلوب كتابة خوارزمية للبحث عن عنصر K فى هذه المنظومة.

إرشاد: يمكنك حل المسألة فى زمن $\Theta(\log n)$.

2-22) يوجد عندنا خوارزميتان لحل مسألة ما ولتكن P وحجمها n (size). أى خوارزمية ينبغى استخدامها؟ ولماذا؟

أ) الخوارزمية الأولى تقوم بحل المسألة فى n^2 من الخطوات.

ب) ب) الخوارزمية الثانية من نوع قَسِّم وتَغَلَّب (divide and conquer) حيث تقوم بتقسيم المسألة إلى مسألتين متساويتين حجم كل منهما $n/2$ فى n من الخطوات. ثم تقوم بحل كل من المسألتين الفرعيتين ارتداديا بالطريقة نفسها. وأخيرا تقوم بدمج / تجميع الحلين للحصول على حل المسألة الأصلية فى n من الخطوات.

الفصل الثالث:

الترتيب / الفَرْز

Sorting

الترتيب بالإدخال

حدود سفلى لسلوك خوارزميات ترتيب معيَّنة

خوارزميات قَسِّم وتَغَلَّب / فَرِّق تَسُدْ .

الفرز/ الترتيب السريع

دمج المتتابعات المُرَتَّبَة

الترتيب بالدمج .

الحد الأدنى لعدد المقارنات بين المفاتيح لأجل الترتيب

الترتيب بالكَوْمَة

بناء الكومة .

مقارنة بين خوارزميات الترتيب

- **تمرينات الفصل الثالث**

الفصل الثالث :

الترتيب / الفَـرْز

Sorting

سنتناول بإذن الله تعالى في هذا الفصل دراسة عدة خوارزميات لترتيب عناصر مجموعة ترتيبًا تصاعديًّا (غير تناقصي) أو تنازليا (غير تزايدي). ومسألة ترتيب مجموعة أشياء هي من أولى مسائل علم الحاسوب التي تمت دراستها بصورة مُكَثَّفة، ولها تطبيقات عملية كثيرة مفيدة. فمثلاً ترتيب الأسماء في دليل الهاتف (التليفون)، وترتيب كلمات القواميس أبجديًا يجعل من السهل استخدامها. وترتيب كميات البيانات الضخمة المخزنة في الحاسوب يجعل من السهل الوصول إليها. ومناقشة عدة خوارزميات في هذا الفصل بإذن الله ستلقى الضوء على عدة نقاط ككيفية تحسين أداء خوارزمية معطاة، وكيفية الاختيار بين عدة خوارزميات لحل مسألة معينة، ومحاولة الوصول إلى حدود سفلى قوية (strong lower bounds) لأسوأ حالة وللسلوك المتوسط [قوية: بمفهوم أنه توجد خوارزميات تقوم تقريبا بتنفيذ أقل كمية شغل محددة، أي أنه تكون لدينا خوارزمية ترتيب مُثْلَى].

وفي وصف معظم الخوارزميات التي سندرسها سنفرض أن المجموعة (set) المراد ترتيب عناصرها مخزونة كمنظومة (an array)، وبالتالي فالعنصر الموجود عند أي موضع يمكن الوصول إليه في أي وقت، وهذا يُطْلَق عليه "**وصول عشوائي**" **(random access)**. إلا أن هناك بعض الخوارزميات التي تفيد أيضا في ترتيب الملفات (files) والقوائم المترابطة (linked lists).

وعندما يكون الوصول (access) للمجموعة بطريقة تتابعية (sequential) فقط، فإننا نستخدم الاصطلاح "**متتابعة**" (sequence) للتأكيد على أن البنية (structure) قد تكون قائمة مترابطة أو ملفا تتابعيا بالإضافة إلى منظومة. وإذا كانت أي منظومة مُعَرَّفة على مدى المؤشرات (range of indexes) 0, …, n-1، فإن أي **مدى (range)** أو **مدى جزئي (subrange)** لهذه المنظومة هو عبارة عن متتابعة متلاصقة (متصلة تلاصقيا) (a contiguous sequence) [أي أن جميع عناصرها متصلة بجوار بعضها البعض] من العناصر بين مؤشرَين محدَّدَيْن **first, last** بحيث أن $0 \le \text{first}, \text{last} \le n-1$. وإذا كانت last < first فيقال للمدى إنه "**فارغ / خاوي**" **(empty)**.

ونفرض أن أي عنصر في المجموعة المراد ترتيبها يحتوي على اسم تعريفي (an identifier) يطلَق عليه "**مفتاح**" (**a key**) وهو عبارة عن عنصر في مجموعةٍ ما مرتبة خطيا (some linearly ordered set)، وأنه يمكن مقارنة مفتاحين لتحديد أيهما أكبر أو أنهما متساويان. ودائما نرتِّب المفاتيح بترتيب غير تناقصي (nondecreasing order). وكل عنصر في المجموعة قد يحتوي على معلومات أخرى بجانب المفتاح. وعندما يعاد ترتيب (rearranging) المفاتيح أثناء عملية الترتيب / الفرز (sorting process)، فإن المعلومات المصاحبة (associated information) يعاد أيضا ترتيبها بما هو مناسب (as appropriate)، ولكننا أحيانا نشير فقط إلى المفاتيح ولا نذكر صراحةً بقية العنصر (the rest of the entry).

وجميع الخوارزميات التي سندرسها بإذن الله في هذا الفصل هي من طائفة خوارزميات الترتيب (sorting algorithms) التي قد تقارن بين

المفاتيح (وتنسخها) ولكن يجب ألا تعمل أي عمليات أخرى على المفاتيح. ونطلِق على هذه الخوارزميات "الخوارزميات التي تُرتِّب بالمقارنة بين المفاتيح" (algorithms that sort by comparison of keys) أو "الخوارزميات المَبْنِيَّة / المؤسسة على المقارنات" (comparison–based algorithms) للترتيب. ومقياس الجهد / الشغل (measure of work) المستخدم أساسا لتحليل الخوارزميات في هذه الطائفة هو عدد المقارنات بين المفاتيح (number of comparisons of keys). وسنحصل بإذن الله أيضا على حدود سفلى لعدد المقارنات التي تجريها هذه الخوارزميات.

وخوارزميات الترتيب التي سنتناولها في هذا الفصل يطلق عليها "ترتيبات داخلية" (internal sorts) وذلك لأن البيانات (data) يُفْتَرَض أنها موجودة في ذاكرة وصول عشوائي عالية السرعة في الحاسوب (computer's high-speed, random-access memory). وأما الخوارزميات الخاصة بترتيب مجموعات كبيرة من البيانات المخزونة على وسائل تخزين خارجية أبطأ (external slower storage devices) مع قيود على طريقة الوصول إلى البيانات فيطلق عليها "ترتيبات خارجية" (external sorts).

وعند تحليل خوارزميات الترتيب فإننا نأخذ في الاعتبار قَدْر الحيِّز الإضافي (how much extra space) الذي تستخدمه (بالإضافة إلى المدخلات)، وإذا كانت كمية (amount) هذا الحيِّز الإضافي ثابتة (constant) بالنسبة لحجم المدخلات (input size)، فَيُقَال إن الخوارزمية "تعمل في مكانها الصحيح" (work in place).

ولجعل الخوارزميات أكثر وضوحا قدر الإمكان نستخدم كلاًّ من

Element و **Key** كَإِسم تعريفي لنوع (type identifier)، ولكننا نعامل **Key** على أنه نوع عددي (numeric type) حيث نستخدم المؤثرات العلاقية (relational "=, ≠, <, …" operators).

الترتيب بالإدخال Insertion Sort

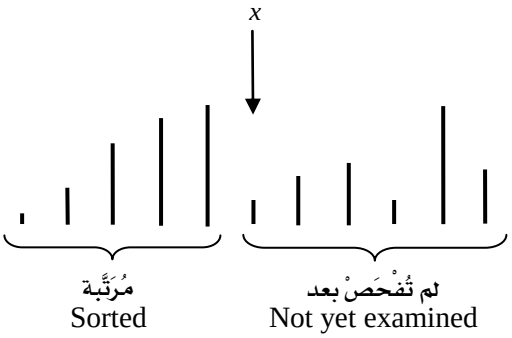
فكرة خوارزمية الترتيب بالإدخال بسيطة وطبيعية وعامة (simple, natural, general)، وتحليلها في أسوأ حالة وكذلك تحليل سلوكها المتوسط سهل.

الاستراتيجية The Strategy

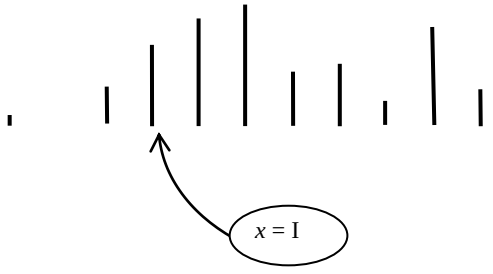
نبدأ بمتتابعة E مكونة من n عنصر بترتيب اختياري (in arbitrary order)، كما يوضح ذلك شكل 3-1. [الترتيب بالإدخال يمكن أن يستخدم على مفاتيح من أي مجموعة مرتبة خطيا (any linearly ordered set)، ولكن بالنسبة للشكل التوضيحي للعِصِيِّ (stick figure illustrations) اعتبر أن المفاتيح هي ارتفاعات / أطوال العِصِيِّ (heights of the sticks) التي تُعَدُّ هي العناصر (elements)].

نفرض أننا قد قمنا بترتيب (sorting) جزء ابتدائي (شريحة مرتبة) من المتتابعة (some initial segment of the sequence). شكل 3-2 يبيِّن صورة للمتتابعة بعد ترتيب الخمسة عناصر اليسرى. الخطوة العامة (general step) هي أن نزيد طول الشريحة المرتَّبة (sorted segment) بإدخال (inserting) العنصر التالي (next element) في مكانه الصحيح / المناسب (proper place).

شكل 3—1
عناصر غير مرتبة Unsorted elements



شكل 3—2
عناصر مرتبة جزئيا Partially sorted elements



شكل 3—3
إدخال x في ترتيبها الصحيح Insertion of x in its proper order

نفرض أن x هو العنصر التالي المطلوب إدخاله في الشريحة المرتبة، أي أن x هو العنصر الموجود أقصى اليسار (leftmost element) في الشريحة التي لم يتم ترتيبها / معالجتها / فحصها (unexamined segment). نقوم أولاً

بنَزْع / برَفْع x (pull) "خارج الطريق" (out of the way) أي ننسخ x في متغير محلي (a local variable) تاركين **فراغا (vacancy)** في موضعه السابق. ثم تكراريا (repeatedly) نقارن x مع العنصر الموجود مباشرة يسار الفراغ، وطالما أن x أصغر نُحَرِّك هذا العنصر إلى الفراغ (into the vacancy) تاركا بذلك فراغا في موضعه الذي كان فيه. أي أن الفراغ يتزحزح (shifts) موضعا واحدا إلى اليسار. هذه العملية تتوقف عندما لا يتبقى لدينا أي عنصر يسار الفراغ الحالي، أو حين يكون العنصر الموجود يسار الفراغ الحالي أصغر من أو مساويا x. عندها نُدْخِل x في الفراغ، كما يبيِّن ذلك شكل 3-3. ولكي نبدأ الخوارزمية، نحتاج فقط لملاحظة أن أول عنصر بمفرده يمكن أن يُعتبر شريحة مرَتَّبة (a sorted segment). ولكتابة ذلك في صورة إجراء (procedure) سنفرض أن المتتابعة عبارة عن منظومة، إلاَّ أن الفكرة تُطَبَّق أيضا على القوائم والهياكل / البنى التتابعية (sequential structures) الأخرى.

الخوارزمية والتحليل The Algorithm and Analysis

نفرض أن البرنامج الفرعي shiftVac (E, vacant, x) وظيفته زحزحة العناصر حتى يصبح الفراغ (vacancy) موجودًا عند الموضع الصحيح (correct position) الذي نضع فيه x بين العناصر المرتبة (sorted elements). ويعيد الإجراء مؤشر الفراغ (index of the vacancy)، وليكن **xLoc**، إلى الدالة المستدعية. وفيما يلي مواصفات **shiftVac** و بيان شروطه السابقة وشروطه اللاحقة. وبكلمات أخرى فإن **shiftVac** يقوم بتنفيذ الانتقال من شكل 3-2 إلى شكل 3-3.

int shiftVac (Element [ ] E, int vacant, Key x)

شروط سابقة (Preconditions):

1) **vacant** غير سالبة.
2) عناصر المنظومة **E** عند المؤشرات الأقل من **vacant** تكون مرتبة تصاعديا.

شـروط لاحقـة (Postconditions): نفـرض أن **xLoc** هـي القيمـة المعـادة (returned value) إلى الدالة المستدعية.

1) عناصر المنظومة **E** عند المؤشرات (indexes) الأقل من **xLoc** ستكون في مواضعها الأصلية ولديها مفاتيح (keys) أقل من أو تساوي x .

2) عناصر المنظومة **E** عند المواضع **xLoc + 1, …, vacant** ستكون أكبر من x، وقد زُحْزِحت لأعلى (shifted up) بموضع واحد عن مواضعها عند استدعاء **shiftVac**.

مواصفات البرنامج الفرعي Specifications for shiftVac

والآن يمكـن للإجـراء insertionSort أن يقـوم بمجـرد الاسـتمرار في استدعاء shiftVac مكوِّنا شريحةً مُرَتَّبة (sorted segment) أطول فأطول عند النهاية اليسرى إلى أن تصبح جميع العناصر في الشريحة المرتبة.

وإجراء shiftVac يأخذ شكلاً نمطيًّا (a typical form) لإجراء البحث الروتيني المُعَمَّم (تعريف 2-4). حيث أنه **إذا** لم توجد أي بيانات أخرى إضافية ننظر فيها (no more data to look at) **فإن** الإجراء يفشل (fail)، **وإلاَّ** فإننا ننظر في وحدة بيانات (one data item) أخرى، **فإنْ** كانت هي ما نبحث عنه، **فإنَّ** الإجراء ينجح (succeed)، **وما عدا ذلك** نستمر بالنظر في البيانات المتبقية [التي لم نفحصها بعد (unexamined data)]. ويمكننا أن نكتب الإجراء **shiftVac** بصيغة تكرارية (iterative) أو بصيغة ارتدادية (recursive). وفيما يلي الصيغة الارتدادية وهي مباشرة (straightforward).

```
int shiftVacRec ( Element[ ] E, int vacant, Key x )
                        int xLoc;
1.   if (vacant) = = 0)
2.       xLoc = vacant;
3.   else if ( E[vacant–1].key ≤ x )
4.       xLoc = vacant;
5.   else
6.       E[vacant] = E[vacant–1];
7.       xLoc = shiftVacRec(E, vacant–1, x);
8.   return xLoc;
```

وللتحقق من أننا نستخدم الارتداد بصورة صحيحة في السطر 7، نلاحظ أن الاستدعاء الارتدادي (recursive call) يعمل على مدى أصغر (a smaller range) ، وأن وسيطه الثاني (second argument) غير سالب، وبالتالي يكون الشرط السابق (precondition) - المذكور في مواصفات shiftVac - متحققاً (satisfied). [لماذا لا يمكن أن يكون vacant–1 سالبا؟]. و نلاحظ أن صحة (correctness) الخوارزمية تكون الآن مباشرة إذا تذكرنا أنه يمكننا أن نفرض أن الاستدعاء الارتدادي في السطر 7 يؤدي مهمته.

ورغم أن الإجراء لـ **shiftVacRec** بسيط جدا، إلا أن رؤية تَتَبُّع نشاط (visualizing the activation trace) العنصر الذي ترتيبه n في المنظومة E المطلوب إدخاله (to be inserted) تجعلنا ندرك أن عمق الارتداد (depth of recursion) أو أن رَصَّةَ الإطار (frame stack) يمكن أن تنمو إلى الحجم n (grow to size). وهذا يمكن أن يكون غير مرغوب فيه لقيم n الكبيرة. ولذلك فهذه حالة حيث يجب تغيير الارتداد إلى تكرار بعد التأكد من أن كل شئ يعمل بصورة صحيحة. والهدف ليس هو توفير وقت (save time) بقدر ما هو تقليل الحيِّز المستخدم (conserve space). والخوارزمية الكاملة التالية تشتمل على صيغة **shiftVac** التكرارية (iterative version).

خوارزمية 3-1 الترتيب بالإدخال Insertion Sort

المدخلات (Input): E: منظومة عناصر.

$n \geq 0$: عدد العناصر.

مدى المؤشرات: 0, …, n - 1 (range of indexes).

المخرجات (Output): المنظومة E حيث عناصرها مرتبة بترتيب غير تناقصي لمفاتيحها (nondecreasing order of their keys).

```
void insertionSort ( Element [ ] E, int n )
    int xindex;
    for (xindex = 1; xindex < n; xindex + +)
        Element current = E[xindex];
        key x = current.key;
        int xLoc = shiftVac ( E, xindex, x );
        E[xLoc] = current;

int shiftVac ( Element[ ] E, int vacant, Key x )
    int xLoc = 0;       // Assume failure.
    while (vacant > 0)
        if ( E[vacant–1].key ≤ x )
            xLoc = vacant;      // Succeed.
            break;
        E[vacant] = E[vacant–1];
        vacant – –;       // Keep looking.
    return xLoc
```

درجة تعقيد أسوأ حالة Worst – Case Complexity

لهذا التحليل سنستخدم ***i*** لترمز إلى **xindex**. ولكل قيمة من قيم i سيكون أكبر عدد ممكن من مقارنات المفاتيح (maximum number of key comparisons possible) في استدعاء واحد لـ **shiftVac** التكراري، أو استدعاء واحد على مستوى القمة (one top–level call) لـ **shiftVacRec** الارتدادي] هو ***i***. وبالتالي يكون العدد الإجمالي

$$W(n)=\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

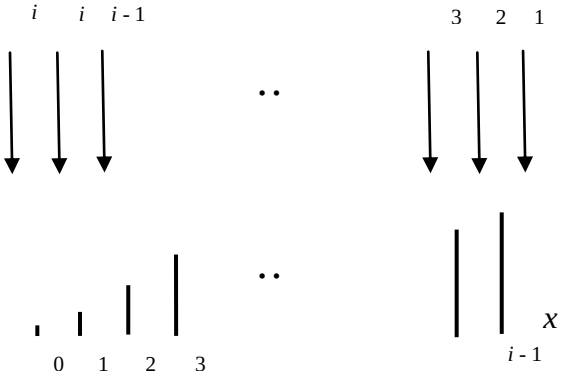
ولاحظ أننا قد وصلنا إلى حد أعلى (an upper bound) لسلوك أسوأ حالة. وبتفكير بسيط يمكننا أن نتحقق من أنه توجد فعلا مُدْخَلات تؤدي إلى إجراء مقارنات عددها $n(n\text{-}1)/2$. إحدى أسوأ الحالات هذه تتحقق عندما تكون المفاتيح في ترتيب معكوس (in reverse order) [أي تناقصي (decreasing)]. وبناءً عليه فإن

$$W(n)=\frac{n(n-1)}{2} \in \Theta\left(n^2\right)$$

السلوك المتوسط Average Behavior

نفرض أن جميع تباديل المفاتيح (permutations of the keys) متساوية الاحتمالات (equally likely) كمدخلات. سنحسب أولا متوسط عدد مقارنات المفاتيح اللازمة لإدخال عنصر واحد جديد في الشريحة المرتبة - أي لاستدعاء واحد للإجراء **shiftVac** - وذلك لأي قيمة معينة من قيم i (التي استخدمناها لترمز إلى **xindex**). ولتبسيط التحليل سنفرض أن المفاتيح متباينة (distinct). [والتحليل شبيه جدا بالتحليل الذي قمنا به لخوارزميات البحث التتابعي في الفصل السابق].

عدد المواضع التي يمكن للعنصر x أن يذهب إلى أيٍّ منها هو $i + 1$.
والشكل التالي (شكل 3-4) يبيِّن عدد المقارنات التي يتم إجراؤها بناءً على الموضع.



شكل 3-4

عدد المقارنات المطلوبة لتحديد الموضع الذي سيوضع فيه العنصر x
Number of comparisons needed to determine the position for x

احتمال أن تنتمي x إلى أحد المواضع المحددة هو $1/(i + 1)$. [هذا يعتمد على حقيقة أن x لم يتم فحصها من قبل بواسطة الخوارزمية. وإذا كانت الخوارزمية قد اتخذت قرارات سابقة بناءً على قيمة x، فقد لا يمكننا بالضرورة فرض أن x عشوائية بانتظام (uniformly random) بالنسبة لأول i مفتاح (w. r. t. the first i keys)]. وبالتالي يكون متوسط عدد المقارنات في **shiftVac** لنجد موضع العنصر الذي ترتيبه i هو (the location for the i– th element):

$$\frac{1}{i+1}\sum_{j=1}^{i} j + \frac{1}{i+1}(i) = \frac{i}{2} + \frac{i}{i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

وبالجمع لجميع الإدخالات التي عددها $n - 1$ إدخال (adding for all insertions) نحصل على:

$$A(n)=\sum_{i=1}^{n-1}\left(\frac{i}{2}+1-\frac{1}{i+1}\right)=\frac{n(n-1)}{4}+n-1-\sum_{j=2}^{n}\frac{1}{j}$$

حيث قمنا بالتعويض $j = i + 1$ لنحصل على المجموع الأخير. وكنا قد رأينا من المعادلة (16) بالفصل الأول (انظر أيضا مثال 1–1) أن $\sum_{j=1}^{n}(1/j)\approx \ln n$. ويمكننا دمج الواحد 1 الذي يسبق المجموع Σ مع هذا المجموع لنجعل النهاية السفلى (lower limit) للمجموع هي $j = 1$. وبإهمال الحدود ذوات الرتبة الأقل (lower–order terms) نحصل على:

$$A(n)\approx\frac{n^2}{4}\in\Theta(n^2)$$

الحيِّز / المكان Space

من الواضح أن خوارزمية الترتيب بالإدخال هي خوارزمية ترتيب في مكانها السليم (an in-place sort) بصيغة **shiftVac** التكرارية. وبالصيغة الارتدادية فإن رَصَّةَ الإطار (frame stack) يمكن أن تنمو إلى $\Theta(n)$.

حدود سفلى لسلوك خوارزميات ترتيب معيَّنَة

Lower Bounds on the Behavior of Certain Sorting Algorithms

ننظر إلى العنصر الذي مفتاحه x وهو يشغل (occupies) الموضع "الفارغ" (vacant) في المنظومة عندما تقارن خوارزمية "الترتيب بالإدخال" x بالمفتاح الذي على يساره. ثم بعد كل مقارنة تقوم خوارزمية الترتيب بالإدخال إما بعدم تحريك أي عنصر أو ببساطة بتبديل (interchanging) عنصرين متجاورين. وسنبيِّن بإذن الله أن جميع خوارزميات الترتيب التي تقوم بمثل هذا التحريك المحلي المحدود للعناصر (limited local moving of elements) بعد كل مقارنة يجب أن تبذل تقريبا كمية الجهد نفسها التي تبذلها خوارزمية الترتيب بالإدخال.

ويمكننا أن نصف أي تبديل (permutation) لعناصر عددها n بدَالَّة واحد لواحد (one-to-one function) من المجموعة $N = \{1, 2, \ldots, n\}$ إلى نفسها (onto itself). وهناك تباديل مختلفة عددها $n!$ على n عنصر. ونفرض أن العناصر في المتتابعة غير المرتبة $\boldsymbol{E}$ (unsorted sequence) هي $x_1, x_2, \ldots, x_n$. ولتبسيط الاصطلاحات في هذه المناقشة نفرض أن العناصر المطلوب ترتيبها مخزونة في المواضع $1, 2, \ldots, n$ في المتتابعة $\boldsymbol{E}$ بدلا من $0, 1, \ldots, n\text{-}1$. هناك تبديل $\boldsymbol{\pi}$ (a permutation) بحيث أنه للقيم $1 \le i \le n$ فإن $\boldsymbol{\pi(i)}$ هو الموضع الصحيح لـ x_i (correct position of) عندما يتم ترتيب (sorting) المتتابعة. وبدون أي فقدان للعمومية (without loss of generality) يمكننا أن نفرض أن المفاتيح هي الأعداد الصحيحة $1, 2, \ldots, n$ نظرا لأنه يمكننا أن نعوِّض 1 عن أصغر مفتاح و 2 عن المفتاح التالي في الصغر (next smallest)، ... وهكذا، بدون إحداث أي تغييرات في التعليمات التي تنفذها الخوارزمية. وبالتالي ستكون المدخلات غير المرتبة $\pi(1), \pi(2), \ldots, \pi(n)$. مثلا دعنا نأخذ في الاعتبار متتابعة الإدخال (input sequence) 2, 4, 1, 5, 3. $\pi(1) = 2$ تعني أن المفتاح الأول 2 ينتمي إلى الموضع الثاني، ومن الواضح أنه فعلا ينتمي إليه. وكذلك $\pi(2) = 4$ لأن المفتاح الثاني 4 ينتمي إلى الموضع الرابع، ... وهكذا. وسنعرِّف (identify) التبديل π بالمتتابعة:

$$\pi(1), \pi(2), \ldots, \pi(n)$$

ويُعَرَّف الانعكاس في التبديل $\boldsymbol{\pi}$ (an inversion of the permutation) بأنه زوج $(\pi(i), \pi(j))$ (a pair) بحيث أن $i < j \ \& \ \pi(i) > \pi(j)$. وإذا كان $(\pi(i), \pi(j))$ انعكاسا فإن المفتاحين i-th & j-th (keys) في المتتابعة يكونان غير مرتبين (out of order) بالنسبة لبعضيهما (relative to each other).

مثلا التبديل 2, 4, 1, 5, 3 فيه أربعة انعكاسات (2, 1), (4, 1), (4, 3), (5, 3). وإذا قامت خوارزمية ترتيب بإزالة انعكاس واحد على الأكثر بعد كل مقارنة مفاتيح [مثلا بتبديل عناصر متجاورة (interchanging adjacent elements)، كما تفعل خوارزمية الترتيب بالإدخال] فإن عدد المقارنات التي يتم إجراؤها على المدخلات $\pi(1), \pi(2), \ldots, \pi(n)$ سيكون على الأقل هو عدد الانعكاسات في π. ولذلك فإننا نقوم بدراسة الانعكاسات.

ومن السهل أن نبيِّن أنه يوجد تبديل يحتوي على انعكاسات عددها $n(n-1)/2$. (أي تبديل؟) وهكذا فإن سلوك أسوأ حالة لأي خوارزمية ترتيب تزيل على الأكثر انعكاسا واحدا في كل مقارنة مفاتيح يجب أن يكون في $\Omega(n^2)$.

وللحصول على حد سفلي لمتوسط عدد المقارنات التي تجريها مثل هذه الخوارزميات للترتيب، فإننا نحسب متوسط عدد الانعكاسات في التباديل. وكل تبديل π يمكن أن يزدوج / يقترن (paired with) بتبديله المدوَّر:

$\pi(n), \pi(n-1), \ldots, \pi(1)$ (its transpose permutation). فمثلا مُدَوَّر (2, 4, 1, 5, 3) (transpose of) هو (3, 5, 1, 4, 2). وكل تبديل له مُدَوَّر وحيد (a unique transpose) ومتميِّز (distinct) عن مدوَّره (للقيم $n > 1$). نفرض أن i, j عددان صحيحان بين 1 و n، ونفرض أن $j < i$. بناءً عليه فإن (i, j) يُعَدُّ انعكاسا في واحد بالضبط من التباديل π ومدوَّر π. وهناك $n(n-1)/2$ من مثل هذه الازواج من الأعداد الصحيحة. وبالتالي فكل زوج من التباديل (pair of permutations) يحتوي على انعكاسات عددها $n(n-1)/2$ بين التبديلين، أي على انعكاسات متوسطها $n(n-1)/4$. وهكذا فعلى العموم متوسط عدد الانعكاسات في التبديل هو $n(n-1)/4$. وبالتالي نكون قد أثبتنا النظرية التالية.

نظرية 3-1: أي خوارزمية تقوم بالترتيب (sorting) عن طريق مقارنة المفاتيح وإزالة انعكاس واحد على الأكثر بعد كل مقارنة يجب أن تقوم بإجراء مقارنات عددها على الأقل $n(n\text{-}1)/2$ في أسوأ حالة، وعددها على الأقل $n(n\text{-}1)/4$ في المتوسط (وذلك لعدد n من العناصر).

نظرا لأن خوارزمية الترتيب بالإدخال تقوم بإجراء $n(n\text{-}1)/2$ مقارنة مفاتيح في أسوأ حالة، و $n^2/4$ تقريبا في المتوسط، فهي تُعَدُّ أفضل ما نستطيعه بأي خوارزمية تعمل "محليا" (locally)، مثلا بتبديل عناصر متجاورة (adjacent elements) فقط. وليس واضحا بالطبع الآن ما إذا كانت هناك أي استراتيجية أخرى تستطيع أن تؤدي أداءً أفضل، ولكن إذا كانت هناك خوارزميات أسرع كثيرا فيجب أن تقوم بتحريك العناصر أكثر من موضع واحد في المرة الواحدة (more than one position at a time).

خوارزميات "قَسِّمْ وتَغَلَّبْ"/"فَرِّقْ تَسُدْ" Divide and Conquer Algorithms

المبدأ الذي يعتمد عليه تصميم أي خوارزمية من خوارزميات "قَسِّمْ وتَغَلَّبْ" هو أنه غالبا ما يكون من الأسهل أن نَحِلَّ عدة أجزاء صغيرة من مسألةٍ ما بدلا من حل جزء واحد كبير. ومعظم الخوارزميات التي سنتناولها بإذن الله في هذا الفصل تستخدم طريقة "قَسِّمْ وتَغَلَّبْ"، حيث تُقَسِّم (divide) المسألة إلى أجزاء أصغر (smaller instances) من المسألة نفسها [في هذه الحالة إلى مجموعات أصغر مطلوب ترتيبها]، ثم تَحِلُّ [تَتَغَلَّب على (conquer)] المسائل الأصغر (smaller instances) ارتداديا (recursively) [أي بالطريقة نفسها (by the same method)]، وأخيرا تجمع (combine) هذه الحلول لتحصل على الحل للمدخلات الأصلية. ومن أجل الهروب من الارتداد نقوم بحل بعض الأجزاء الصغيرة من المسألة بطريقة مباشرة (directly). وعلى النقيض من

ذلك فإن خوارزمية الترتيب بالإدخال تقوم بمجرد قَطْع (chopping off) عنصر واحد وإنشاء مسألة جزئية واحدة (one subproblem).

وقد رأينا سابقا - في الفصل الثاني - مثالاً أوليًّا لطريقة قَسِّم وتَغَلَّب، وذلك في خوارزمية البحث الثنائي، حيث قُسِّمت المسألة الأساسية إلى مسألتين جزئيتين، لم نحتج حتى لحل إحداهما. وعموما يمكننا وصف طريقة "قَسِّم وتَغَلَّب" بالمخطط / بالإجراء الهيكلي (skeleton procedure) التالي:

```
solve (I)
{
    n = size(I);
    if (n ≤ smallSize)
        solution = directlySolve(I);
    else
    {
        divide I into I_1, …, I_k.
        for each i ∈ {1, …, k}
            S_i = solve(I_i);
        solution = combine(S_1, …, S_k);
    }
}
```

مخطط / هيكل إجراء قَسِّم وتَغَلَّبْ

ولتصميم خوارزمية "قَسِّم وتَغَلَّبْ" معيَّنَة (a specific Divide–and–Conquer algorithm) يجب أن نحدِّد / نعيِّن (specify) البرامج الفرعية **directlySolve, divide, combine**. ونفرض أن عدد الأجزاء / المسائل الصغرى (smaller instances) التي تُقَسَّم إليها المدخلات (input) يساوي ***k***. وبالنسبة لمدخلات حجمها ***n*** (input size)، نفرض أن

$B(n)$: عـــدد الخطـــوات الـــتي يقـــوم بهـــا البرنـــامج الفرعـــي **directlySolve**،

$D(n)$: عدد الخطوات التي يقوم بها البرنامج الفرعي **divide**،

$C(n)$: عدد الخطوات التي يقوم بها البرنامج الفرعي **combine**.

وبالتـــالي تكـــون الصـــيغة العامـــة للمعادلـــة التكراريـــة (recurrence equation) الـتي تصـف قَـدْر الجهـد / الشـغل الـذي تبذلـه الخوارزمية (amount of work done by the algorithm) هي:

$$T(n) = D(n) + \sum_{i=1}^{k} T\,(size(I_i)) + \mathrm{C}(n), \qquad n > \mathrm{smallSize}$$

وحيث الحالات الأساسية (base cases) هي:

$$T(n) = B(n), \quad n \le \mathrm{smallSize}$$

وبالنسبة لكثير من خوارزميات "قَسِّم وتَغَلَّبْ" تكون إما خطوة **divide** أو خطوة **combine** بسيطة جدا، وتكون معادلة T التكرارية أبسط من الصيغة العامة.

ومن الخوارزميات التي تطبق طريقة "قَسِّم وتَغَلَّبْ" خوارزمية "الترتيب السريع" (Quicksort) وخوارزمية "الترتيب بالدمج" (Mergesort) واللتان سنشرحهما بإذن الله في الصفحات القادمة. وهما تختلفان في طريقة تقسيم المسألة، وكذلك فيما بعد في تجميع (combining) الحلول أو المجموعات الجزئية المُرَتَّبة (sorted subsets)، حيث تتميز خوارزمية الترتيب السريع بـ "تقسيم صعب وتجميع سهل" (hard division, easy combination)، بينما تتميز خوارزمية الترتيب بالدمج بـ "تقسيم سهل، وتجميع صعب". وإذا

تركنا جانبا تَتَبُّع استدعاءات الإجراء، فسنرى أن كل "الشُّغل الحقيقي" (real work) يتم عمله في الجزء "الصعب" (hard section). ويحتوي إجراءا الترتيب (both sorting procedures) على برامج فرعية روتينية (subroutines) لعمل الجزء الصعب، وفي خوارزمية الترتيب السريع خطوة التقسيم هي عملية تجزئة (partitioning) أَمَّا خطوة التجميع فلا تفعل شيئا. بينما في خوارزمية الترتيب بالدمج فخطوة التجميع هي عملية دمج (merging)، أما خطوة التقسيم فتعمل عملية حسابية (calculation) واحدة بسيطة. وكل من الخوارزميتين تقوم بتقسيم المسألة إلى مسألتين جزئيتين / فرعيتين (two subproblems) . ولكنْ هاتان المسألتان الجزئيتان متساويتان في الحجم (of equal size) - في حدود عنصر واحد (within a margin of one element)- في خوارزمية الترتيب بالدمج، بينما في خوارزمية الترتيب السريع تَساوي التقسيم الجزئي (subdivision) غير مضمون. وهذا الاختلاف بين الخوارزميتين يؤدي إلى تباين واضح في خواص الأداء (performance characteristics)، كما سيتضح لنا بإذن الله أثناء تحليل الخوارزميتين.

وهناك خوارزمية تدعى الفرز / الترتيب بالكَوْمَة (HeapSort)، وهي ليست من خوارزميات "قَسِّم وتَغَلَّب" ولكنها تَستخدم عمليات كَوْمة (heap operations) تُعَدُّ من طائفة "قَسِّم وتَغَلَّب". والصيغة المعجَّلة (accelerated form) من خوارزمية الترتيب بالكومة تَستخدم خوارزمية "قَسِّم وتَغَلَّب" أكثر تعقيدا (a more sophisticated Divide-and-Conquer algorithm).

واستراتيجية "قَسِّم وتَغَلَّب" تُطَبق في مسائل عديدة، كمسألة إيجاد

العنصر الأوسط (median element) في مجموعة. [المسألة العامة يطلق عليها "مسألة الاختيار" (selection problem)] كما تُستخدم طريقة "قَسِّم وتَغَلَّب" في صورة أشجار بحث ثنائية (binary search trees) وصيغتها المتوازنة (balanced versions) المعروفة باسم "الأشجار السوداء - الحمراء" (red–black trees). كما تطبق هذه الطريقة في مسائل المسارات في الرسوم البيانية (paths in graphs) كمسألة "الإغلاق المتعدي" (transitive closure)، وفي مسائل خاصة بالمصفوفات والمتجهات، وتلوين الرسوم البيانية (graph coloring)، والحوسبة المتوازية (parallel computation).

الفرز / الترتيب السريع Quicksort

تُعد خوارزمية الترتيب السريع إحدى أُوليات خوارزميات "قَسِّم وتَغَلَّب" التي تم اكتشافها، وقد نَشَرها "هور" (C.A.R. Hoare) عام 1962، ولا تزال للآن إحدى أسرع الخوارزميات العملية.

استراتيجية الترتيب السريع The Quicksort Strategy

تقوم استراتيجية طريقة الفرز / الترتيب السريع على أساس إعادة ترتيب (rearranging) العناصر المطلوب فرزها بحيث أن جميع المفاتيح "الصغيرة" (small keys) تتقدم المفاتيح "الكبيرة" (precede the large keys) في المنظومة [وهذا هو جزء "التقسيم الصعب" (hard division part)]. ثم تقوم الطريقة (الترتيب السريع) بترتيب الجزئين الفرعيين للمفاتيح "الصغيرة" و"الكبيرة" ارتداديا (sorting the two subranges of "small" and "large" keys recursively)، مما ينتج عنه أن تصبح المنظومة بأكملها مرتبة (sorted). وبالنسبة لتنفيذ الطريقة بمنظومة

(an array implementation)، ليس هناك أي شئ مطلوب عمله في خطوة "التجميع" (combination step)، ولكن يمكن لطريقة الترتيب السريع أن تطبَّق أيضا على القوائم (lists)، وفي هذه الحالة فإن خطوة التجميع ستعمل على تعاقب القائمتين (concatenating the two lists). وللبساطة سنصف التنفيذ بمنظومة.

نفـــرض أن ***E*** هـــي منظومـــة العناصـــر، وأن **first** و **last** همـــا مؤشـــرا (indexes) أول عنصـــر وآخـــر عنصـــر - علـــى الترتيـــب - في المـــدى الجزئـــي (subrange) الـــذي تقـــوم خوارزميـــة الترتيـــب الســـريع حاليًّـــا (currently) بترتيبـه. وفي أعلـى مسـتوى (at the top level) تكـون first = 0, last = n - 1 حيث n هي عدد العناصر.

تقوم خوارزمية الترتيب السريع باختيار عنصر - يطلَق عليه "**العنصر الوتدي" (pivot element)**، ويطلَق على مفتاحه "الوتد" (the pivot) - من المدى الجزئي الذي يجب أن تقوم الخوارزمية بترتيبه، وتدفع هذا العنصر خارج الطريق (pulls it out of the way)، أي أنها تحرك العنصر الوتدي إلى متغير محلي (a local variable) تاركةً **فراغا (a vacancy)** في المنظومة. وحاليا نفرض أن العنصر الموجود أقصى يسار المدى الجزئي هو الذي سنختاره ليكون العنصر الوتدي.

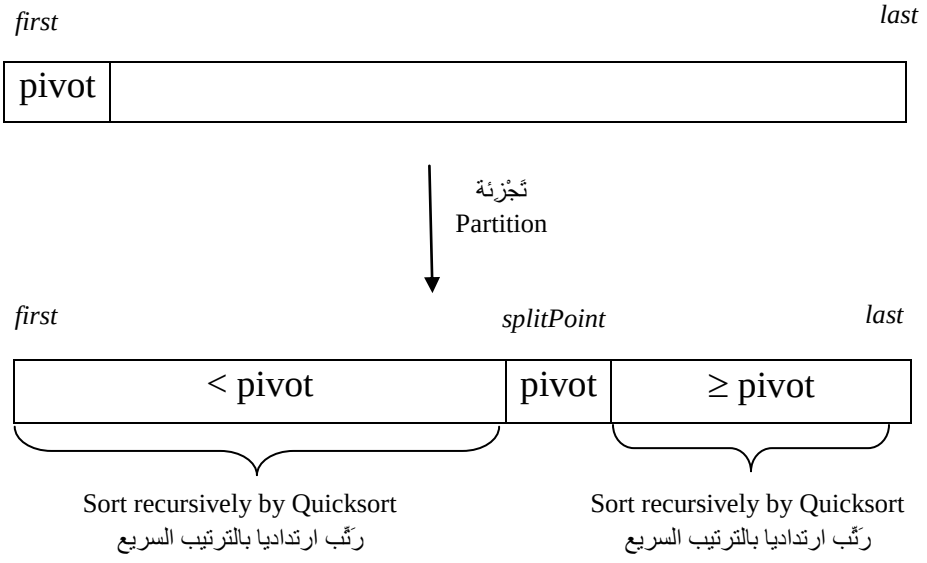
وتقـــوم خوارزميـــة الترتيـــب الســـريع بتمريـــر (passing) الوتـــد [مجال المفتاح فقط (the key field only)] إلى البرنامج الفرعي / الروتيني للتجزئـة (the Partition subroutine)، الـذي يقـوم بإعـادة ترتيـب (rearranging) العناصر **الأخرى**، وإيجاد مؤشر **splitPoint** (an index) بحيث أن:

for first ≤ i < splitPoint E[i].key < pivot (i)

for splitPoint < i ≤ last E[i].key ≥ pivot (ii)

لاحظ أن هناك الآن فراغا (a vacancy) عند splitPoint.

وتقوم خوارزمية الترتيب السريع بإيداع (depositing) **العنصر الوتدي في** E[splitPoint] **والذي هو موضعه الصحيح، ونتجاهل العنصر الوتدي في عملية الترتيب التالية** (the subsequent sorting) **[انظر شكل** 3–5**].**



شكل 3-5
الترتيب السريع Quicksort

وهذا يكمل عملية "التقسيم" (divide process)**، وتستمر خوارزمية الترتيب السريع باستدعاء نفسها ارتداديا لحل المسألتين الفرعيتين الناشئتين بالتجزئة.**

ويمكن لإجراء الترتيب السريع اختيار التجزئة حول أي مفتاح في

المنظومة بين E[first] و E[last] كخطوة قبل التشغيل (a preprocessing step). وأي عنصر يتم اختياره يقوم الإجراء بتحريكه إلى متغير محلي يُدْعَى **pivot**، وإذا **لم يكن** هو E[first]، فإن E[first] يتم تحريكه إلى موضعه، مما يضمن وجود فراغ عند E[first] حين استدعاء البرنامج الفرعي **Partition**. وفيما بعد سنناقش بإذن الله استراتيجيات أخرى لاختيار الوتد.

الخوارزمية 2-3 الترتيب السريع Quicksort

المدخلات (Input): منظومة ***E***، ومؤشِّران first, last (indexes) بحيث أن العناصر $E[i]$ مُعَرَّفة لقيم i: $first \le i \le last$

المخرجات (Output): E[first], …, E[last]

إعادة ترتيب (rearrangement) للعناصر نفسها بحيث تكون مفروزة / مرتَّبَة (sorted).

```
void quicksort ( Element [ ] E, int first, int last )
  if (first < last)
    Element pivotElement = E[first];
    Key pivot = pivotElement.key;
    int splitPoint = partition (E, pivot, first, last);
    E[splitPoint] = pivotElement;
    quickSort (E, first, splitPoint – 1);
    quickSort (E, splitPoint + 1, last);
```

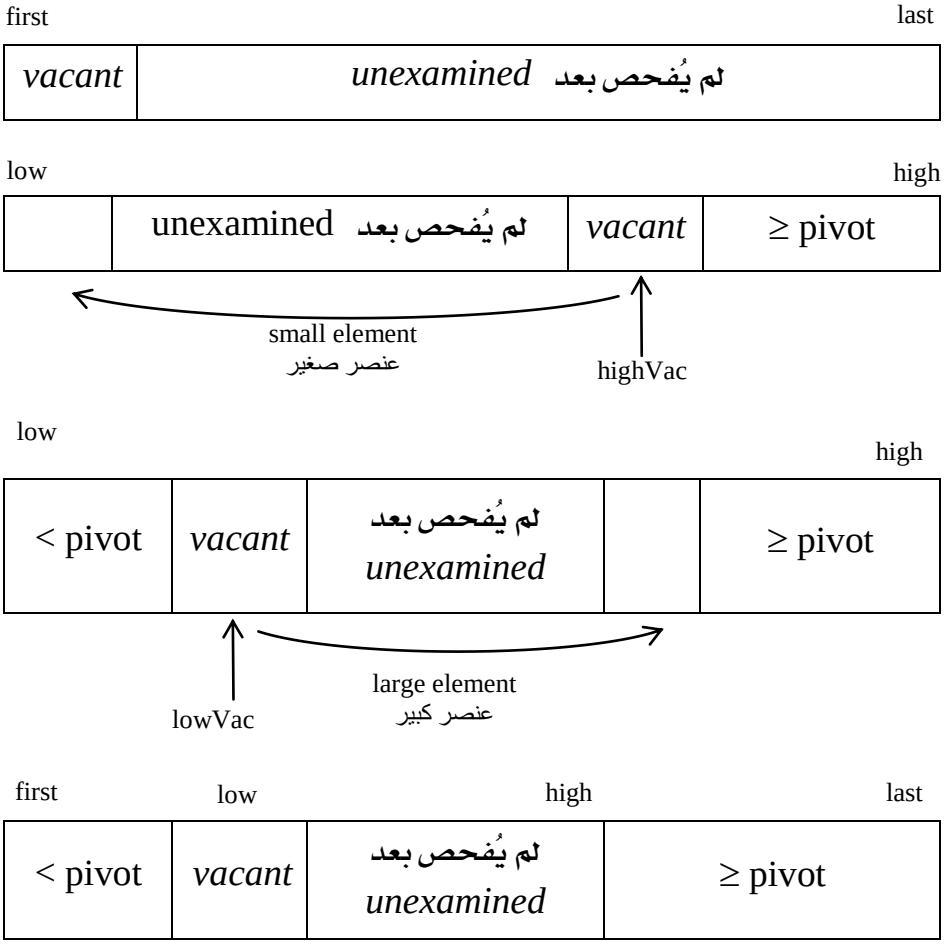
البرنامج الفرعي Partition

The Partition Subroutine

جميع مهمات مقارنة المفاتيح (comparing keys) وتحريك العناصر تتم في البرنامج الفرعي **Partition**. وهناك عدة استراتيجيات مختلفة يمكن استخدامها بهذا البرنامج، وتؤدي إلى خوارزميات لها مزايا وعيوب مختلفة. وسنعرض إحداها هنا. وتعتمد أي استراتيجية على كيفية تنفيذ إعادة ترتيب (rearranging) العناصر. وهناك حل بسيط جدا وهو أن نُحَرِّك العناصر إلى منظومة مؤقتة، ولكن التَّحَدِّي هو أن نعيد ترتيبها بينما هي في أماكنها (in place) أي في المنظومة الأصلية.

وطريقة التجزئة التي سنناقشها هنا هي أساسا الطريقة التي ذَكَرَتْها الخوارزمية الأصلية عند ظهورها. وكحافز (motivation) لاستيعابها تَذَكَّر أن دراستنا السابقة لِلْحَدِّ الأسفل (lower bound) بَيَّنَت أنه لتحسين خوارزمية الترتيب بالإدخال (Insertion Sort) يجب أن نكون قادرين على تحريك عنصر مواضع كثيرة (many positions) بعد مقارنة واحدة (one comparison). وهنا يكون الفراغ / المكان الشاغر (vacancy) ابتداءً (initially) عند **E[first]**. فإذا فرضنا أننا نريد عناصر صغيرة عند النهاية اليسرى للمدى (the left end of the range)، وأننا نريد تحريك عناصر مسافات طويلة كلما كان ذلك ممكنا، فمن المنطقي جدا أن نبدأ البحث للخلف (start searching backward) انطلاقا من **E[Last]** عن عنصر صغير، أَيْ عنصر أصغر من **pivot**. وعندما نجد واحدا، نُحَرِّك هذا العنصر إلى المكان الشاغر (vacancy) [الذي كان عند **first**]. وهذا يترك فراغا / شغورا

جديـدا (a new vacancy) في الموضـع الـذي كـان فيـه العنصـر الصـغير، وسنطلق على هذا الفراغ **highVac**. وهذا الوضـع يُصَوِّره مخططا المنظومتين الأوليتين في شكل 3-6.



شكل 3—6

تعاقب التجزئة خلال دورتها الأولى

The progression of partition through its first cycle

نعلـم أن جميـع العناصـر الـتي مؤشـراتها (indexes) أكـبر مـن **highVac** (إلى **last**) هـي أكـبر مـن أو تسـاوي **pivot**. وإن أمكـن فيجـب تحريـك عنصـر كـبير آخـر إلى **highVac**. ومـرة أخـرى فإننـا نريـد تحريـك عناصـر لمسـافات طويلـة، وبالتـالي فمـن المنطقـي أن نبحـث للأمـام عـن عنصـر "كـبير" (large) هـذه المرة، مبتدئين عند **first+1**. وعندما نجد واحدا، فإننا نحرك هذا العنصر إلى الفراغ (الذي كان عند **highVac**)، وهذا يترك فراغا / شغورا جديدا سنطلق عليه **lowVac**. ونعلم أن جميع العناصر التي مؤشراتها (indexes) أصغر من **lowVac** (إلى **first**) هي أقل من **pivot**.

وأخيرا نقوم بتحديث (updating) المتغيرين **low, high** كما هو مبيَّن في الصف الأخير في شكل 3–6 استعدادا لدورة (cycle) أخرى. وكما كان الحال عند بداية الدورة الأولى، فإن العناصر الواقعة في المدى من **low +** **1** إلى **high** لم يتم فحصها بعد، و **E[low]** فارغ. ويمكننا تكرار (repeating) الدورة التي تم وصفها، حيث نبحث للخلف (search backward) مبتدئين من **high** (from) عن عنصر صغير، ونحركه إلى الفراغ **low**، ثم نبحث للأمام مبتدئين من **low + 1** عن عنصر كبير، ونحركه إلى **highVac**، وهذا ينشأ عنه فراغ عند lowVac، وهو الموضع الذي حَرَّكنا من عنده العنصر الكبير. وأخيرا يتقابل **lowVac** و **highVac** مما يعني أن جميع العناصر قد تمت مقارنتها مع الوتد.

الإجراء **Partition** يتم تنفيذه بتكرار الدورة (cycle) التي وصفناها، واستخدام برنامجين فرعيين (2 subprograms):

(أ) **extendLargeRegion**: وهذا يقوم بعملية مسح (scanning) إلى

الخلـف (backward) مبتـدئا مـن النهايـة اليمنـى (right end)، ومتجاهلا / مهملا (passing over) العناصر الكبيرة إلى أن: (i) يجد عنصرا صغيرا ويحركه إلى الفراغ الموجود عند النهاية اليسرى. أو (ii) يتحرك إلى هذا الفراغ دون أن يجد أي عنصر صغير. وفي هذه الحالة الأخيرة تكون التجزئة (partitioning) قد اكتملت. أما في الحالة الأولى فإن الموضع الفارغ الجديد (new vacant position) يُعاد (returned)، ويتم استدعاء البرنامج الفرعي الروتيني الآخر.

(ب) **extendSmallRegion**: وهذا شبيه بالبرنامج الفرعي السابق إلا أنه يقوم بعملية المسح إلى الأمام (forward) مبتدئا من النهاية اليسرى، ومهملا / متجاهلا العناصر الصغرى إلى أن يجد عنصرا كبيرا ويحركه إلى الفراغ الموجود عند النهاية اليمنى، أو إلى أن ينتهي من البيانات (runs out of data).

في البداية تكون كل من منطقة المفاتيح الصغرى (small–key region) - الموجودة يسار **low** - ومنطقة المفاتيح الكبرى (large–key region) - الموجودة يمين **high** - منطقة خاوية (empty)، ويكون الموضع الشاغر / الفراغ (vacancy) موجودا عند النهاية اليسرى (left end) للمنطقة الوسطى (middle region) [والتي تكون هي كل المدى (the whole range) في هذه اللحظة (أي في البداية)]. وأي استدعاء لأي من البرنامجين الفرعيين (أ) أو (ب) يقلِّص (shrinks) المنطقة الوسطى بواحد على الأقل، وينقل / يحرِّك الفراغ (vacancy) إلى النهاية الأخرى للمنطقة الوسطى. كذلك يضمن لنا البرنامجان الفرعيان أن تذهب العناصر الصغرى

فقط إلى منطقة المفاتيح الصغرى، وتذهب العناصر الكبرى فقط إلى منطقة المفـــاتيح الكـــبرى. وهـــذا يمكننـــا أن نـــراه مـــن الشـــروط اللاحقـــة (postconditions). وعندما تتقلص / تنكمش (shrinks) المنطقة الوسطى إلى موضـع واحـد، فـإن هـذا الموضـع يكـون هـو الفـراغ (vacancy)، ويعـاد (returned) على أنه **splitPoint**.

الخوارزمية 3-3 **التجزئة** Partition

المدخلات (Input): المنظومة ***E***، **pivot**، المفتاح (key) الـذي نقـوم حولـه بالتجزئة، والمؤشران: **first, last**، بحيث أن العناصر **E[*i*]** تكون مُعَرَّفة (defined) للفترة first+1 ≤ i ≤ last، و **E[first]** خاوي (vacant). ونفرض أن first < last.

المخرجـــات (Output): نفـــرض أن **splitPoint** هـــي القيمـــة المعـــادة (returned value). العناصر الموجودة أصلا (originally) في: **first+1, …, last** يعاد ترتيبها (rearranged) في مَدَيَين جزئيَّيْن (two subranges)، بحيث أن:

1) مفاتيح العناصر E[first], …, E[splitPoint - l] تكون أقل من **pivot**.

2) مفاتيح العناصر E[splitPoint+l], …, E[last] تكون أكبر من أو تساوي **pivot**. وأيضًا first ≤ splitPoint ≤ last، و E[splitPoint] فارغ / شاغر (vacant).

الإجراء

```
int partition ( Element[ ] E, Key pivot, int first, int last )
    int low, high;
1.  low = first; high = last;
2.  while (low < high)
3.      int highVac = extendLargeRegion (E, pivot, low, high);
4.      int lowVac = extendSmallRegion (E, pivot, low+1, highVac);
5.      low = lowVac; high = highVac – 1;
6.  return low;   // This is the splitPoint.
/ ** Postconditions for extendLargeRegion;
   * The rightmost element in E[lowVac + 1], …, E[high]
   * whose key is < pivot is moved to E[lowVac] and
   * the index from which it was moved is returned.
   * If there is no such element, lowVac is returned.
   */
int extendLargeRegion ( Element[ ]E, Key pivot, int lowVac, int high )
    int highVac, curr;
    highVac = lowVac;   // In case no key < pivot.
    curr = high;
    while (curr > lowVac)
        if (E[curr].key < pivot)
            E[lowVac] = E[curr];
            highVac= curr;
            break;
        curr– – ; / / Keep looking.
    return highVac;
```

```
/ ** Postconditions for extendSmallRegion; (Exercise) */
int extendSmallRegion ( Element[ ] E, Key pivot, int low, int highVac )
{
  int lowVac, curr;
  lowVac = highVac; // In case no key ≥ pivot.
  curr = low;
  while (curr < highVac)
  {
    if (E[curr].key ≥ pivot)
    {
      E[highVac] = E[curr];
      lowVac = curr;
      break;
    }
    curr ++;  //  Keep looking.
  }
  return lowVac;
}
```

إجراء خوارزمية التجزئة

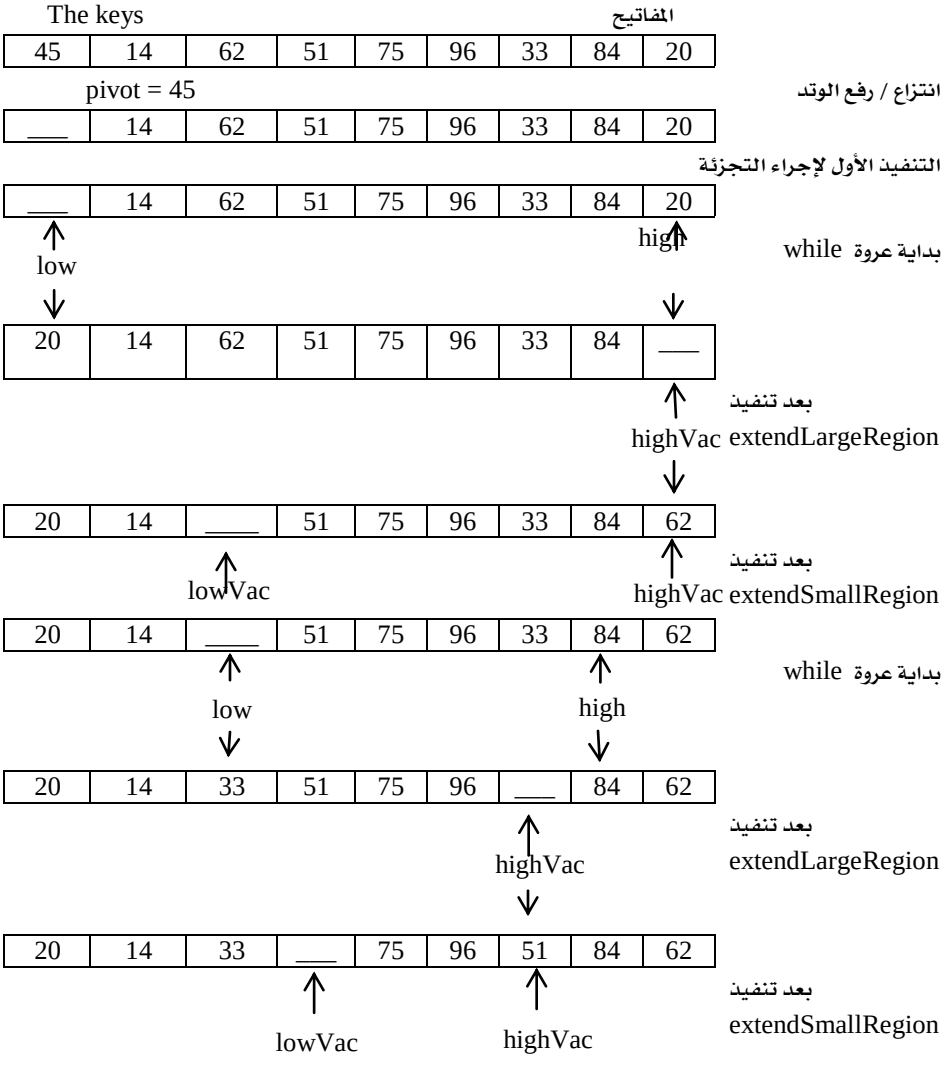
Procedure for the Algorithm Partition

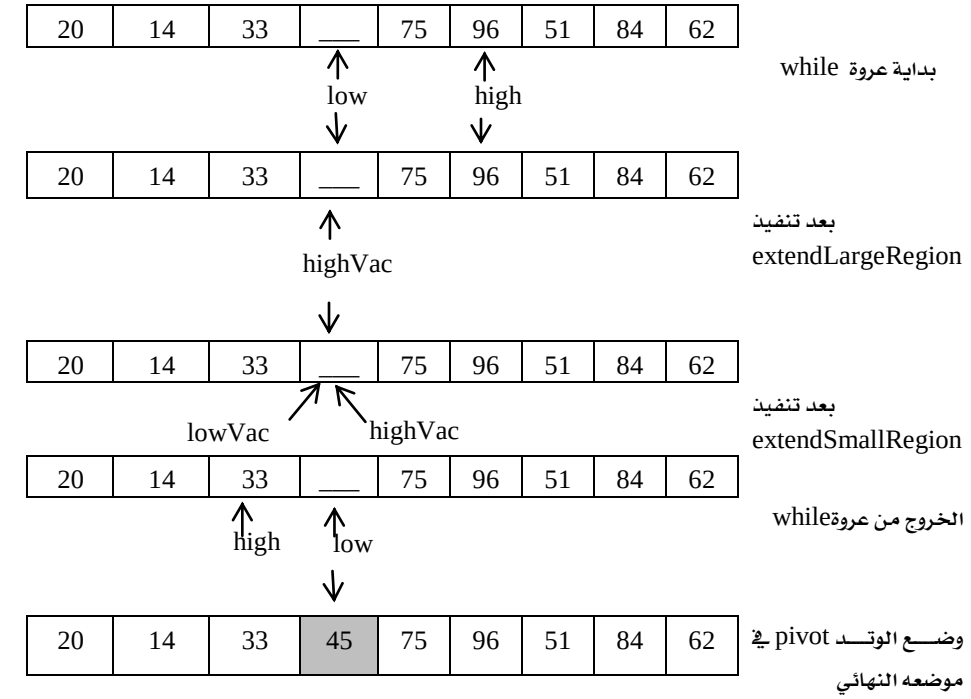
ملاحظة: لِتَجَنُّب مقارنات إضافية داخل عروة **while** في الإجراء **Partition**، لا يوجد أي اختبار للشرط **highVac = lowVac** قبل السطر 5، والذي يعني أن جميع العناصر قد تمت تجزئتها. وبناءً عليه فإن **high** قد تكون أقل من **low** بواحد عندما تتوقف / تنتهي العروة (loop terminates)، بينما الواجب منطقيا أن يكونا متساويين. ولكن نظرا لأننا بعد انتهاء العروة لا نصل إلى (do not access) **high**، فهذا الاختلاف لا قيمة له، ولا ضرر منه.

مثال 3-1: الشكل التالي (شكل 3-7) يوضح مثالا لتطبيق خوارزمية الترتيب السريع **Quicksort**، حيث نبدأ بالمفاتيح:

45	14	62	51	75	96	33	84	20

والشكل يبيِّن تفاصيل عملية التجزئة (detailed operation of Partion) عند استدعاء الإجراء أول مرة فقط . ولاحظ أن العناصر الصغرى (smaller elements) تتجمع / تتراكم يسار **low**، بينما العناصر الكبرى (larger elements) تتجمع / تتراكم يمين **high** .





الجزء الأول من التجزئة Partition (التفاصيل غير مبيَّنة)

Partition first section

14	20	33

الجزء الثاني من التجزئة Partition (التفاصيل غير مبيَّنة)

Partition second section

62	51	75	84	96

الجزء الفرعى الأيسر من التجزئة

Partition left subsection

51	62

الجزء الفرعى الأيمن من التجزئة

Partition right subsection

84	96

المنظومة النهائية

Final array

14	20	33	45	51	62	75	84	96

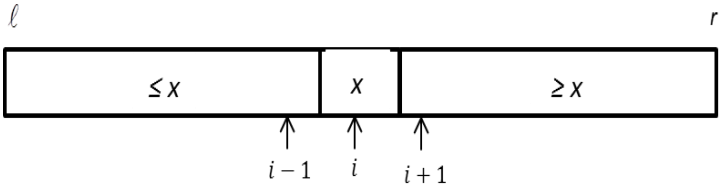
شكل 3-7

مثال للترتيب السريع Example of Quicksort :

وفيما يلى - وقبل إعطاء مثال عددى آخر يوضح كيفية تطبيق طريقة الترتيب السريع - نلخص خطوات هذه الطريقة بشبه كود (pseudo-code)/ كود زائف باستخدام الرموز.

(شبه كود) خوارزمية الترتيب السريع

أ) نقوم بتجزئة / بتقسيم (partitioning/splitting) المنظومة E بحيث تتحقق الشروط التاليه:



$$E_j \le x \qquad \forall\, j < i \qquad \text{(i)}$$

$$E_j \ge x \qquad \forall\, j > i \qquad \text{(ii)}$$

$$E_i = x \qquad \text{(iii)}$$

أى أن الوتد (pivot) موجود فى موضعه النهائي، حيث x هو أحد عناصر المنظومة E.

ب) لترتيب عناصر المنظومة E:

(i) نرتب عناصر المجموعة الجزئيه اليسرى (left subarray) $\ell \to i-1$

(ii) نرتب عناصر المجموعة الجزئيه اليمنى (right subarray) $i+1 \to r$

وأما لترتيب عناصر كل من المنظومتين الجزئيتين اليسرى و اليمنى فإننا نتبع الاستراتيجية / الطريقة نفسها (خوارزمية الترتيب السريع)، أى اننا

نحصل على خوارزمية ارتدادية (recursive algorithm). فهذه الطريقة (الترتيب السريع) هى مثال لطريقة قسم وتغلب (divide and conquer approach).

خوارزمية الترتيب السريع Q-S (E: array, ℓ, r)

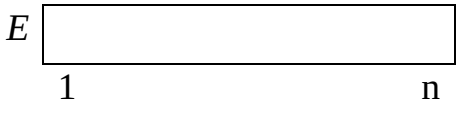
المدخلات: المنظومة E (array) المراد ترتيبها.

ومؤشران ℓ, r (left and right indices).

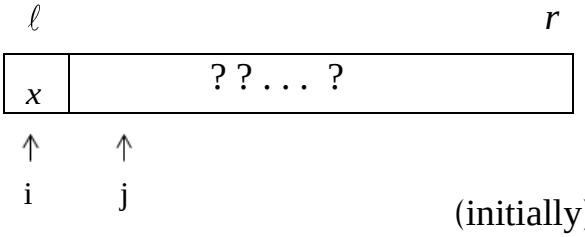
المخرجات: المنظومة E مرتبة ترتيبًا تصاعديًّا (sorted in an ascending order).

```
void Q-S ( E : array, ℓ , r)
    if ( ℓ < r )
        Partition ( E, ℓ, r, i )      // i is output
        Q-S ( E,  ℓ, i - 1 )          // sort the left subarray
        Q-S ( E,  i + 1, r )          // sort the right subarray
```

ولاستدعاء منظومة E بجميع عناصرها لترتيبها كاملة نستدعى الاجراء $Q\text{-}S\ (E, 1, n)$ حيث n هو عدد عناصر المنظومة.

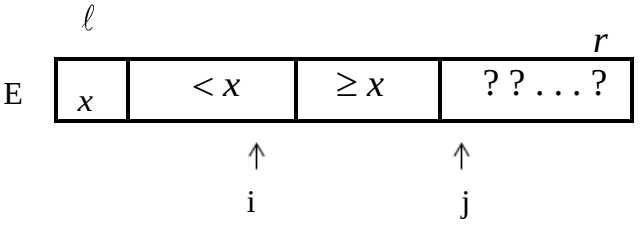


وأما عن كيفية تجزئة / تقسيم (partitioning / splitting) المنظومة:



فى البداية (initially)

At the j-th step **وعند الخطوة رقم j**




```
void partition ( E : array, ℓ, r, i )
    x = E_ℓ
    i = ℓ
    for j = ℓ + 1 to r
        if ( E_j < x )
            i = i + 1
            swap ( E_i , E_j )
    swap ( E_i , E_ℓ )      // put the pivot in its proper place
```

cost = # of comparisons

$= r - \ell$

$= n - 1$ $\qquad (\ell = 1, r = n)$

مثـال 3-2: الشـكل التـالى (شـكل 3-7- أ) يعـرض بصـورة مختصـرة مثـالا آخـر لتطبيق خوارزمية الترتيب السريع Quicksort، حيث نبدأ بالمفاتيح:

20	3	28	12	1	29	11	14	25

التنفيذ الأول لعملية split

الخط الرأسي يشير إلى الحد الفاصل بين المفاتيح

keys < x والمفاتيح keys ≥ x

20	3 ↑ Swap in place	28 ↑ Stay	12 ↑ Swap with 28	1	29	11	14	25
20	3	12	28	1 ↑ Swap with 28	29	11	14	25
20	3	12	1	28	29 ↑ Stay	11 ↑ Swap with 28	14	25
20	3	12	1	11	29	28	14 ↑ Swap with 29	25
20	3	12	1	11	14	28	29	25 ↑ Stay
14	3	12	1	11	20	28	29	25

قَسِّمْ الجزء الأول (التفاصيل غير مبينة)

11	3	12	1	14

قَسِّمْ الجزء الأول

1	3	11	12

قَسِّمْ الجزء الأول

1	3

الأقسام / الأجزاء التي تحتوي على مفتاح واحد فقط تعد مرتبة. و الآن نعود إلى الجزء الثاني من التقسيم الأول

x

25	29	28

قَسِّمْ

29	28	25

الأجزاء التي تحتوي على مفتاح واحد تعد مرتبة.

القائمة النهائية

1	3	11	12	14	20	25	28	29

شكل 3-7- أ

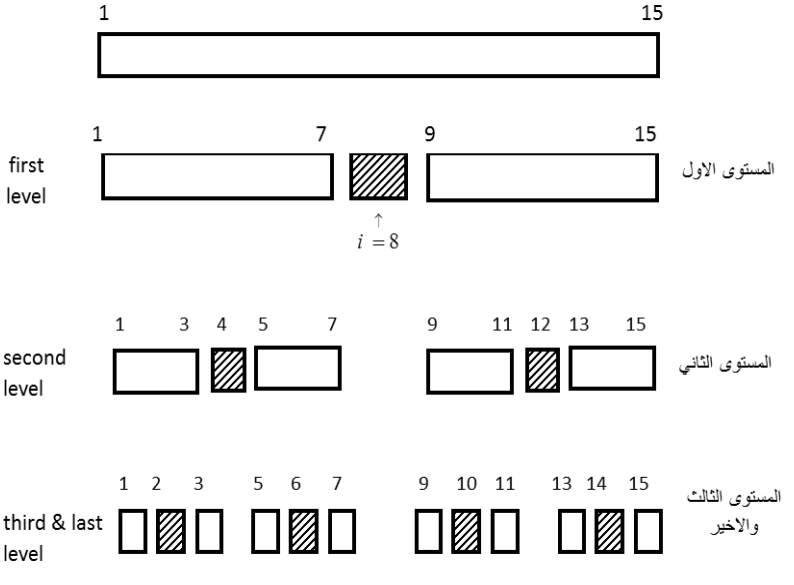
مثال آخر للترتيب السريع

تحليل خوارزمية الترتيب السريع Analysis of Quicksort

(i) أحسن حالة Best case

نحتاج أولا لمعرفة أن عملية التجزئة **partition** ستكلفنا عدد $n-1$ من المقارنات لتقسيم (splitting) منظومة مكونه من n عنصر. نفرض أن $n = 2^k$-1، وأن الوتد (pivot) سيقوم بتقسيم المنظومة E إلى منظومتين فرعيتين متساويتين (2 equal subarrays) وأن ذلك يحدث فى كل خطوه.

مثلا إذا كانت $k = 4$، أى أن $n = 2^4 - 1 = 15$



كم عدد العمليات (operations) الإجمالى لدينا؟ هذا العدد لا يزيد عن حاصل ضرب عدد المستويات وأقصى عدد من العمليات فى المستوى الواحد. أى أن

$B(n) \leq$ (*number of levels*) * (*max. number of operations / level*)

$$= \quad (k-1) \quad * \quad (n-1)$$
$$\approx \quad k * n \quad \approx \quad n \log n$$

أى أن العدد الإجمالى للعمليات

total number of operations $\approx n \log n$

ولاشتقاق النتيجة رياضيا: نفرض أن $n = 2^k$ حيث k عدد صحيح موجب (+ ve integer).

$(\frac{n}{2}-1)$ عنصراً $\frac{n}{2}$ عنصراً

عدد العمليات Q (no. of operations) يحقق العلاقتين:

$$Q(n) = (n-1) + Q(\frac{n}{2}-1) + Q(\frac{n}{2})$$

$$Q(1) = 0$$

وللتبسيط (to simplify) نأخذ فى الاعتبار العلاقتين:

$$Q(n) = n + 2Q(\frac{n}{2})$$

$$Q(1) = 0$$

وهذه علاقة ارتدادية / رجعية (recurrence relation) نقوم بحلها كما يلي:

$$Q(n) = n + 2Q(\frac{n}{2})$$
$$= n + 2(\frac{n}{2} + 2Q(\frac{n}{4}))$$
$$= n + n + 4Q(\frac{n}{4})$$
$$= 2n + 2^2 Q(n / 2^2)$$
$$\vdots$$
$$= in + 2^i Q(n / 2^i); \qquad i = 1, 2, \ldots, k$$

وبوضع $i = k$ نحصل على

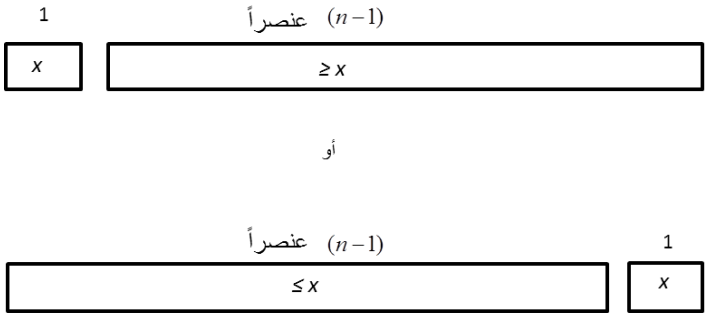
$$Q(n) = kn + 2^k Q(n / 2^k)$$
$$= kn + nQ(1) = kn + 0 \quad = n \log n$$

(ii) أسوأ حالة Worst case

يقوم الإجراء Partition بمقارنة كل مفتاح مع الوتد pivot. وبالتالي فإن كان مدى (range) المنظومة الذي يعمل فيه هو k موضع (k positions) فسيكون عدد المقارنات (key comparisons) $k-1$. الموضع الأول شاغر (vacant)، فإن كان E[first] عنده أصغر مفتاح (smallest key) في المدى الذي يتم تقسيمه (being split) فإنَّ splitPoint = first، وكل ما تم إنجازه هو تقسيم المدى إلى مدى جزئي فارغ (empty subrange) [المفاتيح أصغر من pivot] ومدى جزئي فيه k-1 عنصر. وهكذا فإن كان pivot هو أصغر مفتاح في كل مرة يتم فيها استدعاء Partition، فإنَّ العدد الكلي لمقارنات المفاتيح التي تم إجراؤها هو

$$\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}$$

أى أن أسـوأ حالـه تحـدث عنـدما يكـون الوتـد هـو أصـغر عنصـر [أو أكبر عنصر (وهذه أسوأ لأنها تشتمل على تحركات أكثر)] فى كل مرحلة (in each phase). وهـذه الحالـه تحـدث إن كانـت المنظومـة E مرتبـة ترتيبـا تصاعديا، واخترنا العنصر الأول وتداً. وتوضيح ذلك:



ويكون عدد المقارنات:

$$W(n) = (n-1) + W(n-1) + W(0);\ W(1) = 0,\ W(0) = 0$$

حيث: $(n-1)$ تكلفة التجزئة

$W(n-1)$ عدد مقارنات عناصر المنظومة الفرعية اليسرى (أو اليمنى)

$W(0)$ عدد مقارنات عناصر منظومة فرعية فارغة.

أي أن:

$$
\begin{aligned}
W(n) &= (n-1) + W(n-1) \\
&= (n-1) + (n-2) + W(n-2) \\
&\vdots \\
&= (n-1) + (n-2) + (n-3) \ldots + 1 + W(1) \\
&= (n-1) + (n-2) + (n-3) + \ldots + 1 \\
&= \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = \Theta(n^2)
\end{aligned}
$$

أى أن العدد الإجمالى للمقارنات يساوى

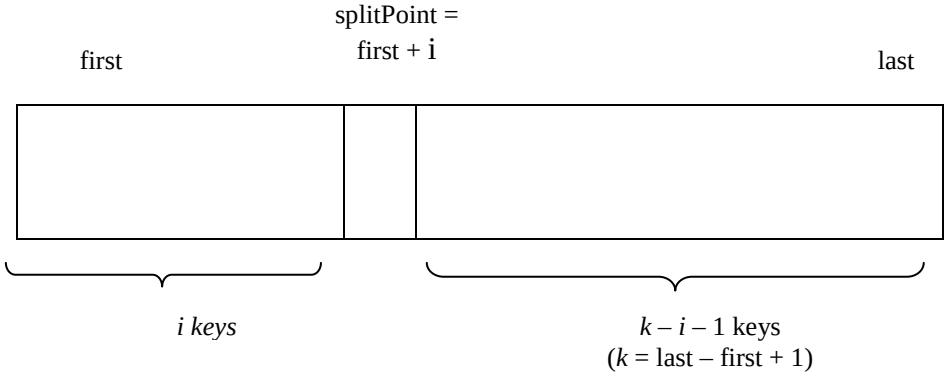
$$W(n) = \frac{n(n-1)}{2} = \Theta(n^2)$$

وهـذا يُعَـدُّ سـيئا بدرجـة السـوء نفسـها في حالـة خوارزميـة الترتيـب بالإدخال [أوخوارزمية الترتيب الفقاعي (bubble sort)]، وكذلك خوارزمية تُدْعَى "الترتيب بالأكبر" (Maxsort). ومن الغريب أن أسوأ حالة هذه تحدث – كما أشرنا سابقا– عندما تكون المفاتيح مُرَتَّبة تصاعديا!

(iii) السلوك المتوسط Average Behavior

رأينا سابقا أنه إذا قامت أَيُّ خوارزميةِ ترتيبٍ بإزالة انعكاس واحد على الأكثر من تبديل المفاتيح بعد كل مقارنة، فيجب أن تقوم الخوارزمية بإجراء $(n^2-n)/4$ مقارنة على الأقل في المتوسط (نظرية 3–1). إلا أن خوارزمية الترتيب السريع لا ينطبق عليها هذا القيد / الشرط (restriction). وتستطيع خوارزمية التجزئة Partition تحريك عنصر عبر جزء كبير من المنظومة، لاغيةً (eliminating) انعكاسات يصل عددها إلى n-1 بحركة واحدة. وتستحق خوارزمية الترتيب السريع اسمها هذا بسبب سلوكها المتوسط.

نفرض أن المفاتيح متباينة (distinct)، وأن جميع تباديل المفاتيح متساوية الاحتمالات (equally likely). ونفرض أن ***k*** هي عدد العناصر في مدى المنظومة التي نقوم بترتيبها، وأن ***A(k)*** هي متوسط عدد مقارنات المفاتيح التي تتم لأكثر من مدى من هذه السعة (for ranges of this size). ونفرض أنه في المرة التالية التي تُنَفَّذ فيها الخوارزمية Partition سَيُوضع **pivot** في الموضع ذي الترتيب ***i*** (i-th position) في هذا المدى الجزئي (شكل 3–8)، بادئين العد من الصفر 0. وتقوم Partition بإجراء k-1 مقارنة مفاتيح، وأي مدى جزئي سيتم ترتيب عناصره بعد ذلك سيحتوي على مدى جزئي من ***i*** عنصر، وآخر من $k-1-i$ عنصر (على الترتيب).



شكل 3–8

السلوك المتوسط لخوارزمية الترتيب السريع

Average behavior of Quicksort

ومن المهم بالنسبة لتحليلنا أنه بعد انتهاء خوارزمية Partition ألا تكون قد تمت أي مقارنة بين أي مفتاحين في المدى الجزئي (first, …, splitPoint–1)، وبالتالي تكون جميع تباديل المفاتيح في هذا المدى

الجزئي لا تزال متساوية الاحتمالات (still equally likely). والكلام نفسه ينطبق على المدى الجزئي (splitPoint+1, …, last). وهذا يبرر التكرار (recurrence) التالي.

وأي موضع محتمل i لنقطة التقسيم (split point) له احتمال متساوي (equally likely) مع احتمالات باقي المواضع (احتماله $1/k$)، وبجعل $k = n$ نحصل على المعادلة التكرارية (recurrence equation):

$$A(n)=n-1+\sum_{i=0}^{n-1}\frac{1}{n}(A(i)+A(n-1-i)) \quad for\ n\geq 2$$
$$A(1)=A(0)=0$$

نلاحظ أن الحدود التي صيغتها $A(n-1-i)$ في المجموع Σ تبدأ بالحد $A(n-1)$ إلى أن تصل إلى الحد $A(0)$، وبالتالي فمجموعها هو نفسه مجموع الحدود التي صيغتها $A(i)$. ونظرا لأن $A(0) = 0$، فنحصل على

$$A(n)=n-1+\frac{2}{n}\sum_{i=1}^{n-1}A(i) \quad for\ n\geq 1 \qquad (1)$$

وهذه معادلة تكرارية أكثر تعقيدا من المعادلات التكرارية التي رأيناها سابقا، وذلك لأن قيمة $A(n)$ تعتمد على جميع القيم السابقة. ويمكننا أن نستخدم ذكاءنا لحل المعادلة التكرارية أو أن نخمن (guess) حلا ثم نثبت صحته بالاستقراء. وهذه الطريقة الأخيرة تناسب بصورة خاصة الخوارزميات الارتدادية. ومن المفيد من الناحية التعليمية أن نرى الطريقتين، ولذلك فسنتناولهما بإذن الله فيما يلي.

لصياغة حل تخميني لقيمة $A(n)$ نأخذ حالةً تعمل فيها خوارزمية الترتيب السريع بطريقة جيدة. ونفرض أنه في كل مرة تنفَّذ فيها الخوارزمية

Partition، فإنها تُجَزِّئ (partitions) المدى إلى قسمين فرعيين متساويين (two equal subranges). ونظرا لأننا نود فقط تحديد تقدير (an estimate) يساعدنا في تخمين مدى سرعة طريقة الترتيب السريع في المتوسط، فسنُقَدِّر حجم / سعة (size) كل من القسمين الفرعيين بأنه $n/2$. والمعادلة التكرارية التالية تصف عدد المقارنات التي تُجرى

$$Q(n) \approx n + 2\ Q(n/2)$$

ويمكن رياضيًا إثبات أن هذه المعادلة تؤدي إلى النتيجة: $Q(n) \in \Theta(n \log n)$. وبالتالي فإن كان E[first] قريبا من العنصر الأوسط (close to the median) في كل مرة يُقَسَّم فيها المدى (range is split)، فإنَّ عدد المقارنات التي تجريها خوارزمية الترتيب السريع ستكون في $\Theta(n \log n)$. وهذا أفضل بكثير من $\Theta(n^2)$. ولكن إذا كانت جميع تباديل المفاتيح (all permutations of the keys) متساوية الاحتمالات (equally likely) ، فهل هناك حالات جيدة كافية (enough good cases) لأن تؤثر على المتوسط؟ سنثبت فيما يلي أن الإجابة على هذا السؤال هي: نعم.

نظرية 3–2:

إذا كانت $A(n)$ مُعَرَّفة بمعادلة التكرار (1)، فإن $A(n)$ تحقق العلاقة

$$\forall n \geq 1 \quad A(n) \leq cn\ ln\ n$$

حيث c ثابتٌ ما. (ملاحظة: قيمة c ستظهر في برهان النظرية).

البرهان:

سنبرهن النظرية بالاستقراء على n (induction on)، حيث n هي عدد العناصر المطلوب ترتيبها. الحالة الأساسية هي $n=1$. ولدينا

$A(1) = 0,\ c\ 1\ ln\ 1 = 0$. وبالنسبة للقيم $n > 1$: نفرض أن $A(i) \le c\ i\ ln(i)$ حيث $1 \le i < n$، و c هو الثابت نفسه الذي يظهر في نص النظرية. باستخدام المعادلة (1) وفرض الاستقراء نحصل على

$$A(n) = n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} A(i) \le n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} ci \ln(i)$$

ويمكننا أن نضع حدا أعلى للمجموع بالتكامل [انظر العلاقة (16) بالفصل الأول]:

$$\sum_{i=1}^{n-1} ci \ln(i) \le c \int_1^n x \ln x\, dx$$

وباستخدام العلاقة (15) بالفصل الأول نحصل على

$$\int_1^n x \ln x\, dx = \tfrac{1}{2} n^2 \ln(n) - \frac{1}{4} n^2 + \frac{1}{4}$$

وبالتالي

$$A(n) \le n - 1 + \frac{2c}{n}\left(\tfrac{1}{2} n^2 \ln(n) - \frac{1}{4} n^2 + \frac{1}{4}\right)$$

$$= cn \ln n + n\left(1 - \frac{c}{2} + \frac{c}{2n^2}\right) - 1$$

ولإثبات أن $A(n) \le cn\ ln\ n$ يكفي أن نثبت أن كلا من الحد الثاني والحد الثالث سالب أو صفر. الحد الثاني يكون سالبا أو صفرا للقيم $c \ge 2$، فدعنا نجعل $c = 2$، وبالتالي نستنتج أن $A(n) \le 2\ n \ln n$.

وبتحليل مشابه يمكننا إثبات أن $A(n) > cn \ln n$ لأي قيمة $c < 2$.

ونظرا لأن $ln\ n \approx 0.693 \lg n$، فلذلك نصل إلى النتيجة التالية:

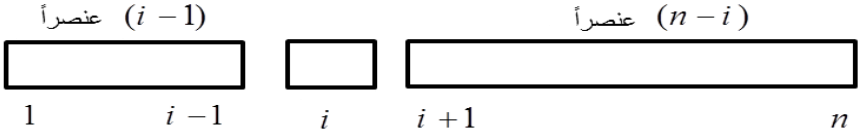
نتيجة (corollary)

بفرض أن جميع تباديل المدخلات (input permutations) متساوية الاحتمالات (equally likely)، ففي المتوسط يكون عدد المقارنات التي تجريها خوارزمية الترتيب السريع (الخوارزمية 3–2) على مجموعات حجمها n (sets of size) مساويا تقريبا $1.386\ n \lg n$ لقيم n الكبيرة [أى مساوياً تقريباً $1.4\ n \lg n$].

وفيما يلى إثبات آخر لهذه النتيجة.

برهان آخر لهذه النتيجة

بفرض أن العنصر الوتدى سيتم اختياره عشوائيا (randomly)، وأنه يقسم (splits) المنظومة كما يوضح الشكل التالي:



سيكون الوتد فى الموضع رقم i (i-th position)، حيث $i = 1, 2, \ldots, n$، باحتمال يساوى $\frac{1}{n}$.

بفرض أن $A(n)$ تمثل متوسط درجه التعقيد (average complexity) لترتيب/ لفرز n عنصراً (sorting n elements)، فإن

$$A(n) = (n-1) + \frac{1}{n}\left[\sum_{i=1}^{n} \left(A(i-1) + A(n\text{-}i) \right)\right]; \qquad n \geq 2$$

$$A(0) = A(1) = 0$$

ونظراً لأن

$$\sum_{i=1}^{n} A(i-1) = A(0) + A(1) + \ldots + A(n-1)$$

وكذلك

$$\sum_{i=1}^{n} A(n-i) = A(n-1) + A(n-2) + \ldots + A(1) + A(0)$$

فلذلك نحصل على العلاقة

$$A(n) = (n-1) + \frac{2}{n} \sum_{i=1}^{n} A(i-1) \qquad (*)$$

وبضرب طرفى العلاقة $(*)$ فى n نحصل علي

$$n\,A(n) = n(n-1) + 2\sum_{i=1}^{n} A(i-1)$$

$$\Rightarrow (n-1)A(n-1) = (n-1)(n-2) + 2\sum_{i=1}^{n-1} A(i-1)$$

وبطرح طرفى العلاقتين الأخيرتين الأيمن والأيسر، نحصل علي

$$n\,A(n) - (n-1)A(n-1) = 2(n-1) + 2A(n-1)$$

$$\Rightarrow nA(n) = 2(n-1) + (n+1)A(n-1)$$

$$\Rightarrow \frac{A(n)}{n+1} = 2\frac{n-1}{n(n+1)} + \frac{A(n-1)}{n}$$

وباستخدام التعويض

$$F(n) = \frac{A(n)}{n+1}$$

نحصل على:

$$F(n) = 2\,\frac{n-1}{n(n+1)} + F(n-1); F(0) = 0,\ F(1) = 0$$

وبحل هذه العلاقة الارتدادية بالمفكوك (solving by expansion)، نحصل علي:

$$
\begin{aligned}
F(n) &= 2\frac{n-1}{n(n+1)} + 2\frac{(n-1)-1}{(n\text{-}1)(n-1+1)} + F(n-2) \\
&= 2\frac{n-1}{n(n+1)} + 2\frac{(n-1)-1}{(n-1)(n-1+1)} + \frac{(n-2)-1}{(n-2)(n-2+1)} + F(n-3) \\
&\quad \vdots \\
&= 2\sum_{i=2}^{n}\frac{(i-1)}{i(i+1)} + \mathrm{F}(1) \\
&= 2\sum_{i=2}^{n}\frac{(i-1)}{i(i+1)} \\
&= 2\sum_{i=2}^{n}(\frac{2}{i+1} - \frac{1}{i})
\end{aligned}
$$

وبإضافة حد وطرحه:

$$+2\sum_{i=2}^{n}\frac{1}{i} - 2\sum_{i=2}^{n}\frac{1}{i}$$

نحصل على العلاقة:

$$F(n) = 2\sum_{i=2}^{n}\frac{1}{i} + 4\sum_{i=2}^{n}(\frac{1}{i+1} - \frac{1}{i})$$

ولكن

$$
\begin{aligned}
4\sum_{i=2}^{n}(\frac{1}{i+1} - \frac{1}{i}) &= 4\left[(\frac{1}{3} - \frac{1}{2}) + (\frac{1}{4} - \frac{1}{3}) + (\frac{1}{5} - \frac{1}{4}) + \ldots + (\frac{1}{n+1} - \frac{1}{n})\right] \\
&= 4\left[\frac{1}{n+1} - \frac{1}{2}\right] = 4\frac{2-n-1}{2(n+1)} \\
&= -2\frac{n-1}{n+1}
\end{aligned}
$$

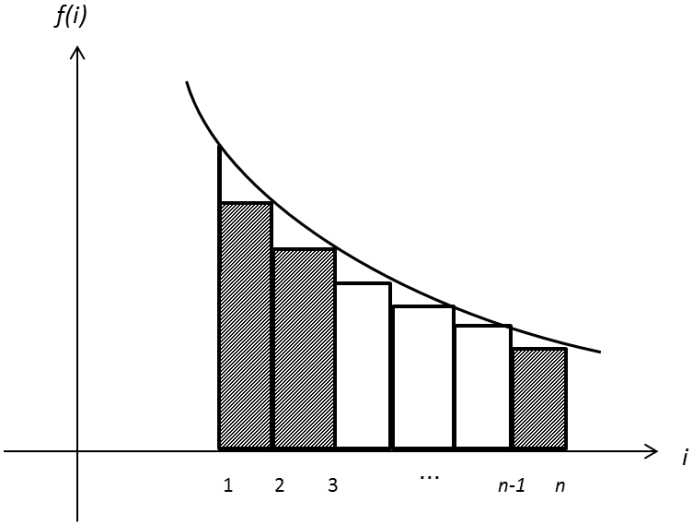
وبالتالى نحصل علي

$$F(n) = 2\sum_{i=2}^{n}\frac{1}{i} - 2\frac{n-1}{n+1}$$

ولكننا نعلم ان

$$\sum_{i=2}^{n}\frac{1}{i} \le \int_{1}^{n}\frac{1}{x}\,dx = \ln x]_1^n = \ln n$$

نظراً لان الدالة $f(i) = \frac{1}{i}$ تناقصية (decreasing).



$$\Rightarrow F(n) \cong 2\ln n$$

$$\Rightarrow A(n) \cong 2n\ \ln n \cong 1.4\ n\ \log_2 n = 1.4n\ \lg n$$

الحيِّز المستخدم Space usage

قـد يبـدو مـن النظـرة الأولى أن خوارزميـة الترتيـب السـريع ترتيـب في مَحَلِّـهِ / في موضـعه (an in–place sort). ولكنهـا ليسـت كـذلك. فبينمـا تعمـل الخوارزميـة علـى نطـاق جزئـي / مـدى جزئـي واحـد (working on one subrange)، فـإن مؤشِّـرَي البدايـة والنهايـة (beginning and ending indexes) [وسـنطلق عليهمـا "حَـدَّي" (the borders)] لجميـع النطاقـات الجزئيـة الأخـرى (other subranges) التي يجب ترتيبها (yet to be sorted) يتم حفظهما (saved) على رَصَّة الإطار (on the frame stack)، ويعتمد حجم الرصة على عدد النطاقات الجزئية التي سينقسم إليها النطاق / المدى (range will be split). وهذا بالطبع يعتمد على n. وفي أسوأ حالة يقوم إجراء التجزئة Partition بتقسيم عنصر واحد في المرة الواحدة (splitting off one entry at a time)، وعمـق الارتـداد (depth of the recursion) يسـاوي n. وهكـذا فـإن قـدر الحيِّـز (amount of space) الـذي تسـتخدمه الرصـة في أسـوأ حالـة هـو في $\Theta(n)$. وفيمـا يلـي نَـدْرُس أحـد التعـديلات (modifications) الـتي يمكـن إجراؤهـا علـى الخوارزميـة بحيـث يمكننـا تقليـل أكـبر حجـم للرصـة (maximum stack size) بدرجة كبيرة.

تحسينات على الخوارزمية الأساسية للترتيب السريع
Improvements on the Basic Quicksort Algorithm

اختيار الوتد Choice of Pivot

رأينـا أن خوارزميـة الترتيـب السـريع تعمـل جيـدا إذا كـان مفتـاح الوتـد (pivot key) الـذي يسـتخدمه إجـراء التجزئـة Partition لتجزئـة قطعـةٍ / شـريحةٍ مـا (a segment) قريبـا مـن وسـط (middle) الشـريحة. [موضـعه (position) هـو القيمـة **splitPoint** الـتي يعيـدها Partition]. واختيـار E[first] على أنه العنصر الوتدي (pivot element) يجعل خوارزمية الترتيب

السريع Quicksort تتصرف بطريقة سيئة في حالات يُفترض/ يجب أن تكون عملية الترتيب (sorting) فيها سهلة [مثلا حين تكون المنظومة أصلا مُرَتَّبة (already sorted)]. وهناك استراتيجيات أخرى عدة لاختيار العنصر الوتدي: إحداها أن نختار عددا صحيحا عشوائيا q بين **first** و **last**، ونجعل pivot=E[q].key. واستراتيجية أخرى أن نجعل pivot هو المفتاح الأوسط (median key) للعناصر الثلاثة E[first], E[(first+last)/2], E[last]. [وفي أي من الحالتين / الاستراتيجيتين العنصر الموجود في E[first] سيتم تبديله مع (swapped with) العنصر الوتدي (pivot element) قبل البدء بخوارزمية التجزئة (the Partition algorithm)]. وأي من هاتين الاستراتيجيتين تتطلب بعض الجهد الإضافي لاختيار pivot، ولكنها في المقابل تقوم بتحسين متوسط زمن التشغيل / وقت التنفيذ (average running time) لبرنامج ترتيب سريع (a Quicksort program).

الترتيب الصغير Small Sort

خوارزمية الترتيب السريع Quicksort لا تعد جيدة بالنسبة للمجموعات الصغيرة (small sets) بصورة خاصة، نظرا لعبء استدعاءات الإجراءات (procedure calls). ولكن بالنسبة لقيم n الكبيرة، فإن خوارزمية الترتيب السريع Quicksort بطبيعتها تُقَسِّم المجموعة / البِنيَة (break the set up) الأصلية إلى مجموعات جزئية صغيرة، وتُرَتِّبُها ارتداديًا (sort them recursively). وهكذا حينما يكون حجم أي مجموعة جزئية صغيرا، فإن الخوارزمية تصبح كفاءتها منخفضة. ويمكننا التغلب على هذه المشكلة باختيار قيمة صغيرة **smallSize**، وترتيب المجموعات الجزئية التي حجم أي منها أقل من أو يساوي **smallSize** بخوارزمية ترتيب بسيطة غير ارتدادية (simple nonrecursive sort) يطلَق عليها **smallSort** في الخوارزمية المُعَدَّلة (modified algorithm). [خوارزمية الترتيب بالإدخال (Insertion Sort) تُعَدُّ اختيارًا جيدًا].

```
void quicksort ( E, first, last )
    if (last – first > smallSize)
        pivotElement = E[first];
        pivot = pivotElement.key;
        int splitPoint = partition (E, pivot, first, last);
        E[splitPoint] = pivotElement;
        quicksort (E, first, splitPoint – 1);
        quicksort (E, splitPoint + 1, last);
    else
        smallSort (E, first, last);
```

وكتعديل على هذه الفكرة نتخطى (skip) خطوة استدعاء **smallSort**. وبعد ذلك حين تخرج (exits) الخوارزمية Quicksort تكون المنظومة غير مرتبة، ولكن لا يحتاج أى عنصر أن يتحرك أكثر من عدد smallSize من المواضع ليصل إلى موضعه المُرَتَّب الصحيح (correct sorted position). ولذلك فإن تنفيذا واحدا بعد التشغيل (one postprocessing run) لخوارزمية الترتيب بالإدخال سيكون ذا كفاءة عالية (very efficient)، وسيقوم بإجراء المقارنات نفسها تقريبا (about the same comparisons) مثل جميع الاستدعاءات له في دوره (in its role) كخوارزمية smallSort.

وأفضل اختيار لقيمة smallSize يعتمد على التنفيذ الخاص للخوارزمية (أى الحاسوب المستخدَم وتفاصيل البرنامج)، لأننا نقوم بإجراء بعض المقايضات بين الإضافيات ومقارنات المفاتيح. وقد تكون أى قيمةٍ قريبةٍ من 10 قيمةً جيدةً ومعقولة.

تحسين حيِّز الرَّصَّة Stack Space Optimization

لاحظنا أن عمق الارتداد (depth of recursion) لخوارزمية

Quicksort يمكن أن ينمو بدرجة كبيرة متناسبا مع n (proportional to) **في أسوأ حالة** (عندما يفصل Partition **عنصرا واحدا فقط** في كل مرة). وكثير من عمليات الدَّفع (pushing) والرَّفْع (popping) التي ستتم في رَصَّة الإطار (frame stack) غير ضرورية. وبعد التجزئة Partition يبدأ البرنامج بترتيب المدى الجزئي E[first], …, E[splitPoint–1] (sorting the subrange)، وبعد ذلك يجب أن يقوم بترتيب المدى الجزئي E[splitPoint+1], …, E[last].

الاستدعاء الارتدادي الثاني هو آخر عبارة في الإجراء، وبالتالي يمكن تحويله إلى تكرار (iteration) بالطريقة التي رأيناها سابقا مع **shiftVac** في خوارزمية الترتيب بالإدخال (Insertion Sort). ويبقى الاستدعاء الارتدادي الأول كما هو، وبالتالي نكون قد حذفنا الارتداد جزئيا فقط.

ومع بقاء استدعاء ارتدادي واحد فقط في الإجراء، فإننا لازلنا في حاجة إلى الاهتمام بعمق الارتداد الزائد (excessive depth of recursion). وهذا يمكن أن يحدث من خلال تعاقب استدعاءات ارتدادية (succession of recursive calls) يعمل كل منها على مدى جزئي أصغر قليلا فقط من المدى الجزئي السابق له. وبالتالي فإن الحيلة الثانية التي نستخدمها هي أن نتجنب إجراء الاستدعاء الارتدادي للمدى الجزئي الأكبر.

وإذا ضَمنَّا (by ensuring) أن يعمل كل استدعاء ارتدادي على عناصر عددها يساوي على الأكثر نصف عدد عناصر الاستدعاء السابق [الاستدعاء "الوالد" (its "parent" call)] فإننا سنضمن (guarantee) أن يبقى عمق الارتداد (depth of recursion) في حدود lg n تقريبا. وقد جَمَعنا (combined) بين هاتين الفكرتين في الإجراء التالي الذي تشير الرموز "TRO" في اسمه إلى "تحسين الارتداد الذَّيْلى" (tail recursion optimization). والفكرة هي أنه بعد كل تجزئة يعمل الاستدعاء الارتدادي التالي على المدى الجزئي الأصغر، بينما يتم التعامل (handling) مع المدى الجزئي الأكبر مباشرةً في عروة **while**.

```
void quickSortTRO ( E, first, last )
    int first1, lastl, first2, last2;
    first2 = first; last2 = last;
    while (last2 - first2 > 1)
      pivotElement = E[first];
      pivot = pivotElement.key;
      int splitPoint = partition (E, pivot, first2, last2);
      E[splitPoint] = pivotElement;
      if (splitPoint < (first2 + last2)/2)
        firstl = first2; lastl = splitPoint - 1;
        first2 = splitPoint + 1; last2 = last2;
      else
        firstl = splitPoint + 1; lastl = last2;
        first2 = first2; last2 = splitPoint - 1;
      quickSortTRO (E, firstl, lastl);
      // Continue the loop for first2, last2.
```

تحسينات مُجَمَّعَة Combined Improvements

ناقشنا التحسينات / التعديلات السابقة مستقلة عن بعضها البعض، ولكنها قابلة للجمع بينها في برنامج واحد.

ملاحظات Remarks

من الناحية العملية يتم تنفيذ برامج الترتيب السريع بسرعة كبيرة في المتوسط لقيم n الكبيرة، وتُستخدم على نطاق واسع. إلا أنه في أسوأ حالة يُعَدُّ

سلوك خوارزمية الترتيب السريع سيئاً. فزمن خوارزمية الترتيب السريع في أسوأ حالة ينتمى إلى $\Theta(n^2)$، مثل خوارزميات الترتيب بالإدخال Insertion Sort – التي درسناها سابقًا – والترتيب بالأكبر MaxSort، والترتيب الفقاعي Bubble Sort. ولكن السلوك المتوسط لخوارزمية الترتيب السريع –خلافاً لهذه الخوارزميات الأخرى- ينتمى إلى $\Theta(n \log n)$. فهل هناك خوارزميات ترتيب ينتمى زمن تنفيذها في أسوأ حالة إلى $\Theta(n \log n)$؟ أو هل يمكننا الوصول إلى حد أدنى في أسوأ حالة (a worst case lower bound) ينتمي إلى $\Theta(n^2)$؟ وقد منحتنا طريقة "قَسِّم وتَغَلَّب" تحسينا في السلوك المتوسط. وسنقوم الآن بإذن الله بدراسة هذه الطريقة العامة مرة أخرى ونرى كيفية استخدامها لتحسين السلوك في أسوأ حالة.

دمج المتتابعات المُرَتَّبة Merging Sorted Sequences

نفرض أن لدينا متتابعتين A, B مرتبتين ترتيبا غير تناقصي (sorted in non decreasing order)، والمطلوب دمجهما معا لإنشاء متتابعة مرتبة C. ودمج المتتابعات الجزئية المرتبة (sorted subsequences) يُعَدُّ أساسيًّا (essential) بالنسبة لاستراتيجية خوارزمية الترتيب بالدمج (Mergesort)، كما أن له تطبيقات عديدة. ومقياس الجهد المبذول بخوارزمية الدمج سيكون هو عدد مقارنات المفاتيح التي تجريها الخوارزمية.

نفرض أن k, m هما عدد العناصر في المتتابعتين A, B على الترتيب. ونفرض أن $n = k + m$ هى "حجم المسألة" (problem size). وبفرض أن كلا من المتتابعتين A, B غير خاوية، يمكننا أن نحدد فورا العنصر الأول في المتتابعة C: فهو العنصر الأصغر بين العنصر الأول في A والعنصر الأول في B. فماذا عن بقية عناصر C؟ إذا فرضنا أن العنصر الأول في A كان هو الأصغر، فإن بقية عناصر C ستكون هي نتيجة دمج جميع عناصر A بعد العنصر الأول مع جميع عناصر B. ولكن هذا هو مجرد صيغة أصغر (a smaller version) للمسألة نفسها (same problem) التي بدأنا بها. والوضع متماثل لو كان العنصر الأول في B هو الأصغر. وفى أي من الحالتين فإن حجم المسألة المتبقية (إيجاد بقية عناصر C) هو $n - 1$.

وفيما يلي شبه كود (pseudocode) الحل الارتدادي العام لعملية دمج المتتابعتين A, B في متتابعة واحدة C:

```
void merge ( A, B, C )
    if (A is empty)
        rest of C = rest of B
    else if (B is empty)
        rest of C = rest of A
    else if (first of A ≤ first of B)
        first of C = first of A
        merge (rest of A, B, rest of C)
    else
        first of C = first of B
        merge (A, rest of B, rest of C)
```

والآن نرى كيفية صياغة حل تكرارى (iterative) لعملية الدمج. ويمكن تطبيق فكرة الحل على جميع هياكل البيانات التتابعية (all sequential data structures)، ولكننا سنذكر الخوارزمية بدلالة المنظومات للتحديد (definiteness). وسنُعَرِّف ثلاثة مؤشرات (indexes) لاقتفاء / لتَتَبُّع (to keep track of) مواضع بدايات "بقية عناصر A" (rest of A) و"بقية عناصر B"، و"بقية عناصر C" عند أى مرحلة (stage) من مراحل التكرار. [وهذه المؤشرات ستكون "وسطاء" (parameters) في الصيغة الارتدادية].

الخوارزمية 3–4 الدمج Merge

المدخلات (Input): منظومة **A** من **k** عنصر، ومنظومة **B** من **m** عنصر، وكل من المنظومتين مرتبة ترتيبا غير تناقصي (nondecreasing order) بالنسبة لمفاتيحها.

المخرجات (**Output**): منظومة **C** تحتوى على n = k + m عنصر من المنظومتين **A, B** مرتبة ترتيبا غير تناقصي، حيث تُمَرَّر **C** (is passed in)، وتقوم الخوارزمية بملئها (filling in).

```
void merge (Element []A, int k, Element []B, int m,
      Element [] C, int & n)
  int indexA = 0, indexB = 0, indexC = 0;
  // indexA is the beginning of the rest of A; same for B, C.
  n = k + m;
  while (indexA < k && indexB < m)
    if (A[indexA].key ≤ B[indexB].key)
      C[indexC] = A[indexA];
      indexA + +;
      indexC + +;
    else
      C[indexC] = B[indexB];
      indexB + +;
      indexC + +;
    // Continue loop
  if (indexA ≥ k)
    Copy B[indexB, ..., m-1] to C[indexC, ..., n-1];
  else
    Copy A[indexA, ..., k-1] to C[indexC, ..., n-1];
```

أسوأ حالة Worst Case

كلما تمت مقارنة بين مفاتيح المنظومتين **A, B**، فإن عنصراً واحداً على الأقل يتم تحريكه إلى المنظومة **C**، ولا يُختبر بعد ذلك مرة أخرى أبدا (never examined again). وبعد آخر مقارنة يكون هناك على الأقل عنصران لم يتم تحريكهما بعد إلى المنظومة **C**، وهما العنصران اللذان تمت للتَّوِّ المقارنة بينهما في آخر مقارنة هذه. وسيتم تحريك أصغر العنصرين فورا، ولكن المنظومة **C** ستحتوى الآن على الأكثر على n-1 عنصرا، وسوف لا تكون هناك أي مقارنات أخرى. والعناصر المتبقية في المنظومة الأخرى سيتم تحريكها إلى المنظومة **C** بدون أى مقارنات أخرى. وهكذا فإن عدد المقارنات التي تم إجراؤها هي على الأكثر n-1 مقارنة. وأسوأ حالة [مستخدمين جميع المقارنات التي عددها n-1] تحدث عندما يكون العنصران [A[k-1], B[m-1] في آخر موضعين (belong in the last two positions) في المنظومة **C**.

أمثلية الدمج Optimality of Merge

نثبت فيما يلى أن خوارزمية الدمج 3–4 هى خوارزمية مُثْلى في أسوأ حالة بين الخوارزميات القائمة على أساس المقارنات (comparison-based algorithms) عندما تكون $k = m = n/2$. أى أنه لأى خوارزمية مبنية على أساس المقارنات تقوم بالدمج بطريقة صحيحة (merges correctly) لجميع المدخلات (for all inputs) التي تتحقق فيها العلاقة $k = m = n/2$ فإنه يجب أن يوجد مدخل ما (some input) تتطلب له الخوارزمية مقارنات عددها *n-1* مقارنة (for which it requires *n-1* comparisons) [وهذا لا يعنى القول بأنه لمُدْخَلٍ معين (for a particular input) لا توجد خوارزمية أفضل من خوارزمية 3–4]. وبعد أَخْذِنا في الاعتبار العلاقة $k = m = n/2$ ننظر إلى بعض العلاقات الأخرى بين ***k, m***.

نظرية 3–3: أى خوارزمية تقوم بدمج منظومتين مرتبتين تحتوى كل منهما على $k = m = n/2$ عنصر عن طريق المقارنة بين العناصر (المفاتيح)، تقوم على الأقل (at least) بإجراء عدد $n - 1$ من هذه المقارنات في أسوأ حالة.

البرهـــان: نفـــرض أننـــا قـــد أُعطينـــا خوارزميـــة دمـــج اختياريـــة (an arbitrary merge algorithm). ونفـرض أن a_i , b_i هما عنصرا المنظومتين **A, B** – على الترتيب – في الموضع i.

سـنثبت أنـه يمكـن اختيـار المفـاتيح (choosing keys) بحيـث أن الخوارزمية يجب أن تقارن بين a_i و b_i للقيم $0 \le i < m$، وبين a_i و b_{i+1} للقيم $0 \le i < m-1$. وبصورة محددة فإننا نختار المفاتيح بحيث أنه عندما تقارن الخوارزمية بين a_i و b_j: إذا كانت $i<j$، فإن النتيجة هى أن $a_i < b_j$، بينما إذا كانت $i \ge j$، فإن النتيجة هي أن $b_j < a_i$. واختيار المفاتيح بحيث أن

$$b_0 < a_0 < b_1 < a_1 < \ldots < b_i < a_i < b_{i+1} < \ldots < b_{m-1} < a_{m-1} \quad (2)$$

سيحقق هذه الشروط. إلا انه إذا حدث أنه لقيمةٍ ما i أن الخوارزمية لم تقارن أبدا بين a_i و b_i، فإن اختيار المفاتيح بالترتيب نفسه الذي يظهر في العلاقات (2)، بإستثناء $a_i < b_i$ سيحقق أيضا هذه الشروط، ولن تستطيع الخوارزمية تحديد الترتيب الصحيح (correct ordering).

وبالمثل إذا حدث أنه لقيمةٍ ما i أن الخوارزمية لم تقارن أبدا بين a_i و b_{i+1}، فإن الترتيبة (arrangement) المذكورة في العلاقات (2) باستثناء $a_i < b_{i+1}$ ستكون متوافقة (consistent) مع نتائج المقارنات التي أجريت، ومرة أخرى فإن الخوارزمية لا تستطيع تحديد الترتيب الصحيح.

والآن هـــل يمكننـــا تعمـــيم هـــذا الاســـتنتاج؟ نفـــرض أن قيمتـــى **k, m** تختلفـان عـن بعضيهما اختلافا بسيطا (كما سنرى بإذن الله احتمال حدوث هذا عند الترتيب بالدمج Mergesort).

نتيجة corollary

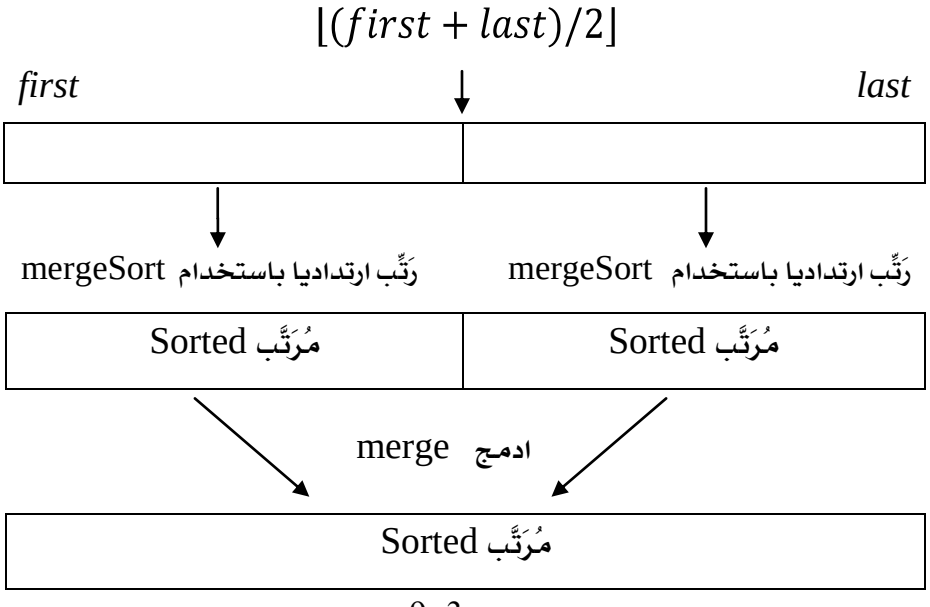
عنـد دمـج منظـومتين مـرتبتين (two sorted arrays) عـن طريـق المقارنة بين المفاتيح، بحيث أن المدخلات تحتوى على عناصر عددها k, m على الترتيب، حيث قيمتا k, m تختلفان عن بعضيهما بواحد، وحيث n = k + m، فإن أى خوارزمية تقوم بعملية الدمج هذه ستُجرى على الأقل n-1 (at least) من هذه المقارنات في أسوأ حالة.

البرهان:

هو نفسه برهان نظرية 3–3 باستثناء أنه لا يوجد a_{m-1}.

الترتيب بالدمج Mergesort

المشـكلة الـتي واجهتنـا عنـد تطبيـق خوارزميـة الترتيـب السـريع (Quicksort) هـى أن إجـراء التجزئـة (Partition) لا يقـوم دائمـا بتقسـيم المنظومة إلى جزئين متساويين في المدى (two equal subranges). خوارزمية الترتيب بالـدمج تقـوم بتقسـيم المنظومـة إلى نصـفين (two halves)، وتُرتِّـب النصفين منفصلين (separately) [وبالطبع ارتداديا (recursively)]، أى تقوم بترتيب كل نصف (ارتداديا) على حدة. ثم تقوم بدمج (merging) النصفين المُرتَّبين (the sorted halves)، كما هو موضح في شكل 3–9.



شكل 3–9

استراتيجية الترتيب بالدمج Mergesort strategy

وهكذا باستخدام أسلوب "قَسِّم وتَغَلَّب" (divide-and-conquer) فإن "قَسِّم" هنا تفيد مجرد حساب مؤشر وسط / منتصف المدى الجزئي (middle index of the subrange) ولا تقوم بإجراء أى مقارنة مفاتيح (key comparisons) و "جَمِّع" (combine) تعنى إجراء عملية الدمج (merging).

ونفرض هنا أن إجراء الدمج (Merge) سيُعَدَّل (modified) ليقوم بدمج المجوعات الجزئية (مجموعات المدى الجزئي) المتجاورة (to merge adjacent subranges) ضمن منظومة واحدة، واضعا (putting) المنظومة المُدْمَجَة الناتجة (resulting merged array) مرة أخرى في الخلايا (back into the cells) التي كانت تشْغَلُها

أصلاً (originally occupied) العناصر التي تم دمجها. والوسطاء (parameters) الآن هى: اسم المنظومة E، والمؤشرات (indexes): first, mid, last الخاصة بمجموعات المدى الجزئي (subranges) التي سيقوم بدمجها، أى أن مجموعات المدى الجزئي المرتَّبة (sorted subranges) هي: E[first], ..., E[mid] و E[mid+1], …, E[last]. والمدى المُرَتَّب النهائي (final sorted range) سيكون E[first], ..., E[last]. وفي هذا التعديل (modification) سيكون الإجراء merge مسئولاً أيضاً عن تعيين مواضع (allocating) لِحَيِّز العمل الإضافي الذي نحتاج إليه (additional workspace needed).

أخوارزميث 3–5 الترتيب بالدمج Mergesort

المدخلات (Input): منظومة E، ومؤشران first, last (indexes)، بحيث أن عناصر E[i] مُعَرَّفة للقيم (first ≤ i ≤ last).

المخرجات (Output): E[first], ..., E[last] العناصر نفسها مرتَّبَة (a sorted rearrangement of the same elements).

```
void mergeSort ( Element [ ] E, int first, int last )
    if (first < last)
        int mid = (first+last)/2;
        mergeSort (E, first, mid);
        mergeSort (E, mid + 1, last):
        merge (E, first, mid, last);
```

لاحظ أن هناك فارقا بين خوارزمية الدمج merge وخوارزمية الترتيب

بالدمج mergeSort، ولا تخلط بينهما، حيث قد يلتبس الأمر على البعض. فتَـذَكَّر أن خوارزميـة الترتيـب بالـدمج mergeSort هـى خوارزميـة ترتيـب (a sorting algorithm). فهى تبدأ بمنظومة واحدة عناصرها مبعثرة، وتقـوم بترتيبها. أمـا خوارزميـة الـدمج merge فهـى تبـدأ بمنظـومتين مـرتبتين أصلاً (already sorted)، وتقوم بدمجهما / بالجمع بينهما (combining them) في منظومة واحدة مرتبة (one sorted array).

ولزيـادة إيضـاح هـذا الفـارق نعطـي فيمـا يلـي شـبه كـود (الكـود الزائـف pseudo-code) كل من هاتين الخوارزميتين.

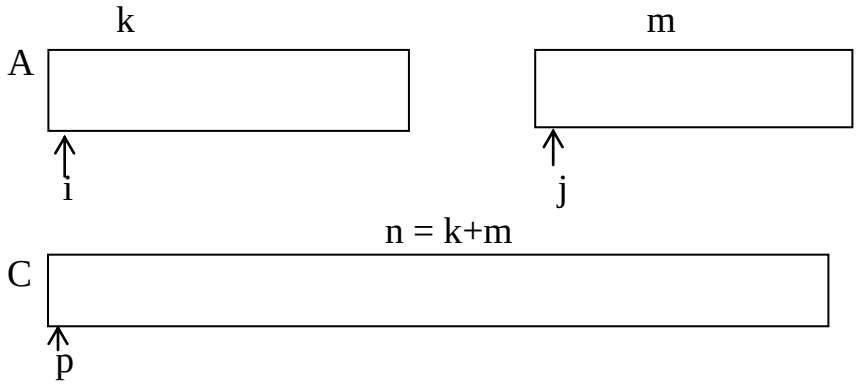
شبه كود خوارزمية دمج قائمتين / منظومتين مرتبتين

Pseudocode of merging two sorted lists / arrays

شبه كود خوارزمية الدمج Pseudocode of the algorithm merge

المـدخلات (input): منظومـة A مـن k عنصـر، ومنظومـة B مـن m عنصر، وكل من المنظومتين مرتبة ترتيبا غير تناقصي بالنسبة لمفاتيحها.

المخرجـات (output): منظومـة C تحتـوي علـى n=k+m عنصـر مـن المنظومتين A, B مرتبة ترتيبا غير تناقصي.



شكل 3–10

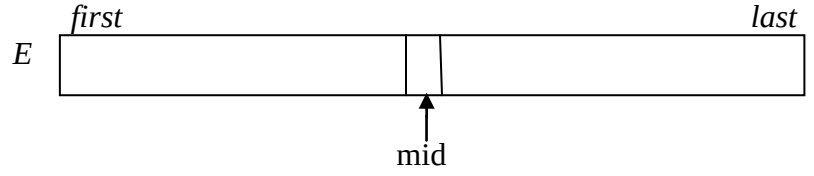
```
i = 1;  j = 1;  p = 1;
n = k + m;
while ( i ≤ k and j ≤ m )
    if ( Ai < Bj )
        Cp = Ai
        i = i + 1
    else
        Cp = Bj
        j = j + 1
    p = p + 1
//  copy remaining elements of A or of B to C
if ( i > k )          //  copy rest of B to C
    for j = j to m
        Cp = Bj
        p = p + 1
else                  //  copy rest of A to  C
    for i = i to k
        Cp = Ai
        p = p + 1
```

شبه كود خوارزمية الترتيب بالدمج

Pseudocode of the mergesort algorithm

المدخلات (Input): منظومة E، ومؤشران first, last، بحيث أن عناصر E[i] مُعَرَّفة للقيم $first \le i \le last$.

المخرجات (Output): العناصر نفسها E[first], ..., E[last] مُرَتَّبة.



شكل 3–11

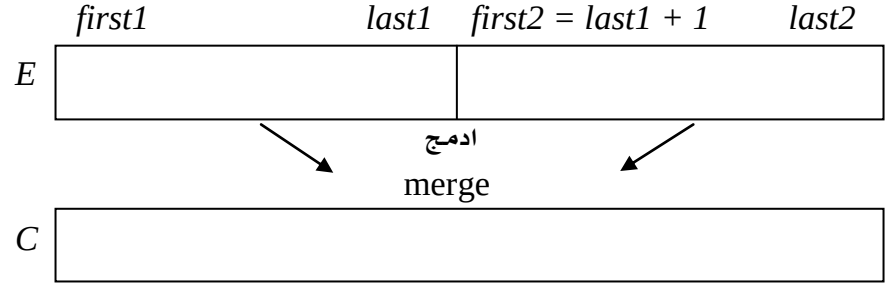
```
void mergesort ( E [ ], first, last )
  if ( first < last )
    mid = ⌊( first  + last )/2  ⌋ ;
    mergesort ( E, first, mid ) ;
    mergesort ( E, mid + 1, last ) ;
    merge ( E, first, mid , mid + 1, last ) ;
                 1st          2nd
```

وأما عن كيفية استدعاء خوارزمية الترتيب بالدمج: فإذا فرضنا أن المنظومة E تحتوي على n عنصر، ومطلوب ترتيب جميع عناصرها، فإننا نستدعي mergesort هكذا:

```
mergesort ( E, 1 , n)
```

ونفرض هنا أن الإجراء merge سيعمل كما يلي (انظر شكل 3–12):

```
void merge ( E [ ], first1, last1, first2, last2 )
C : array ;          //  local array
```



شكل 3–12

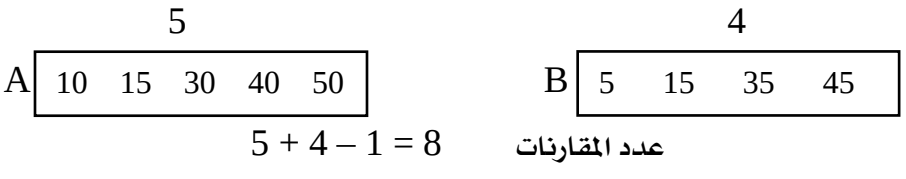
وعند انتهاء التنفيذ تُنسخ المنظومة C (وهي المنظومة الناتجة بعد الدمج) في المنظومة الأصلية E.

درجة تعقيد خوارزمية الدمج في أسوأ حالة (عدد المقارنات)

Worst Case Complexity of the merge algorithm

في أسوأ حالة سنقوم بإجراء عدد k + m - 1 من المقارنات، وذلك إذا تم تحريك جميع عناصر المنظومتين A, B إلى المنظومة C داخل عروة while، فيما عدا آخر عنصر في المنظومة C. وكل عنصر من العناصر التي تذهب إلى المنظومة C سوف لا يُقارن بعد ذلك مرة أخرى أبدا (never be compared again). أي أننا نحتاج لاجتياز العروة عدد k + m - 1 من المرات، وفي كل مرة نجري مقارنة واحدة.

مثال لأسوأ حالة A worst case example



شكل 3–13

مثال لحالة ليست أسوأ حالة not a worst case example

عدد المقارنات = 4

شكل 3–14

مثال لأحسن حالة A best case example

تتحقق لدينا أحسن حالة (أي أن عدد المقارنات يكون أقل ما يمكن) عندما تكون مثلا: $k<<m$ (أي ان عدد عناصر المنظومة A أصغر كثيرا من عدد عناصر المنظومة B)، وكذلك جميع عناصر المنظومة A أصغر من جميع عناصر المنظومة B:

$k << m$ & all A's < all B's.

درجة تعقيد خوارزمية الترتيب بالدمج في أسوأ حالة

Worst Case Complexity of the mergeSort algorithm

نفرض أن $n = 2^k$ حيث k عدد صحيح موجب. عدد العمليات فى أسوأ حالة W(n) يحقق العلاقتين:

$$W(n) = (n-1) + 2\,W(\frac{n}{2})$$

$$W(1) = 0$$

حيث $(n-1)$ هو عدد العمليات لدمج منظومتين (2 arrays) طول كل منهما $\frac{n}{2}$. للتبسيط نأخذ فى الاعتبار العلاقتين:

$$W(n) = n + 2W(\frac{n}{2})$$

$$W(1) = 0$$

وهذه علاقة ارتدادية / رجعية (recurrence relation) نقوم بحلها كما يلي:

$$\begin{aligned} W(n) &= n + 2W(\frac{n}{2}) \\ &= n + 2(\frac{n}{2} + 2W(\frac{n}{4})) \\ &= n + n + 4W(\frac{n}{4}) \\ &= 2n + 2^2 W(n/2^2) \\ &\vdots \\ &= in + 2^i W(n/2^i); \qquad i = 1, 2, \ldots, k \end{aligned}$$

وبوضع $i = k$ نحصل على

$$\begin{aligned} W(n) &= kn + 2^k W(n/2^k) \\ &= kn + n\,W(1) = kn + 0 = n \log n \end{aligned}$$

و هكذا نكون قد أثبتنا أن $W(n) \in \Theta(n \lg n)$، و كذلك تكون درجة التعقيد في المتوسط هي $A(n) \in \Theta(n \lg n)$.

الحد الأدنى لعدد المقارنات بين المفاتيح لأجل الترتيب

Lower Bound on Number of Comparisons of Keys for Sorting

رأينا أن عدد المقارنات بين المفاتيح في كل من الترتيب بالإدخال (Insertion Sort) والترتيب السريع (Quicksort) في أسوأ حالة هو في حدود $\Theta(n^2)$. واستطعنا تحسين هذه النتيجة بالترتيب بالدمج (Mergesort) حيث كان العدد في أسوأ حالة في حدود $\Theta(n \log n)$. فهل لا زال في إمكاننا الوصول إلى تحسين أفضل؟ وإلى أي مدى يمكن أن تصل سرعة عملية الترتيب؟ أي ما هو أقل عدد ممكن من المقارنات اللازمة لترتيب عناصر منظومة مكونة من n عنصر؟

فيما يلى سنثبت بإذن الله أن أى خوارزمية ترتيب تقوم بعملية الترتيب عن طريق المقارنة بين المفاتيح يجب أن تجرى على الأقل عدد $\Theta(n \lg n)$ من المقارنات على مدخلات عددها n عنصر، وذلك فى أسوأ حالة. وسنفرض أن جميع مفاتيح المنظومة المطلوب ترتيب عناصرها متمايزة (distinct).

أشجار القرار لخوارزميات الترتيب

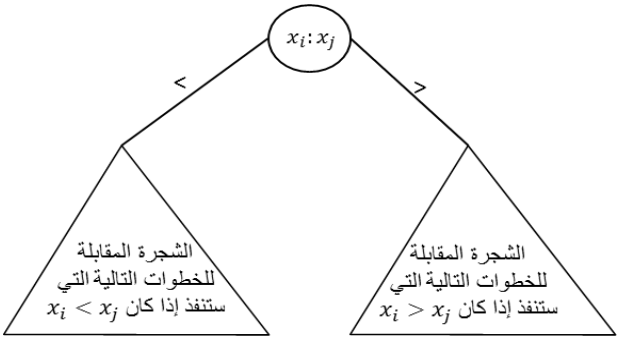
Decision Trees for Soring Algorithms

نفرض أن n عدد محدد (fixed)، وأن المفاتيح هى $x_1, x_2, \ldots, x_n$. سنعطى لكل خوارزمية وعدد صحيح موجب n شجرة قرار (ثنائية) تصف (describes) متتابعة المقارنات التى تجريها الخوارزمية على أى مدخلات حجمها n (any input of size). نفرض أن Sort هى أى خوارزمية تقوم بالترتيب عن طريق المقارنة بين المفاتيح. أى مقارنة ستتفرع الى طريقين (has a two-way branch) نظراً لأن المفاتيح متمايزة (distinct)، ونفرض أن الخوارزمية Sort تحتوى على أمر طباعة / تعليمة إخراج (output instruction) لطباعة منظومة المفاتيح بعد إعادة ترتيبها. ويتم تعريف شجرة القرار للخوارزمية Sort عن طريق الاستقراء (inductively) بأن نلحق (associate) شجرة بكل مقارنة وكل أمر طباعة كما يلي:

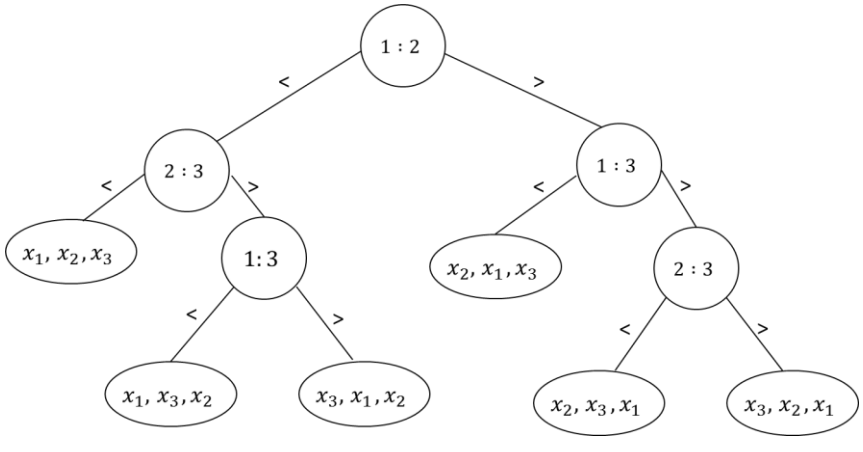
الشجرة المرتبطة بأى أمر طباعة تتكون من عقدة (node) واحدة معنونة (labeled) بإعادة ترتيب المفاتيح (rearrangement of the keys). والشجرة المرتبطه بتعليمة تقارن بين المفتاحين x_i, x_j تتكون من جذر (root) معنون (i : j)، وشجرة فرعية يسرى (a left subtree) تمثل الشجرة المرتبطة بتعليمة (المقارنة أو الإخراج)[(comparison or output instruction)] التى ستنفذ في الخطوة التالية إذا كان $x_i < x_j$، وشجرة فرعية يمنى تمثل الشجرة المرتبطة بتعليمة (المقارنة أو الأخراج) التى ستنفذ فى الخطوة التالية إذا كان $x_i > x_j$. وشجرة القرار للخوارزمية Sort هى الشجرة المرتبطة بأول تعليمة مقارنة تقوم الخوارزمية بتنفيذها أى أن:

نموذج شجرة القرار Decision Tree Model

هـو عبـارة عـن شـجرة ثنائيـة موسـعة / ممتـدة (extended binary tree) عناصرها/ عقدها الداخلية (internal nodes) تمثل المقارنات $x_i : x_j$ ، وكل من أوراقها (leaves) تمثل أحد المخرجات (an output) [أو أحد التباديل (a permutation) الذى يقابل الترتيب المطلوب (required order)].



مثـال 3–3: الشـكل التـالى شـكل (3–15) يوضـح مثـالا لشـجرة قـرار للحالـة $n = 3$ ، **أى لترتيب ثلاثة مفاتيح** $x_1,\ x_2,\ x_3$.



شكل 3–15

شجرة قرار خوارزمية ترتيب ثلاثة عناصر

فى هذا المثال نلاحظ ما يلي:

- **الشجرة تشتمل على ست أوراق** (6 leaves) **تقابل المخرجات الستة المحتملة** (3! = 6 possible outcomes).
- **أى مسار** (a path) **فى الشجرة (من الجذر إلى ورقة) يقابل متتابعة** (sequence) **المقارنات التى تجريها الخوارزمية على المدخلات المعطاة** (given input). **وعدد هذه المقارنات هو عدد العناصر / العقد الداخلية** (number of internal nodes) **فى هذا المسار لهذه المدخلات.**
- **وبالتالى فإن عدد المقارنات التى تجريها هذه الخوارزمية فى أسوأ حالة هو عدد العقد الداخلية فى أطول مسار، وهو عمق / ارتفاع الشجرة** (of the (depth / height tree.

مثال 3–4: أوجد كلا من:

أ - عدد المقارنات فى أسوأ حالة

ب - متوسط عدد المقارنات

التى تجريها خوارزمية ترتيب ثلاثة عناصر، والتى يوضح شكل 3- 15 شجرة قرارها.

الحل

أ - عدد المقارنات فى أسوأ حالة = 3

ب - متوسط عدد المقارنات = متوسط أطوال جميع المسارات من الجذر الى ورقة

$$= \frac{3+3+2+3+3+2}{6} = \frac{16}{6} = 2\frac{2}{3}$$

الحد الأدنى لأسوأ حالة Lower Bound for the Worst case

نستنتج فيما يلى أن ارتفاع مثل هذه الشجرة هو حد أدنى لعدد المقارنات التى تجريها أى خوارزمية فى أسوأ حالة.

للحصول على حد أدنى لأسوأ حالة لخوارزميات الترتيب بالمقارنة نستنتج حداً أدنى لارتفاع (height) / عمق (depth) شجرة ثنائية بدلالة عدد الأوراق، وذلك لأن المعلومات الكمية (quantitative information) الوحيدة المتوفرة لدينا عن أشجار القرار هى عدد الأوراق. فشجرة القرار لأى خوارزمية يجب أن تحتوى على الأقل على عدد !n من الأوراق. فما هو ارتفاع / عمق مثل هذه الشجرة؟

تمهيدية 3–1: اذا كان ℓ هو عدد الاوراق في شجرة ثنائية، و h هو ارتفاعها، فإن

$$\ell \le 2^h$$

البرهان: بالاستقراء المباشر علي الارتفاع h.

الخطوة الأساسية

عندما $h = 0$ (شجرة ارتفاعها 0) فإن الشجرة تحتوى على ورقة واحدة $\ell = 1$ أى أن

$$2^h = 2^0 = 1$$

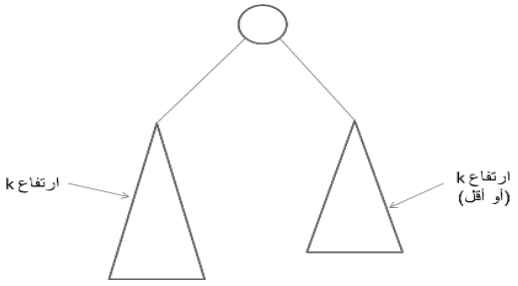
وبالتالى فإن المتباينة $\ell \le 2^h$ **متحققة**.

الخطوة الاستقرائية

نفرض ان المتباينة $\ell \le 2^h$ متحققة عندما $h = k$

أى أن جميع الأشجار الثنائية التى ارتفاعها k تحتوى على الأكثر علي 2^k ورقة.

عندما $h = k + 1$:



العدد الكلى للأوراق = عدد أوراق الشجرة الفرعية اليمنى + عدد أوراق الشجرة الفرعية اليسرى

$$\ell \le 2^k + 2^k = 2^{k+1}$$

تمهيدية 3–2: إذا كان ℓ هو عدد الاوراق فى شجرة ثنائية، و h هو ارتفاعها، **فإن**

$$h \ge \lceil \lg \ell \rceil$$

البرهان : بأخذ لوغاريتمى طرفى المتباينة فى التمهيدية 3–1 السابقة، نحصل على

$$lg\ \ell \le h$$

ونظراً لأن h عدد صحيح، فإن $h \ge \lceil \lg \ell \rceil$.

تمهيدية 3–3: نفرض أن n عدد صحيح موجب معطي. شجرة القرار لأى خوارزمية تقوم بالترتيب عن طريق المقارنة بين المفاتيح التى عددها n يكون ارتفاعها مساويا على الأقل $\lceil \lg n! \rceil$.

البرهان: ضع $\ell = n!$ فى متباينة التمهيدية 3–2.

وهكذا فإن عدد المقارنات المطلوبة للترتيب في أسوأ حالة هو على الأقل $\lceil \lg n! \rceil$. وأفضل ترتيب (best sort) رأيناه حتى الآن هو الترتيب بالدمج Mergesort، ولكن ما مدى قرب $\lceil \lg n! \rceil$ من $n \lg n$؟ للوصول الى إجابة على هذا السؤال نحتاج لوضع $\lg n!$ في صيغة أكثر ملاءمة، ثم نحصل على حد أدنى لقيمتها.

تمهيدية 3–4:

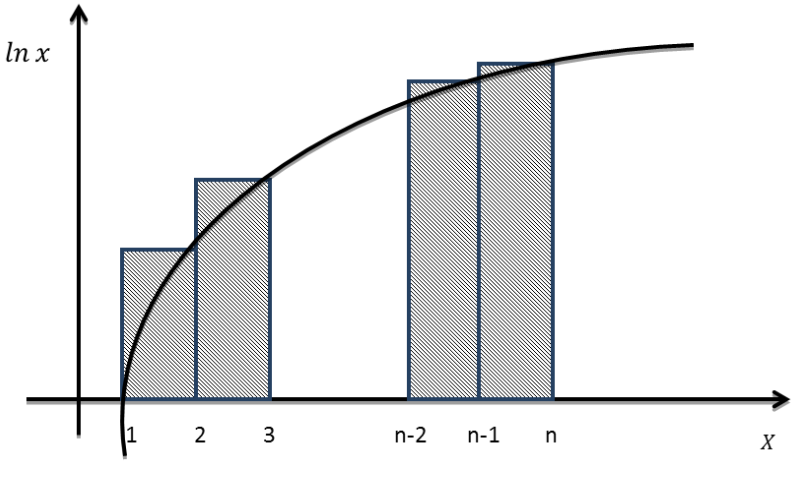
$n! \geq (n/e)^n$ (Stirling formula)

$[n! \geq (n/2)^{n/2}$ أو]

البرهان:

$$n! = n(n-1)(n-2)\ldots 3.2.1$$

$$\log n! = \log n + \log(n-1) + \cdots + \log 2 + \log 1 = \sum_{i=1}^{n} \log i = \sum_{i=2}^{n} \log i$$



شكل 3-16

$$\Rightarrow \log_e n! = \sum_{i=2}^{n} \log_e i$$

$$\Rightarrow ln\, n! = \sum_{i=2}^{n} ln\, i$$

ومن منحنى اللوغاريتم المبين فى شكل 3- 16 يتضح لنا أن

$$\sum_{i=1}^{n} \ln i \;\geq\; \int_{x=1}^{n} \ln x\, dx$$

(مجموع مساحات المستطيلات أكبر من أو يساوى المساحة أسفل المنحني)

$$ln\, n! \geq \int_{x=1}^{n} ln\, x \;\; dx = [x \;\; ln\, x \text{-} x]_1^n$$
$$= n \;\; ln\, n \text{-} n + 1 \geq n \;\; ln\, n \text{-} n = n(ln\, n - 1)$$
$$= n\, ln(n/e) = ln(n/e)^n$$
$$\Rightarrow \quad n! \;\geq\; (n/e)^n$$

وهو المطلوب إثباته.

وبناء عليه فإن حداً أدنى – فى أسوأ حالة – (a worst case lower bound) لعدد المقارنات يساوى أقل ارتفاع لشجرة قرار عدد أوراقها $\ell = n!$.

$$h \geq \lceil log_2 \ell \rceil = \lceil \lg \ell \rceil = \lceil \lg n! \rceil \;\geq\; \lceil \lg (n/e)^n \rceil$$
$$= \lceil n \lg(n/e) \rceil = \lceil n \lg n \text{-} n \lg e \rceil = \lceil n \lg n \text{-} 1.443n \rceil = \Theta(n \lg n)$$

وهكذا فإن ارتفاع شجرة القرار يساوى على الأقل $\lceil n \lg n \text{-} 1.443\, n \rceil$.

نظرية 3–4: أى خوارزمية تقوم بترتيب عناصر عددها n عن طريق المقارنات بين المفاتيح يجب أن تقوم على الأقل بإجراء عدد $\lceil \lg n! \rceil$ – أو تقريبًا $\lceil n \lg n - 1.443\, n \rceil$ – من المقارنات فى أسوأ حالة.

وهكذا فإن الترتيب بالدمج قريب جدا من الحالة المثلى (optimal).

الحد الادنى للسلوك المتوسط

Lower Bound for Average Behavior

يمكننا كذلك إثبات أن $\Theta(n \lg n)$ هو أيضا حد أدنى لمتوسط عدد المقارنات اللازمة لترتيب عناصر عددها n .

نظرية 3–5: متوسط عدد المقارنات التى تجريها أى خوارزمية لترتيب عناصر عددها n عن طريق المقارنة بين المفاتيح يساوى على الأقل $\lg(n!)$ ، والذى يساوى تقريبا $n \lg n - 1.443\, n$.

الفارق الوحيد بين هذا الحد الأدنى والحد الأدنى فى أسوأ حالة هو أنه لا يوجد هنا تقريب لأعلى لعدد صحيح (rounding up to an integer)، وذلك لأن المتوسط (average) لا يلزم أن يكون عددا صحيحاً، بينما أسوأ حالة تتطلب ذلك.

الترتيب بالكومة

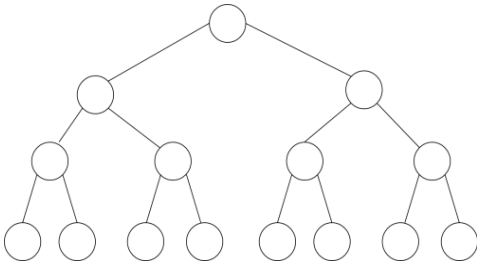
Heapsort

تقـــوم خوارزميـــة الترتيـــب الســـريع Quicksort بإعـادة ترتيـب (rearranging) العناصر فى المنظومة الأصلية (original array)، ولكن لا يمكنها أن تكون متأكدة من إجراء تقسيم جزئى متساوى للمسألة (an even subdivision of the problem). وأما خوارزمية الترتيب بالدمج mergesort فيمكنها ضمان التقسيم الجزئى المتساوى وتُعد تقريبا مُثلى بالنسبة لأسوأ حالة (a nearly optimal worst case)، إلا أنها لا تستطيع إعادة ترتيب العناصر فى المنظومة الأصلية، بل تحتاج إلى حيز إضافى مساعد كبير للعمل (sizeable auxiliary workspace). وخوارزمية الترتيب بالكومة Heapsort التى سندرسها الآن بإذن الله تعيد ترتيب العناصر فى المنظومة الأصلية، وأسوأ حالة لها فى حدود $\Theta(n \log n)$ والتى تعد مُثلى من حيث معدل النمو (in terms of growth rate)، وبذلك فهى نوعا ما تجمع بين ميزتى (advantages) الترتيب السريع والترتيب بالدمج. وأما عيب خوارزمية الترتيب بالكومة فهو المعامل الثابت الأكبر (higher constant factor) من معاملى الخوارزميتين الأخريتين. إلا أن هناك صيغة أحدث لخوارزمية الترتيب بالكومة تقلل هذا المعامل الثابت إلى مستوى تنافسى مع خوارزميتي الترتيب السريع والترتيب بالدمج. وهذه الصيغة الأحدث يطلق عليها "الترتيب المُعَجَّل بالكومة" (Accelerated Heapsort). وهكذا فإن خوارزمية الترتيب المعجل بالكومة قد تصبح أفضل اختيار لإجراء عملية الترتيب.

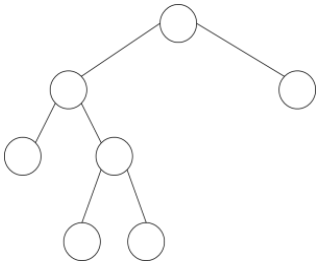
الكومة Heap

تستخدم خوارزمية الترتيب بالكومة هيكل بيانات / بنية معطيات (a data structure) يطلق عليها "كَوْمَة" (a heap)، وهي عبارة عن شجرة

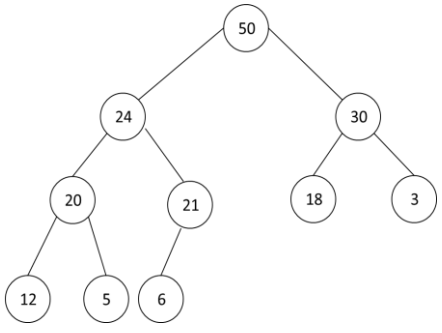
ثنائية لها خواص معينة (special properties). وتعريف الكومة يتضمن وصفاً للبنية وشرطاً علي البيانات الموجودة في العقد (data in the nodes) يطلق عليه "خاصية شجرة الترتيب الجزئي" (partial order tree property) أو "خاصية الكومة" (heap property). ونقول – بصفة غير رسمية (informally) – إن "بنية / هيكل كومة" (a heap structure) عبارة عن شجرة ثنائية كاملة (a complete binary tree) ربما قد أزيلت بعض أوراقها الموجودة أقصي اليمين (some rightmost leaves removed) [انظر شكل 3–17 للتوضيح].



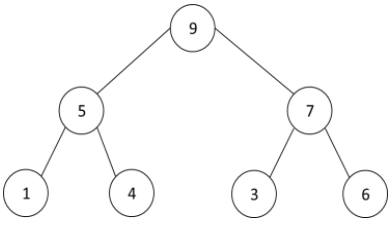
شجرة ثنائية كاملة



شجرة 2–



كومة 2



كومة 1

شكل 3–17

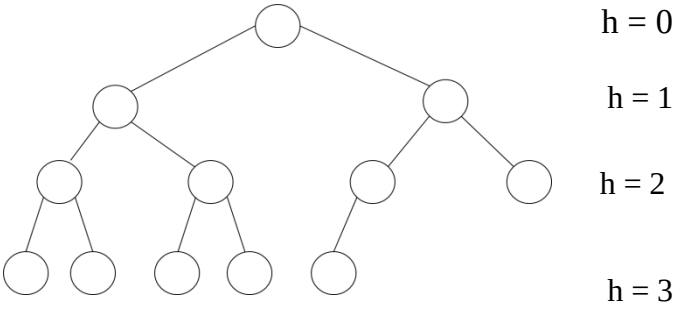
وسنستخدم الرمز S للدلالة علي مجموعة عناصر (elements) مفاتيحها (with keys) ذوات ترتيب خطي (a linear ordering)، والرمز T للدلالة علي شجرة ثنائية ارتفاعها h (height) وعقدها (nodes) تحتوي علي عناصر المجموعة S.

تعريف 3–1 : بنية / هيكل كومة Heap structure

يقال إن شجرة ثنائية T (a binary tree) هي بنية / هيكل كومة (a heap structure) إذا وفقط إذا (if and only if) حققت الشروط التالية:

1) T شجرة كاملة (complete) علي الأقل حتي الارتفاع / العمق (at least h-1 through depth).
2) جميع أوراق (leaves) الشجرة موجودة عند العمق h أو h-1.
3) جميع المسارات (paths) إلي ورقة عمقها h (of depth) تكون علي يسار (left) جميع المسارات إلي ورقة عمقها h-1.

والعقدة الداخلية أقصي اليمين (rightmost internal node) عند العمق h-1 (at depth) في بنية كومة قد يكون لها ابن أيسر (a left child) دون ابن أيمن (no right child) [و العكس غير صحيح]. وجميع العقد الداخلية الأخرى لها ابنان (two children.) وهناك اسم آخر لبنية / هيكل الكومة وهو "شجرة ثنائية كاملة من جهة اليسار" (left-complete binary tree).



شكل 3–18 بنية / هيكل كومة

والكومة (heap) هي هيكل كومة (a heap structure) وشرط علي البيانات الموجودة في العقد (nodes):

فيقال إن شجرة ثنائية T هي كومة (a heap) إذا و فقط إذا حققت الشرطين:

i. جميع العقد الداخلية (internal nodes) [ربما عدا واحدة هي العقدة الموجودة أقصي اليمين في المستوي *h-1*] درجتها 2 (have degree)، والأوراق (leaves) الموجودة في المستوي *h-1* تقع جميعها يمين العقد الداخلية، حيث h هي ارتفاع / عمق الشجرة.

ii. المفتاح (key) [أي المعلومات المخزونة (stored information)] عند أي عقدة (node) يكون أكبر من أو يساوي (≤) مفتاح / مفاتيح [key(s)] أبناء (children) هذه العقدة [إن كان لها أبناء (if any)].

وأحياناً يطلق علي هذه الكومة الاسم "كومة تكبيرية" "a maximizing heap" [أو اختصارا a max-heap] بسبب الشرط (ii).

ويوضح الشكل 3–17 مثالين للكومة (التكبيرية): كومة 1، كومة 2.

تعريف 3–2: خاصية شجرة الترتيب الجزئي Partial order tree property

يقـال إن شـجرة T هـي **شـجرة ترتيـب جزئـي (تكبيريـة)** [a (maximizing) partial order tree] إذا و فقط إذا (iff) كان المفتاح (key) عند أى عقدة (node) أكبر من أو يساوي المفاتيح عند كل من أبنائها (children) [إن كان لها أبناء (if any)].

وأحياناً يطلق علي هذه الخاصية "خاصية الكومة التكبيرية"-**(max-heap property)** أو اختصاراً "خاصية الكومة" (**heap property**)، ويطلق علي الشجرة التي تحققها "**كومة تكبيرية**" (**a max-heap**)، أو اختصاراً "**كومة**" (**a heap**) حيث يكون أكبر عنصر عند الجذر (largest element at the root):

لجميع العقد i باستثناء الجذر: $E[PARENT(i)] \geq E[i]$

(for all nodes i, excluding the root)

وفي المقابل فإن "**الكومة التصغيرية**" (**a min-heap**) – حيث يكون أصغر عنصر عند الجذر- تحقق "**خاصية الكومة التصغيرية**" (**min-heap property**):

لجميع العقد i باستثناء الجذر: $E[PARENT(i)] \leq E[i]$

(for all nodes i, excluding the root)

وبالنسبة للخوارزميات التي سنتناولها فسنستخدم دائماً الكومة التكبيرية (max-heap)، ونطلق عليها الاسم المختصر كومة (a heap).

ملاحظة: عموماً الكومة يمكن أن تكون شجرة متعددة الأفرع (k-ary tree) بدلاً من شجرة ثنائية (binary tree).

لاحظ أن أى شجرة ثنائية كاملة (complete) هي بنية كومة. وعند إضافة عقد جديدة إلي كومة فيجب أن تضاف من اليسار إلي اليمين عند المستوي السفلي (bottom level)، وعند حذف عقدة فيجب أن تكون هي العقدة الموجودة أقصي اليمين (rightmost node) في المستوي السفلي إذا أردنا أن تظل البنية الناتجة (resulting structure) – بعد الإضافة أو الحذف – كومة (a heap). ولاحظ أن الجذر (root) يجب أن يحتوي علي أكبر مفتاح (largest key) في الكومة.

استراتيجيات الترتيب بالكومة Heapsort Strategy

نفرض أن العناصر المطلوب ترتيبها تصاعديا موجودة في كومة معطاة. يمكننا بإذن الله بناء متتابعة مرتبة (a sorted sequence) بالحصول علي عناصرها بترتيب عكسي (in reverse order) عن طريق تكرار إزالة (removing) العنصر الموجود في الجذر (أكبر مفتاح باقي)، وإعادة ترتيب العناصر التي لا تزال في الكومة كي تحقق خاصية شجرة الترتيب الجزئي، وبالتالي وضع المفتاح الأكبر التالي (next largest key) في الجذر. وسنطلق علي هذه العملية deleteMax.

وفيما يلي سنعرض بإذن الله أولاً المخطط العام (outline) لهذه الاستراتيجية، ثم نتبع ذلك بإعطاء التفاصيل. وكالمعتاد نفرض أن العناصر التي عددها n مخزونة في منظومة E، ولكن هذه المرة نفرض أن مدى المؤشرات (range of indexes) هو $1, \ldots, n$ لأسباب ستتضح عندما نقوم بتنفيذ (implementation) الكومة. والآن نفرض أن الكومة (والتي اسمها H) موجودة في مكان آخر.

```
void heapSort ( E, n )      // outline
{
    construct H from E, the set of n elements to be sorted.
    for ( i = n; i ≥ 1; i - - )
    {
        curMax = getMax ( H );
        deleteMax ( H );
        E [ i ] = curMax;
    }
}
```

شكل 3–19

المخطط العام (outline) لخوارزمية heapSort

الخطوات التنفيذية لخوارزمية heapSort

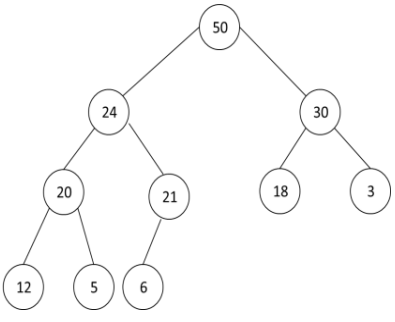
1 - قم بإنشاء كومة من منظومة الإدخال (input array) المعطاة.

2 - مبتدئاً بالجذر (أكبر عنصر) ضع أكبر عنصر في مكانه الصحيح (correct place) في المنظومة بتبديله (swapping it) مع العنصر الموجود في آخر موضع في المنظومة (last position in the array).

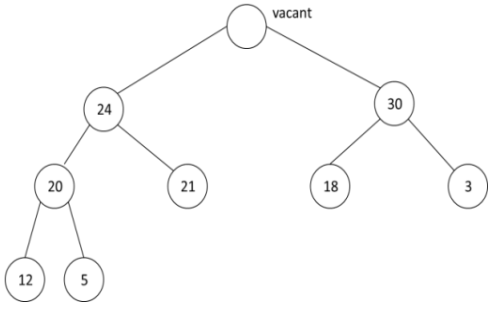
3 - "اهمل" (discard) هذه العقدة الأخيرة (عارفين أنها في موضعها الصحيح) بإنقاص سعة الكومة (decreasing the heap size)، واستدعاء (calling) الخوارزمية deleteMax لتطبيقها بالنسبة للجذر الجديد (new root) [المحتمل أن يكون في غير موضعه الصحيح (possibly incorrectly placed)].

4 - كرر عملية "الإهمال" هذه ("discarding" process) حتي تتبقي (remains) لدينا عقدة واحدة فقط، أي أصغر عنصر (smallest element)، وبالتالي يكون في الموضع الصحيح في المنظومة.

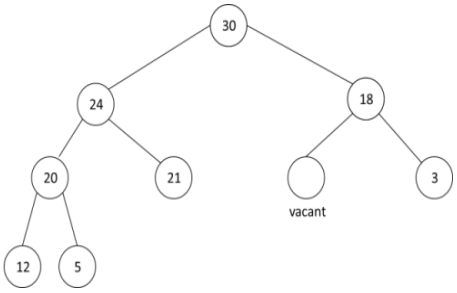
مثال 3–5: نفرض أن لدينا الكومة التي تظهر في الشكل 3–20–(أ). الكومة الأخيرة 3–20–(هـ) في شكل 3–20 هي التي نحصل عليها بعد تطبيق تكرير واحد من عروة for في خوارزمية heapSort المعطاة في شكل 3–19. بينما الأشكال الأخري من 3–20– (ب) إلي 3–20–(د) توضح خطوات عمليات إعادة الترتيب (rearrangements) التي تقوم بتنفيذها خوارزمية deleteMax عن طريق استدعاء البرنامج الفرعي fixHeap والذي توكل إليه مهمة إعادة ترتيب العناصر المتبقية في الكومة – بعد إزالة المفتاح الموجود في الجذر – كي تحقق خاصية شجرة الترتيب الجزئي.



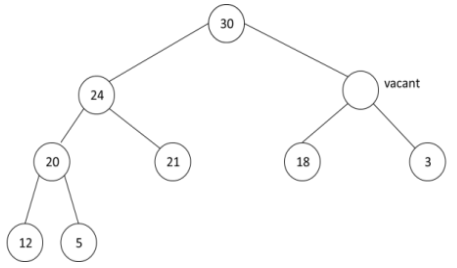
(ا) الكومة المعطاة



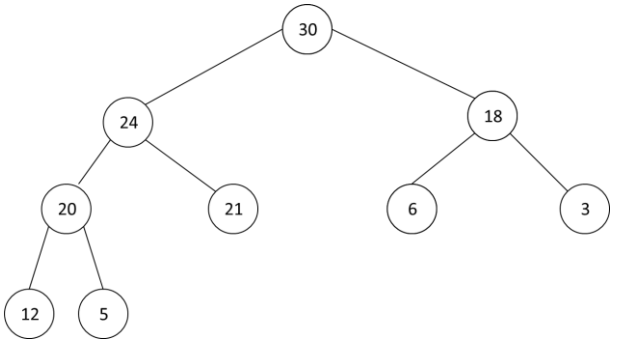
(ب) أزيل المفتاح الموجود عند الجذر. وأزيلت الورقة الموجودة أقصى اليمين في المستوى السفلي. والآن يجب إعادة ادخال k = 6.



(جـ) الابن الأكبر للعنصر vacant (وهو 30) أكبر من k، ولذلك فهو يتحرك لأعلى، وينزل vacant لأسفل.



(د) الابن الأكبر للعنصر vacant (وهو 18) أكبر من k، ولذلك فهو يتحرك لأعلى، وينزل vacant لأسفل.



(هـ) أخيرا نظرا لأن vacant ورقة، فلذلك يتم ادخال k = 6.

شكل 3–20

إزالة العنصر الموجود في الجذر وإعادة تحقيق خاصية شجرة الترتيب الجزئي

وفيما يلي نعرض المخطط العام لخوارزمية deleteMax.

```
void deleteMax ( H )          // outline
[ Copy the rightmost element on the lowest level of H into K.
  Delete the rightmost element on the lowest level of H.
  fixHeap ( H, K )
```

شكل 3–21

المخطط العام لخوارزمية deleteMax

كما نرى فإن معظم الخطوات يقوم بها الإجراء fixHeap. ونحتاج الآن إلي خوارزمية لبناء (constructing) كومة، وخوارزمية أخري للإجراء fixHeap. ونظراً لأن fixHeap يمكن أن يستخدم أيضا لحل مسألة بناء كومة، فلذلك سنتناوله أولاً فيما يلي.

الإجراء fixHeap

يقوم الإجراء fixHeap باستعادة خاصية شجرة الترتيب الجزئي في بنية كومة تتحقق فيها فعلاً هذه الخاصية في جميع المواضع ربما عدا الجذر. وعلي وجه التحديد فحين يبدأ fixHeap عمله يكون لدينا بنية كومة ذات جذر"فارغ" ("vacant" root). الشجرتان الجزئيتان (subtrees) هما شجرتا ترتيب جزئي، ولدينا عنصر إضافي – K مثلاً – نريد إدخاله. ونظراً لأن الجذر فارغ، فإننا نبدأ هناك ونحاول وضع K والعقدة الفارغة في مكانيهما الصحيحين. وفي موضعه النهائي يجب أن يكون العنصر K [وبتعبير أدق مفتاح العنصر K] أكبر من أو يساوي كلاً من ابنيه. وبناءً عليه ففي كل خطوة نقارن K مع أكبر ابنَيْ العقدة الفارغة حالياً. فإذا كان K هو الأكبر (أو مساوياً) فإنه يمكننا إدخاله في العقدة الفارغة حالياً، وإلا فإن الابن الأكبر يتم تحريكه لأعلي (moved up) إلي العقدة الفارغة، ثم تكرر العملية.

تابع مثال 3–5: يوضح شكل 3–20 في أجزائه من 3–20– (ب) إلي 3–20– (هـ) في مثال 3–5 عمل الإجراء fixHeap. وبالعودة للوراء قليلاً فالشكل 3–20– (أ) يعطي البنية الابتدائية عند بداية عروة for في مخطط استدعاء heapSort (شكل 3–19). في البداية يقوم heapSort بنسخ المفتاح 50 من جذر H إلي curMax جاعلاً جذر الشجرة فارغاً فعلياً. ثم يستدعي الخوارزمية deleteMax التي تنسخ المفتاح 6 من العقدة الموجودة أقصى يمين المستوي السفلي في الشجرة إلي K، وتحذف فعلياً هذه العقدة.

وهذا ينتقل بنا إلي شكل 3–20– (ب) حيث يبدأ fixHeap (H, K). الابن الأكبر للعقدة الفارغة هو 30، وهو أيضاً أكبر من K والذي يساوي 6، وبالتالي فإن العنصر 30 يتحرك لأعلي إلي الموضع الفارغ، وتنزل العقدة الفارغة لأسفل لنصل إلي شكل 3–20– (جـ). ومرة أخرى فإن الابن الأكبر للعقدة الفارغة أكبر من K، وبالتالي تنزل العقدة الفارغة لأسفل مرة ثانية لنصل الي شكل 3–20– (د). وفي المرة التالية تكون العقدة الفارغة ورقة، وبالتالي يمكننا إدخال K فيها. وهكذا تستعيد H خاصية شجرة الترتيب الجزئي ونصل الي شكل 3–20– (هـ).

وفيما يلي نعرض المخطط العام للإجراء fixHeap.

أكوارزميث 3–6 : المخطط العام للإجراء fixHeap (Outline)

المدخلات: شجرة ثنائية غير خالية H (a nonempty binary tree) ذات جذر "فارغ" ("vacant" root)، بحيث أن كلاً من شجرتها الفرعية اليمني وشجرتها الفرعية اليسرى شجرة ترتيب جزئي، وعنصر K مطلوب إدخاله (to be inserted). ونوع H (type of) سنطلق عليه Heap في هذا المخطط العام. ونفرض أن عروات H (nodes of) من النوع Element.

المخرجات: شجرة ثنائية H تتكون من K وعناصر H الأصلية، وتحقق خاصية شجرة ترتيب جزئي (partial order tree property).

ملاحظة: بنية / هيكل (structure of) H **لا يتغير، ولكن محتويات** (contents) **عقدها** (nodes) **تتغير.**

```
void fixHeap ( H, K )        //OUTLINE
  if (H is a leaf)
      insert K in root (H)
  else
    set largerSubHeap to leftSubtree (H) or rightSubtree (H),
    whichever has larger key at its root. This involves one key
    comparison, unless rightSubtree is empty.
    if ( K.key ≥ root (largerSubHeap).key )
        insert K in root(H);
    else
      insert root (largerSubHeap) in root (H);
      fixHeap (largerSubHeap, K);
```

تمهيدية 5-3: الإجراء fixHeap **يجري في أسوأ حالة** (worst case) **عدد** $2h$ **من المقارنات بين المفاتيح علي كومة ارتفاعها** h .

البرهان: علي الأكثر (at most) تجري مقارنتان بين المفاتيح في كل تشغيل للإجراء، وارتفاع الشجرة ينقص (decreases) بواحد في الاستدعاء الارتدادي (recursive call). [هناك مقارنة واحدة ضمنية (implicit) في عملية تحديد largerSubHeap] .

الحفاظ علي خاصية الكومة Maintaining the heap property

وأهمية fixHeap ترجع الي معالجة الكومات، وهو يستخدم للحفاظ علي خاصية الكومة. ويمكننا أن نضع الشروط السابقة (preconditions) والشروط اللاحقة (post conditions) لهذه الخوارزمية كما يلي:

- قبل fixHeap: قد يكون E[i] أصغر من أحد ابنيه أو من كليهما.
- نفرض أن شجرتي i الفرعيتين اليسري واليمنى (left and right subtrees) كومتان.
- بعد fixHeap: الشجرة الجزئية التي جذرها i (subtree rooted at i) تصبح كومة (a heap).

صيغة آخري للإجراء fixHeap

```
void fixHeap ( E, i, n )
{
    ℓ = LEFT (i)
    r = RIGHT (i)
    if ( ℓ ≤ n  and  E[ℓ] > E[i] )
        largest = ℓ
    else
        largest = i
    if ( r ≤ n  and  E[r] > E[largest] )
        largest = r
    if ( largest ≠ i )
    {
        exchange E[i] with E[largest]
        fixHeap ( E , largest, n )
    }
}
```

شكل 3–22

الخطوات التنفيذية لخوارزمية fixHeap

1 - قـــارن E[i] مـــع كـــل مـــن ابنيـــه E[RIGHT(i)] & E[LEFT(i)].

2 - إذا لـــزم الأمـــر بَـــدِّل (swap) العنصـــر E[i] مـــع أكـــبر ابنيـــه للحفاظ علي خاصية الكومة (heap property).

3 - استمر في تنفيذ هذه العملية: المقارنة والتبديل متجهاً أسفل الكومة (down the heap)، حتي تصبح الشجرة الجزئية (subtree) المتجذرة عند الورقة (rooted at the leaf) كومة ببساطه (is trivially a heap). [انظر مثال 3–8].

بناء الكومة Heap Construction

نفرض أننا سنبدأ بوضع جميع العناصر في بنية كومة بترتيب اختياري (arbitrary order)، أي أنه ليس ضرورياً ان تتحقق خاصية شجرة الترتيب الجزئى فى أى كومة جزئية (subheap). تتبني خوارزمية fixHeap طريقة قَسِّمْ وتَغَلَّبْ أو فرق تسد (Divide–and–Conquer) لتحقيق خاصية شجرة الترتيب الجزئي. يمكن تحويل الشجرتين الفرعيتين (two subtrees) الي كومتين ارتدادياً (recursively)، ثم يمكننا استخدام fixHeap لإنزال العنصر الموجود في الجذر إلي موضعه الصحيح، ومن ثم دمج (combining) الكومتين الصغرتين (two smaller heaps) والجذر في كومة واحدة كبيرة. والحالة الأساسية (base case) هي شجرة مكونة من عقدة واحدة (أي ورقة)، وهذه فعلاً كومة. والخوارزمية التالية تنفذ هذه الفكرة.

الخوارزمية 3–7 المخطط العام للإجراء Constructheap (Outline)

[إجراء ارتدادي (recursive)]

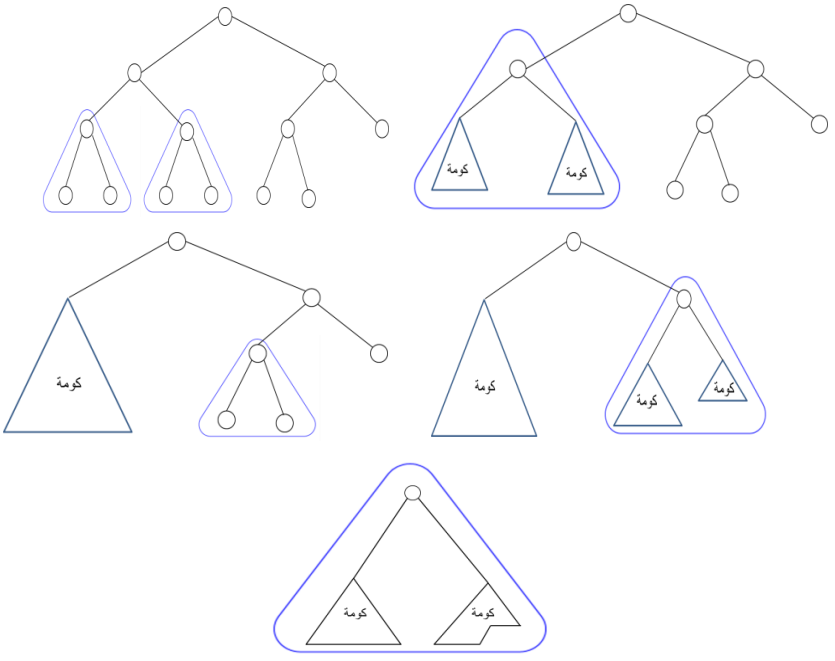
المدخلات: بنية كومة H لا تحقق بالضرورة خاصية الشجرة جزئية الترتيب.

المخرجات: H مع إعادة ترتيب عُقَدها نفسها بحيث تحقق خاصية الشجرة جزئية الترتيب.

الإجراء:

```
void constructHeap (H)       // OUTLINE
    if ( H is not a leaf )
        constructHeap (left subtree of H);
        constructHeap (right subtree of H);
        Element K = root (H);
        fixHeap ( H, K );
```

إذا كشفنا عن الخطوات التي تقوم بها هذه الخوارزمية من خوارزميات قسم وتغلب فسنري أنها تبدأ فعلياً بإعادة ترتيب (rearranging) العناصر القريبة من الأوراق أولاً، ثم تستمر في عملها متوجهة نحو أعلي الشجرة (up the tree). إهي تعد نوعاً من الاجتياز لاحق الترتيب (postorder traversal)] انظر شكل 3–23 الذي يوضح ذلك.



شكل 3–23

بناء الكومة

الأوراق تعد كومات. الإجراء fixHeap يُستدعي لكل شجرة فرعية محاطة بإطار

تنفيذ الكومة وأخوارزميث heapSort للترتيب بالكومة

Implementation of a heap and the heapSort Algorithm

عادة يتم تنفيذ الأشجار الثنائية كهياكل أو بنيات مترابطة (linked structures) حيث تحتوي كل عقدة (node) علي مؤشرين (pointers) [أو نوع آخر من الإسناد / المراجع (references)] إلي جذري شجرتيها الفرعيتين. وإنشاء واستخدام هذه البنية (structure) يتطلب وقتاً إضافياً (extra time) وحيزاً إضافياً (extra space) للمؤشرات. إلا أنه يمكننا تخزين واستخدام الكومات بكفاءة دون أي مؤشرات (pointers)

إطلاقاً. ففي أي كومة لا توجد أي عقد (nodes) عند عمق d (depth) – مثلاً – إلا إذا كان العمق d-1 ممتلئاً تماماً (completely filled). وبالتالي يمكننا تخزين أي كومة في منظومة (an array) مستوي بعد مستوي (level by level) مبتدئين بالجذر (root)، ومن اليسار إلي اليمين في كل مستوي. والشكل 3–24 يبين ترتيبات تخزين (storage arrangement) الكومتين اللتين تظهران في شكل 3–17.

مثال 3–6: قم بتخزين كل من الكومة 1 والكومة 2 اللتين تظهران في شكل 3–17 في منظومة.

الحـــل:

9	5	7	1	4	3	6

الكومة Heap 1

50	24	30	20	21	18	3	12	5	6

الكومة Heap 2

شكل 3–24

تخزين كومتي شكل 3–17 في منظومتين

وكي يكون هذا المخطط مفيداً يجب أن نكون قادرين علي الوصول إلي ابني أي عقدة بسرعة، وتحديد ما إذا كانت أي عقدة هي ورقة أم لا بسرعة. ومن المهم أن يكون الجذر مخزوناً في المنظومة عند المؤشر index 1 وليس 0 بالنسبة للصيغ التي سنعرضها فيما يلي.

نفـرض أننـا قـد أُعطينـا المؤشـر i لعقـدة مـا. يمكننـا اسـتخدام طريقـة للعد (counting argument) لبيـان أن الابـن الأيسـر للعقـدة سـيكون مؤشـره $2i$ والابـن الأيمـن مؤشـره $2i+1$. وبالمثـل يمكـن إثبـات أن الوالـد parent هـو $\lfloor i/2 \rfloor$. وكى نحـافظ علـي هـذه الصيغ البسـيطة فقـد اسـتخدمنا مؤشـرات تبـدأ عند 1 بالنسبة للكومات.

int PARENT (i)
    return $\lfloor i/2 \rfloor$

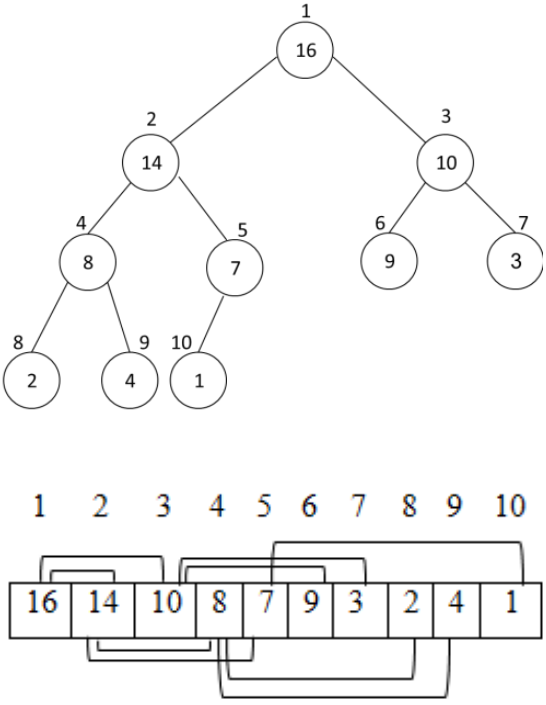
int LEFT (i)
    return $2i$

int RIGHT (i)
    return $2i + 1$

تخزين كومة (a heap) في منظومة array E:

- جذر الشجرة هو E[1]
- الابن الأيسر للعنصر E[i] هو E[$2i$]
- الابن الأيمن للعنصر E[i] هو E[$2i+1$]
- والد E[i] هو E[$\lfloor i/2 \rfloor$]

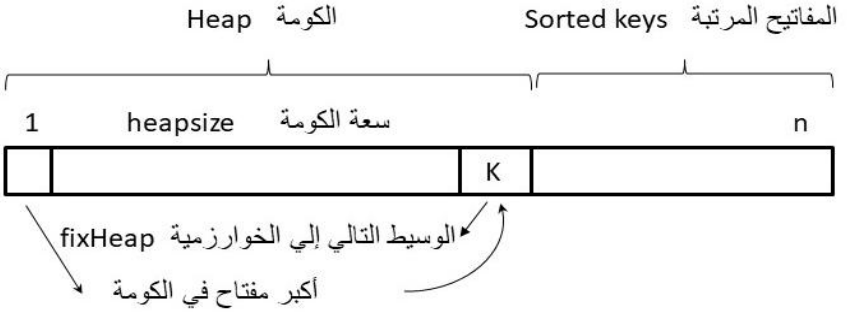
مثال 3–7: شكل 3–25 يوضح عملية تخزين كومة في منظومة، حيث الأقواس (arcs) – أعلي أو أسفل المنظومة – تربط بين الآباء parents والأبناء children.



شكل 3–25

والخاصـــية المذهلـــة بالنســـبة لخوارزميـــة heapSort أن إجـــراء الترتيب بأكمله يمكن تنفيذه في الموضع نفسه in place: فالكومات الصغيرة (small heaps) التي تبني أثناء مرحلة الإنشاء (construction phase)، والكومة التي تبني فيما بعد، والعناصر التي تحذف (deleted) يمكنها جميعاً أن تشغل occupy المنظومة E التي احتوت أصلاً (originally) علي مجموعة العناصر غير المرتبة (unsorted). وأثناء مرحلة الحذف (deletion phase) عندما تحتوي الكومة على k عنصر مثلاً، فإنها ستشغل المواضع الأولي – التي عددها k – في المنظومة. وهكذا فإننا نحتاج إلي متغير واحد

فقط للدلالة على نهاية الكومة (to mark the end of the heap).
ويوضح شكل 3–26 المشاركة في المنظومة (sharing of the array) بين كل من الكومة (heap) والعناصر المرتبة (sorted elements).
لاحظ أن مخطط fixHeap في الخوارزمية 3–6 كان يشتمل علي وسيطين (two parameters) فقط، بينما التنفيذ الأكثر تفصيلاً (more detailed) للخوارزمية فيما يلي يشتمل علي أربعة وسطاء.



شكل 3–26

الكومة والعناصر المرتبة في المنظومة

أخوارزميث 3–8 : خوارزميث heapSort

المدخلات: منظومة غير مرتبة E، وعدد العناصر n، حيث $n \geq 1$. ومدى المؤشرات (range of indexes) هو 1, …, n.

المخرجات: المنظومة E حيث عناصرها مرتبة ترتيباً غير تنازلي بالنسبة لمفاتيحها (in non-decreasing order of their keys).

ملاحظة: E[0] غير مستخدم. تذكر أن تحجز (allocate) عدد n + 1 من المواضع للمنظومة E.

الإجـــراء

```
void heapSort ( Element [ ] E, int n )
    int heapsize;
    constructHeap (E, n);
    // repeatedly remove root element and rearrange heap
    for ( heapsize = n; heapsize ≥ 2; heapsize -- )
        Element curMax = E[1];
        Element K = E[heapsize];
        fixHeap (E, heapsize - 1, 1, K);
        E[heapsize] = curMax;
```

وفيما يلي نعرض تنقيحاً (revision) للخوارزمية fixHeap (الخوارزمية 3–6) بعد تنفيذها باستخدام المنظومات (array implementation). وبالنسبة للخوارزمية constructHeap (الخوارزمية 3–7) فإن تنقيحها يمكن أن يتم أيضا بطريقة مماثلة، ولذلك سيحذف.

أخوارزميت 3–9: خوارزميت fixHeap

المدخلات: منظومة E تمثل بنية كومة (a heap structure)، heapsize: عدد العناصر في الكومة، root: جذر الكومة الجزئية (subheap) المراد ضبطها (to fix) – موضع فارغ (a vacant position)، K: العنصر المراد إدخاله (to be inserted) في الكومة الجزئية بطريقة تستعيد خاصية شجرة الترتيب الجزئي.

الشروط السابقة / القبلية (preconditions): الكومتان الجزئيتان (subheaps) المتجذرتان عند الابن الأيسر والابن الأيمن للجذر (rooted at the left and right children of root) تحققان خاصية شجرة الترتيب الجزئي.

المخرجات: الكومة الجزئية متجذرة عند root (rooted at)، وقد تم إدخال K فيها، وتحقق خاصية شجرة الترتيب الجزئي.

الإجـــراء

```
void fixHeap ( Element [ ] E, int heapsize, int root, Element K )
 int left = 2 * root, right = 2 * root + 1;
if ( left > heapsize)
 E[root] = K;        //  root is a leaf.
 else
   //  Determine largerSubHeap.
   int largerSubHeap;
   if ( left = = heapsize )
        largerSubHeap = left;       //  no right SubHeap.
   else if ( E[left].key > E[right].key )
        largerSubHeap = left;
   else
        largerSubHeap = right;
   //  Decide whether to filter K down.
   if ( K.key ≥ E[largerSubHeap].key )
        E[root] = K;
   else
     E[root] = E[largerSubHeap];
     fixHeap ( E, heapsize, largerSubHeap, K );
```

تحليل الخوارزمية heapSort heapSort Analysis

يمكننـا الآن أن نـري بوضـوح أن خوارزميـة heapSort هـي خوارزميـة ترتيـب في الموضـع نفسـه (an in-place sort) مـن حيـث حيـز العمـل (work space) المسـتخدم للعناصـر المراد ترتيبهـا. ورغـم أن بعـض الـبرامج الفرعيـة (subprograms) تسـتخدم الارتـداد، إلا أن عمـق (depth) الارتـداد محدود لنحو lg n، والذي لا يعد عادة مبعث قلق. ومع ذلك فيمكننا إعادة كتابـة هـذه الـبرامج الفرعيـة مـع الغـاء الارتـداد والعمـل في الموضـع نفسـه (work in place).

ويمكن إثبات أن عدد المقارنات التي تجريها خوارزمية constructHeap هو في حدود $\Theta(n)$. والآن ننظر إلي العروة الرئيسية في خوارزمية heapSort. بتطبيق التمهيدية 3- 5 نرى أن عدد المقارنات التي تجريها fixHeap علي كومة عدد عقدها k هو علي الأكثر $2\lfloor \lg k \rfloor$، وبالتالي فإن العدد الاجمالي لجميع عمليات الحذف deletions هو علي الأكثر $2\sum_{k=1}^{n-1} \lfloor \lg k \rfloor$. والمجموع يمكن أن يكون محدوداً (bounded) بتكامل صيغته

$$2\sum_{k=1}^{n-1} \lfloor \lg k \rfloor \le 2\int_{1}^{n} (lge)\, ln\ x\ dx$$

$$= 2(lge)\,(n\ ln\text{-}n) = 2(n\ lg\ (n) - 1.443\ n).$$

والنظرية التالية تلخص ما وصلنا اليه من نتائج.

نظرية 3–5: عدد المقارنات بين المفاتيح التي تجريها خوارزمية heapSort في أسوأ حالة هو 2n lg n + O(n). وخوارزمية heapSort هي خوارزمية ترتيب $\Theta(n \lg n)$.

البرهان: مرحلة إنشاء الكومة (heap construction phase) تجري علي الأكثر O(n) من المقارنات، وعمليات الحذف deletions تجري علي الأكثر 2 n lg (n).

وهكـذا نصـل إلـي النتيجـة أن خوارزميـة heapSort تجـري مقارنـات عددها $\Theta(n \lg n)$ في المتوسط (on the average) وكذلك في أسوأ حالة (in the worst case).

والآن قبـل أن نلخـص خطـوات تطبيـق طريقـة خوارزميـة heapSort لترتيب مجموعة عناصر معطاة في منظومة إدخال (input array) ترتيباً غير تنازلي، والحصول علي منظومة إخراج مرتبة (sorted output array)، نعرض أولاً ملخصاً لأسماء الخوارزميات التي احتجنا إليها في تطبيق هذه الطريقة، ووظيفة كل من هذه الخوارزميات.

(1) الخوارزمية fixHeap:

تقوم بتحويل بنية كومة (a heap structure) شجرتاها الجزئيتان الفرعيتان (2 subtrees) كومتان (heaps) إلي كومة (a heap).

fixHeap: a heap structure whose 2 subtrees are heaps → a heap

(2) الخوارزمية constructHeap:

تقوم بتحويل بنية كومة (a heap structure) [لا تحقق بالضرورة خاصية الشجرة جزئية الترتيب] الي كومة (a heap).

ConstructHeap: a heap structure → a heap

(3) الخوارزمية heapSort:

تقوم بتحويل منظومة غير مرتبة (unsorted array) إلي منظومة مرتبة (sorted array).

heapSort: unsorted array → sorted array

وفيما يلي – وقبل إعطاء أمثلة عددية أخرى توضح كيفية تطبيق طريقة الترتيب بالكومة – نستعرض تمثيل كل من هذه الخوارزميات الثلاث بشبه كود (pseudocode) / كود زائف باستخدام المؤشرات (pointers).

(1) شبه كود الخوارزمية fixHeap (p)

المدخلات: بنية كومة (a heap structure) فيها الشجرتان الفرعيتان (2 subtrees) المتجذرتان عند p (rooted at) كومتان (heaps)، حيث p هو مؤشر إلي الجذر (pointer to the root).

المخرجات: كومة (a heap) متجذرة عند p (rooted at). أي أن المعلومات / المفاتيح الصحيحة (correct information / keys) موجودة عند جميع عقد الكومة المتجذرة عند p.

الإجـــراء:

```
void fixHeap (p)
    done = false
    while ( (p is not a leaf) & (not done) )
        let m be a pointer to the child of p with the larger key
        if  ( key (p) <  key (m) )
            interchange key (p) with key (m)
            p = m
        else
            done = true
```

شكل 3–27

صيغة أخرى للإجراء fixHeap

يمكن كتابة هذا الإجراء بالصيغة المبينة في شكل 3- 22 حيث n هو عدد عناصر بنية الكومة المعطاة [أي عدد عناصر منظومة الإدخال E المعطاة].

(2) شبه كود الخوارزمية constructHeap

المدخلات: بنية كومة (a heap structures) حيث المفاتيح فيها في أي ترتيب اختياري (arbitrary order) [أي أنها ليست كومة (not a heap)].

المخرجات : بنية كومة حيث المفاتيح / المعلومات (keys / information) فيها مرتبة / صحيحة (ordered/correct) [أي أنها كومة (a heap)].

الإجـــــــراء:

```
void constructHeap (p)
    // let d be the depth of the heap structure
    // for each level, except the last do
    for (ℓ = d − 1; ℓ ≥ 0; ℓ--)
        for each non-leaf node p at level ℓ do
            fixHeap (p)
```

شكل 3–28

صيغة أخرى للإجراء constructHeap

يمكن كتابة هذا الإجراء بالصورة التالية، حيث n هو عدد عناصر بنية الكومة المعطاة [أي عدد عناصر منظومة الإدخال E المعطاة].

```
void constructHeap ( E, n )
```

```
for ( i = ⌊n/2⌋ ; i >= 1; i-- )
    fixHeap (E, i, n)
```

شكل 3–29

(3) شبه كود الخوارزمية heapSort

المدخلات: منظومة غير مرتبة E (unsorted array) عدد مفاتيحها $n \geq 1$.

المخرجات: المنظومة E مرتبة ترتيباً غير تنازلي (sorted in non-decreasing order).

الإجـــراء

```
void heapSort (p)
    constructHeap
    let p be a pointer to the root of the heap
    for ( i = n; i ≥ 2; i -- )
        E_i = key (p)
        let q be a pointer to the rightmost leaf in the last level
        in a heap of i nodes
        key (p) = key (q)
        // Remove q form the heap
        fixHeap (p)        // a heap with i-1 nodes
    E_1 = key (p)
```

شكل 3–30

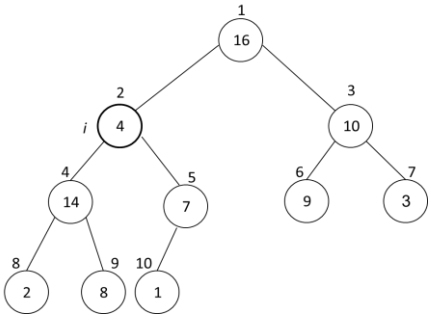
صيغة أخرى للإجراء heapSort

```
void heapSort ( E, n )
[   constructHeap
    for ( i = n; i ≥ 2; i --)
    [   exchange E[1] with  E[i]
        fixHeap (E, 1, i-1)
```

شكل 3–31

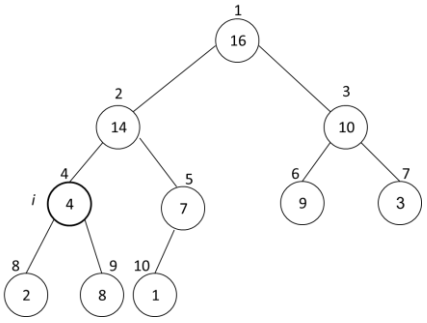
والآن نعود لمتابعة الأمثلة التي توضح بإذن الله خطوات طريقة الترتيب بالكومة.

مثال 3–8 : طبق خوارزمية fixHeap علي بنية الكومة التالية (شكل 3–32).

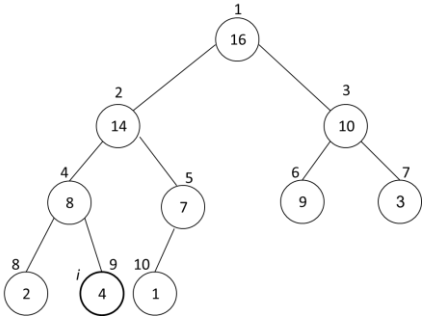


شكل 3–32

الحـــل: العقدة 2 (node) [أي العقدة رقم 2 التي تحتوي علي المعلومات / المفتاح 4] تنقض خاصية الشجرة جزئية الترتيب (4 < 14 & 4 < 7) وبمقارنة العقدة 2 مع كل من ابنيها [المفتاحين 7 ,14] وتبديلها (swapping it) مع أكبرهما (المفتاح 14) نحصل علي بنية الكومة التالية.



وبالاستمرار نحو أسفل الشجرة والتبديل حتي توضع القيمة (4) في مكانها الصحيح (properly placed) في جذر شجرة جزئية (root of a subtree) هي كومة (تكبيرية) (a max-heap) نحصل علي الشكل التالي، وفي هذه الحالة الكومة (التكبيرية) هي ورقة (a leaf).



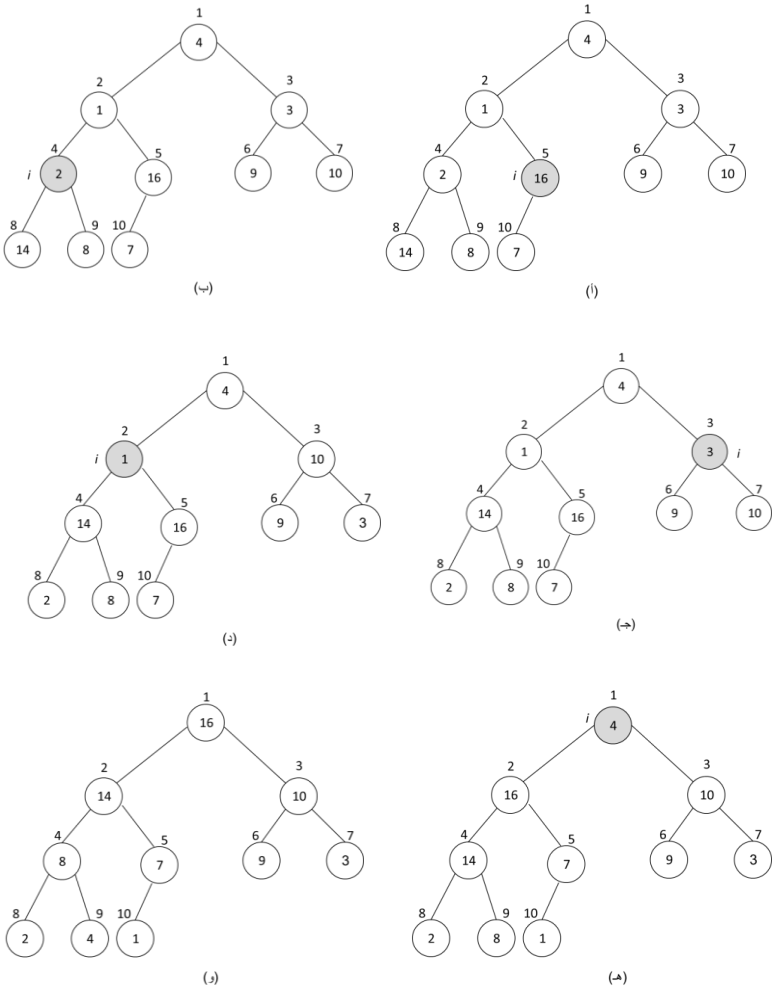
مثـال 3–9: طبق خوارزمية heapSort لترتيب عناصر منظومة الإدخال (input E array) التالية (شكل 3–33) ترتيباً غير تنازلي.

E	4	1	3	2	16	9	10	14	8	7

شكل 3–33

الحـــل: تبدأ الخوارزمية بإنشاء كومة (a heap) من عناصر منظومة الإدخال E المعطاة، وعدد هذه العناصر n = 10. ولذلك نبدأ بإذن الله بوضع

هذه العناصر في بنية كومة (a heap structure) H [شكل 3–34(أ)] ثم نطبق الخوارزمية constructHeap لتحويل H إلي كومة (a heap) [شكل 3–34 (و)].



شكل 3–34

بناء الكومة H من عناصر المنظومة E

وفيما يلي بيان الخطوات التي يتم تنفيذها في هذه الأشكال (أ) إلى (و)

(أ) الشجرة الثنائية / بنية الكومة H التي تمثل المنظومة E، وقد وضعت فوق عقدها أرقام مؤشرات (indexes) عناصر E، وهي: 1, 2…, 10. ويظهر في الشكل (أ) مؤشر عروة i (loop index) مشيراً إلي (referring to) العقدة رقم 5 node قبل استدعاء fixHeap، حيث يلاحظ أن $i = \lfloor 10/2 \rfloor = 5$ هي أول قيمة تأخذها i في عروة for [شكل 3–29]، حيث ستبدأ الخوارزمية constructHeap عملها بإعادة ترتيب العناصر القريبة من الأوراق أولاً مبتدئة عند المستوي $\ell = d - 1$ [شكل 3- 28، والخوارزمية 3–7]. ونظراً لأن العقدة 5 لا تنقض خاصية الكومة (حيث 7 < 16) فإنه لا يحدث أي تعديل / تبديل في العقد، وينتهي تطبيق أول تكرير من عروة for.

(ب) البنية الناتجة بعد تطبيق تكرير واحد من عروة for. والآن المؤشر i ينتقل إلي القيمة $i = 5\text{-}1 = 4$ في التكرير التالي، ولذلك فإن i يشير إلي العقدة رقم 4 في الشكل (ب). و نلاحظ الآن أن العقدة 4 تنقض خاصية الكومة، حيث أنها أصغر من ابنيها 8 > 2 & 14 > 2، ولذلك نبدل مفتاحها 2 مع أكبر مفتاحي ابنيها (14, 8) أي مع 14. وبعد هذا التبديل يصبح المفتاح 2 في ورقة، ولذلك ينتهي تطبيق التكرير الثاني لعروة for.

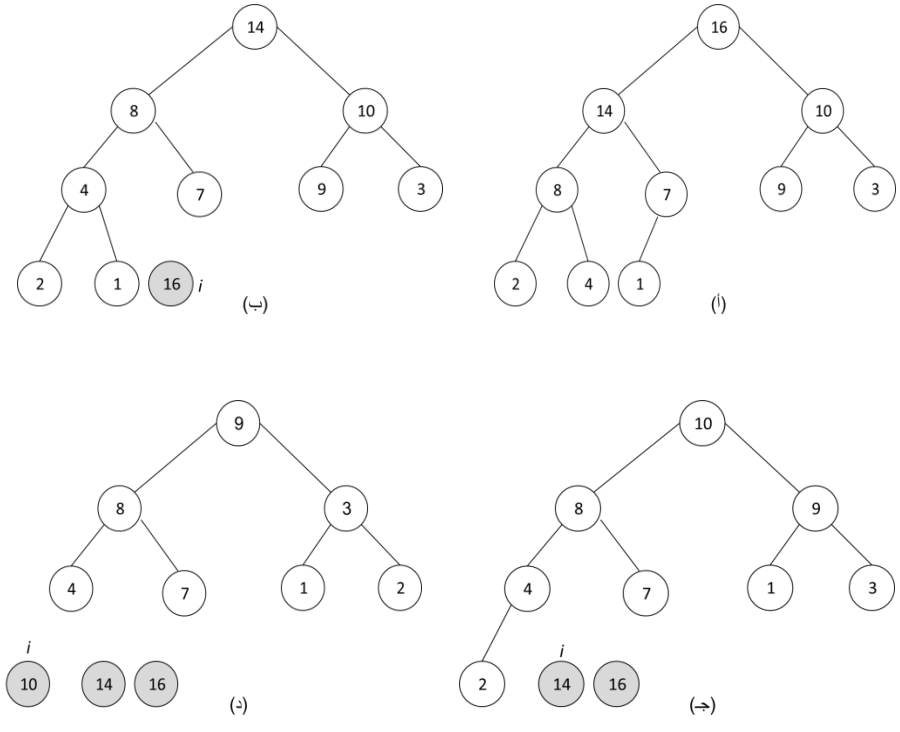
(جـ) البنية الناتجة بعد تطبيق التكرير الثاني. والآن $i = 3$ يشير إلي العقدة رقم 3. وحيث أن 10 > 3 & 9 > 3 فإننا نبدل 10 & 3. وتصبح 3 في ورقة، وينتهي هذا التكرير.

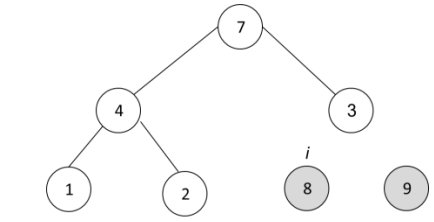
(د) البنية الناتجة بعد التكرير الثالث. والآن $i = 2$ مشيراً إلي العقدة رقم 2. و حيث أن 16 > 1 & 14 > 1 نبدل 16 & 1. والآن يصبح المفتاح 1 أصغر من مفتاح ابنه (7 > 1) فنبدلهما ليصبح 1 في ورقة، وينتهي هذا التكرير.

(هـ) البنية الناتجة بعد التكرير الرابع. والآن $i = 1$ مشيراً إلي الجذر. وحيث أن $10 > 4$ & $16 > 4$، فإننا نبدل 16 & 4. ثم نجد أن $14 > 4$ & $7 > 4$، فنبدل 14 & 4. فنجد أن $8 > 4$، حيث 8 هو مفتاح الابن الأيمن. فنبدل 8 & 4 لتصبح 4 في ورقة وينتهي هذا التكرير.

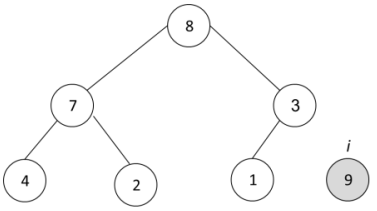
(و) البنية الناتجة بعد التكرير الخامس والأخير، وهي الكومة H المطلوبة. ولاحظ أنه كلما كنا نستدعي fixHeap لأي عقدة، فإن شجرتيها الفرعيتين تكونان فعلا كومتين.

والآن بعد أن انتهينا من إنشاء الكومة H تقوم الخوارزمية heapSort بوضع عناصرها في منظومة مرتبة. ويوضح شكل 3- 35 [الأشكال الجزئية (أ) إلى (ك)] خطوات هذه العملية حتي الحصول علي المنظومة المرتبة المطلوبة E.

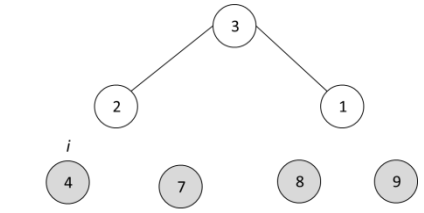




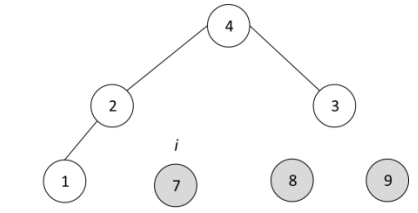
(و)



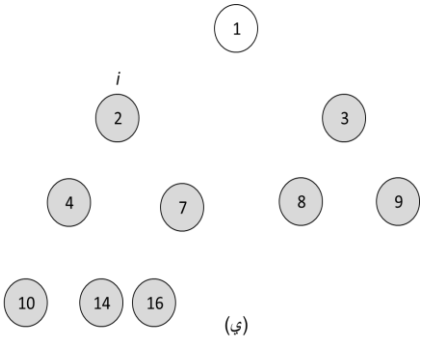
(هـ)



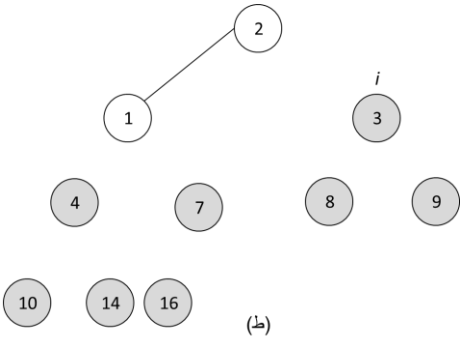
(ح)



(ز)



(ي)



(ط)

E	1	2	3	4	7	8	9	10	14	16

(ك)

شكل 3- 35

تكوين المنظومة المرتبة E من عناصر الكومة H

(أ) الكومة H التي أنشأتها الخوارزمية constructHeap. نبدأ بالجذر وهو أكبر عنصر (16) فنضعه في مكانه الصحيح E[10] في المنظومة بتبديله مع العنصر الموجود في آخر موضع في المنظومة. أي ننفذ عملية التبديل E[i] ↔ E[1]، حيث المؤشر i يساوي i = n = 10 [أنظر شكل 3–31 الذي يبين خطوات الإجراء heapSort]. وبأسلوب آخر نبدل المفتاحين 1 & 16 في العقدتين 10 & 1. ثم "نهمل" العقدة الأخيرة 10 التي تحتوي الآن علي المفتاح 16 (عارفين أنها في موضعها الصحيح)، ونعتبر أن سعة الكومة الآن أصبحت 9، ثم نستدعي الخوارزمية fixHeap لتطبيقها بالنسبة للجذر الجديد الذي يحتوي علي المفتاح 1. ونظرًا لأن 10 > 1 & 14 > 1 نبدل المفتاحين 14 & 1، ثم نبدل المفتاحين 8 & 1، ثم نبدل المفتاحين 4 & 1، ليصل 1 إلي ورقة، وينتهي تطبيق الخوارزمية fixHeap بالنسبة للجذر الجديد 1، ونحصل علي الشكل (ب).

(ب) يبين هذا الشكل كلا من الكومة الجديدة التي سعتها 9، وأكبر عنصر (16) تم حذفه/ إهماله/وضعه في موضعه الصحيح في المنظومة المرتبة. ونكرر الخطوات السابقة علي هذه الكومة الجديدة التي سعتها 9 والتي فيها أكبر عنصر هو 14. فالآن i=9، ونضع 14 في مكانه الصحيح E[9] في المنظومة، ونبدل المفتاحين 1 & 14، ونهمل العقدة التي تحتوي علي المفتاح 14، وتصبح سعة الكومة 8. ثم نستدعي fixHeap لتطبيقها بالنسبة للجذر الجديد الذي يحتوي علي المفتاح 1. ونظراً لأن 10 >1 & 8 >1 نبدل 10 & 1 ثم نبدل 9 & 1، ليصبح 1 في ورقة، وينتهي تطبيق الخوارزمية fixHeap بالنسبة للجذر الجديد 1، ونحصل علي الشكل (جـ).

(جـ) يبين الكومة الجديدة التي سعتها 8، وأكبر عنصرين 16 & 14 تم وضعهما في موضعيهما الصحيحين في المنظومة المرتبة. ونكرر الخطوات السابقة علي هذه الكومة الجديدة التي سعتها 8 وأكبر عنصر فيها 10، والذي يوضع في مكانه الصحيح E[8] . ثم نبدل 2 & 10، ونهمل العقدة التي تحتوي علي 10، وتصبح سعة الكومة 7. ثم نستدعي fixHeap لتطبيقها بالنسبة للجذر الجديد 2. ونظراً لأن 9 > 2 & 8 > 2 نبدل 9 & 2، ثم نبدل 3 & 2 لتصبح 2 في ورقة، ونحصل علي الشكل (د).

(د) يبين الكومة الجديدة التي سعتها 7، وأكبر 3 عناصر 16 , 14 ,10 والذين تم وضعهم في المنظومة المرتبة. أكبر عنصر في الكومة الجديدة 9 يوضع في مكانه الصحيح، ونبدل 2 & 9، ثم 8 & 2، ثم 7 & 2، ونحصل علي الشكل (هـ).

وهكذا . . . فالأشكال من (ب) إلي (ي) تبين الكومة الناتجة مباشرة بعد كل استدعاء للخوارزمية fixHeap [السطر الأخير في صيغة الإجراء heapSort في شكل 3–31]، كما تبين قيمة المؤشر *i* في ذلك الوقت.

(ك) هذا الشكل يعطي المنظومة المرتبة الناتجة E.

مقارنة بين خوارزميات الترتيب

Comparison of Sorting Algorithms

الجدول التالي يلخص نتائج تحليل (analysis) سلوك (behavior) خوارزميات الترتيب التي درسناها في هذا الفصل. ورغم أن خوارزمية الترتيب بالدمج Mergesort قريبة من الحالة المثلي (optimal) في أسوأ حالة

(in the worst case)، إلا أنه توجد خوارزميات تجري عدداً أقل من المقارنات. والحد الأدني (lower bound) الذي حصلنا عليه سابقاً يعتبر جيداً. ومن المعلوم أنه دقيق (exact) لبعض قيم n، أي أن عدد $\lceil \lg n! \rceil$ من المقارنات يعد كافياً (sufficient) لبعض قيم n. ومن المعلوم أيضاً أن هذا العدد $\lceil \lg n! \rceil$ من المقارنات ليس كافياً لجميع قيم n. فمثلاً $\lceil \lg 12! \rceil = 29$ ، ولكن تم إثبات أن 30 مقارنة تعد ضرورية (necessary) [وكافية (sufficient)] لترتيب 12 عنصراً في أسوأ حالة.

الخوارزمية	أسوأ حالة	المتوسط	الحيز المستخدم
الترتيب بالإدخال InsertionSort	$n^2/2$	$\Theta(n^2)$	في موضعه
الترتيب السريع QuickSort	$n^2/2$	$\Theta(n \lg n)$	حيز إضافي متناسب مع lg n
الترتيب بالدمج MergeSort	n lg n	$\Theta(n \lg n)$	حيز إضافي متناسب مع n للدمج
الترتيب بالكومة HeapSort	2 n lg n	$\Theta(n \lg n)$	في موضعه

جدول 3–1

نتائج تحليل أربع خوارزميات ترتيب القيم المذكورة هى أعداد المقارنات

تمرينات الفصل الثالث

تمرينات الفصل الثالث

3–1) لاحظنا أن حالة من أسوأ حالات خوارزمية الترتيب بالإدخال (a worst case for Insertion Sort) تحدث عندما تكون المفاتيح ابتداءً مرتبة ترتيباً تنازلياً (initially sorted in decreasing order). صف (describe) ترتيبين ابتدائيين آخرين للمفاتيح يُعدَّان أيضاً من أسوأ حالات هذه الخوارزمية. بَيِّن المدخلات (inputs) التى يكون لها عدد مقارنات المفاتيح بالضبط (exact number of key comparisons) [وليس مجرد الرتبة المقاربة (not just the asymptotic order)] هو الأسوأ فى الإمكان (the worst possible).

3–2) كم عدد مقارنات المفاتيح التى تقوم بإجرائها طريقة الترتيب السريع (Quicksort) [الخوارزميتان 3–2 و 3–3] إذا كانت المنظومة مرتبة فعلاً (already sorted)؟ كم عدد تحريكات العناصر (element movements) التى تجريها؟

3–3) نفرض أن المنظومة E تحتوى على المفاتيح 1, 2, …, 8, 9, 10، ونود ترتيبها بطريقة الترتيب السريع.

أ) وضح كيفية ترتيب المفاتيح بعد كل من الاستدعائين الأولين لإجراء التجزئة فى خوارزمية 3–3. واذكر كم عدد تحريكات العناصر التى يجريها كل من هذين الاستدعائين للتجزئة.

ب) من هذا المثال قَدِّر العدد الكُلِّى لتحريكات العناصر التي يتم إجراؤها لترتيب عدد n من العناصر موجودة بترتيب تنازلي فى وضعها الابتدائى (initially in decreasing order).

3–4) كم عدد مقارنات المفاتيح التى تجريها خوارزمية الترتيب بالدمج Mergesort إذا كانت المفاتيح مرتبة فعلا (already in order) عندما تبدأ عملية الفرز/ الترتيب؟

3–5) ارسم شجرة القرار لخوارزمية الترتيب السريع حيث عدد العناصر n = 3. يجب تعديل الاصطلاحات (modifying the conventions) قليلا، حيث ستحمل بعض الفروع (branches) التسمية / العنوان (label) " ≤ " أو " ≥ ".

3–6) طَبِّق خوارزمية الترتيب بالدمج لترتيب القائمة (المنظومة) التالية، موضحاً خطوات الخوارزمية خطوة خطوة.

27 10 12 20 25 13 15 22

3–7) طَبِّق خوارزمية الترتيب السريع لترتيب القائمة (المنظومة) التالية، موضحاً خطوات الخوارزمية خطوة خطوة.

الوتد (pivot)
↓

15 27 12 22 13 10 20 25

3–8) يُعَدُّ إجراء التجزئة PARTITION المفتاح لخوارزمية الترتيب السريع Quicksort، حيث يقوم بإعادة ترتيب rearranging عناصر منظومة جزئية A[p .. r] (subarray) فى موضعها (in place). والشكل التالى (شكل 3–36) يلخص إجراء التجزئة واستدعاءه بخوارزمية الترتيب السريع لترتيب عناصر المنظومة A[p .. r]. ولترتيب عناصر منظومة A بأكملها فإن الاستدعاء الابتدائي (initial call) لخوارزمية الترتيب السريع يكون

```
QUICKSORT (A, 1, A.length)
PARTITION (A, p, r)
1-  x = A[r]
2-  i = p - 1
3-  for j = p to r - 1
4-      if ( A[j] ≤ x )
5-          i = i+1
6-          exchange A[i] with A[j]
7-  exchange A[i+1] with A[r]
8-  return  i + 1
QUICKSORT (A, p, r)
1-  if ( p < r )
2-      q = PARTITION (A, p, r)
3-      QUICKSORT (A, p, q - 1)
4-      QUICKSORT (A, q + 1, r)
```

شكل 3–36

أ) ما هى القيمة "q" التى يعيدها الإجراء PARTITION عندما تكون جميع عناصر المنظومة A[p .. r] لها القيمة نفسها (same value)؟

ب) عدِّل الإجراء PARTITION بحيث أن q = (p + r)/2 عندما تكون لجميع عناصر المنظومة A[p .. r] القيمة نفسها.

3–9) اذكر برهاناً مختصراً يفيد أن زمن تشغيل (running time)

الإجراء PARTITION على منظومة جزئية (subarray) حجمها n هو $\Theta(n)$.

3–10) ما هو التعديل الذى تجريه على الخوارزمية QUICKSORT كى تقوم بترتيب العناصر ترتيباً غير تزايدى (nonincreasing order)؟

3–11) ما هو زمن تشغيل (running time) خوارزمية الترتيب السريع QUICKSORT عندما تكون جميع عناصر المنظومة A لها القيمة نفسها (same value)؟

3–12) اثبت أن زمن تشغيل خوارزمية الترتيب السريع QUICKSORT يكون $\Theta(n^2)$ عندما تحتوى المنظومة A على عناصر متمايزة (distinct elements) ومرتبة ترتيباً تنازلياً (sorted in decreasing order).

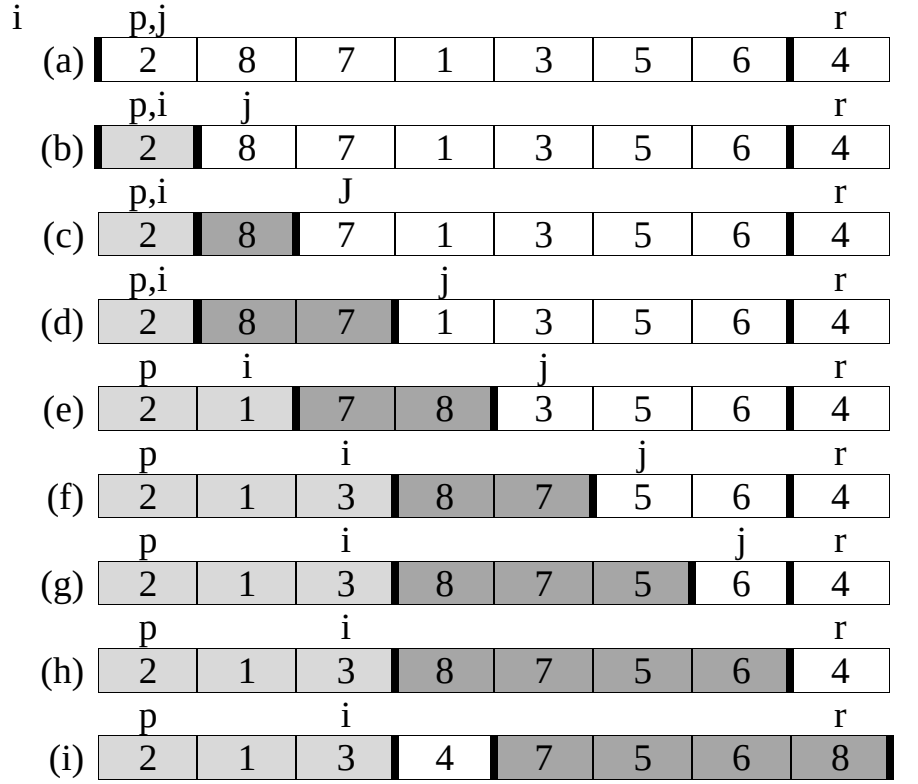
3–13) يوضح الشكل التالى (شكل 3–37) كيفية تطبيق إجراء التجزئة PARTITION [شكل 3–36 بالسؤال 3–8] على منظومة مكونة من ثمانية عناصر، حيث يختار الإجراء دائما عنصراً كعنصر وتدى (**pivot** element) x = A[r] يُجَزِّئ حوله المنظومة الجزئية A[p .. r]. ونلاحظ أن عناصر المنظومة المظللة تظليلاً خفيفاً (lightly shaded) تقع جميعها فى القسم الأول (first partition) حيث القيم ليست أكبر من x، بينما العناصر المظلله تظليلاً ثقيلاً (heavily shaded) تقع في القسم الثاني حيث القيم أكبر من x. وأما العناصر غير المظللة (unshaded) فهى

العناصر التى لم توضع بعد في أيٍّ من القسمين الأولين، والعنصر الأبيض الأخير هو الوتد x. ويوضح شكل 3–38 هذه المناطق الأربع (four regions) التى يحافظ عليها الإجراء PARTITION على المنظومة الجزئية A[p .. r].

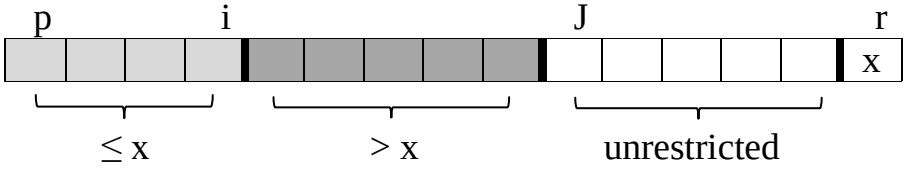
المطلوب: استخدام شكل 3–37 كنموذج (model) لتوضيح كيفية تطبيق إجراء التجزئة PARTITION على المنظومة

i) A = [8, 1, 6, 4, 0, 3, 9, 5]

ii) A = [13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21]



شكل 3–37



مطلق (لم يوضع بعد)

شكل 3–38

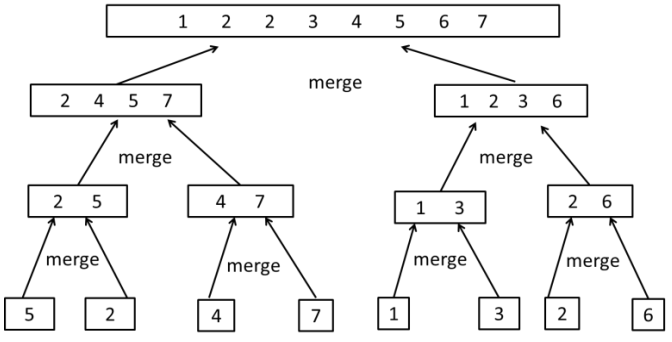
3–14) يوضح الشكل التالي (شكل 3–39) كيفية تطبيق إجراء الترتيب بالدمج MERGE-SORT (شكل 3–40) على منظومة مكونة من ثمانية عناصر. ولاستخدام هذا الإجراء لترتيب عناصر منظومة A[1 .. n] بأكملها فإننا نستدعى الإجراء ابتداءً (initial call) هكذا:

MERGE-SORT (A, 1, n)

المطلوب: استخدام شكل 3–39 كنموذج (model) لتوضيح كيفية تطبيق إجراء الترتيب بالدمج على المنظومة

A = [3, 41, 52, 26, 38, 57, 9, 49]

المتتابعة النهائية المرتبة
sorted sequence



initial sequence المتتابعة الابتدائية المعطاة

شكل 3–39

ترتيب عناصر المنظومة A = [5, 2, 4, 7, 1, 3, 2, 6] بالدمج

```
MERGE-SORT (A, p, r)
1- if ( p < r )
2-     q = ⌊(p + r)/2⌋
3-     MERGE-SORT (A, p, q)
4-     MERGE-SORT (A, q+1, r)
5-     MERGE (A, p, q, r)
```

شكل 3–40

3–15) هناك طريقة للترتيب / للفرز تُدْعى طريقة الترتيب الفقاعي (Bubble Sort). وتقوم هذه الطريقة بتنفيذ عدة مراحل / اجتيازات (passes) عبر المنظومة:

في كل مرحلة يقارن العنصر الأول فى المنظومة مع العنصر الثانى، ويوضع أصغرهما في الموضع الأول، ثم يقارن العنصر الثانى (الجديد، أي بعد ترتيب العنصرين الأولين) مع العنصر الثالث، ويوضع أصغرهما فى الموضع الثانى، وهكذا حتى ننتهى من كل عناصر المنظومة. ونواصل في كل مرحلة من المراحل التالية ترتيب العناصر بهذه الكيفية إلى أن لا يحدث أي تبديل لأي عنصرين، فتكون العناصر كلها حينئذ مرتبه ترتيباً تصاعدياً / غير تنازلى.

مثال: الشكل التالى (شكل 3–41) يوضح مراحل تطبيق طريقة الترتيب الفقاعي لترتيب عناصر المنظومة :[8, 6, 3, 5, 11, 2, 7, 4] = E.

المنظومة المعطاة	المرحلة الأولى	المرحلة الثانية	المرحلة الثالثة	المرحلة الرابعة	المرحلة الخامسة	المرحلة السادسة
8	6	3	3	3	2	2
6	3	5	5	2	3	3
3	5	6	2	5	4	4
5	8	2	6	4	5	5
11	2	7	4	6	6	6
2	7	4	7	7	7	7
7	4	8	8	8	8	8
4	11	11	11	11	11	11

شكل 3–41

(وتسمى الطريقة باسم "الترتيب الفقاعى" لأن العناصر الموجودة أسفل وضعها الصحيح تميل إلى التحرك لأعلى كالفقاعات. انظر مثلا إلى تحركات أصغر عنصر وهو 2).

الخوارزمية التالية تطبق هذه الطريقة.

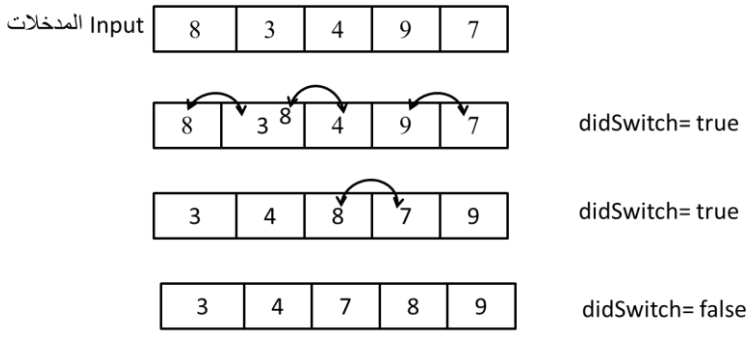
أخوارزميث 3–10 : خوارزميث الترتيب الفقاعى Bubble Sort

المدخلات: E: منظومة من عناصر، $n \geq 0$: عدد عناصر المنظومة.

المخرجات: المنظومة E حيث عناصرها مرتبة ترتيباً غير تنازلى (non decreasing order) بالنسبه لمفاتيحها (keys).

```
void bubbleSort ( Element [ ] E, int n )
    int numpairs;              // the number of pairs to be compared
    boolean didSwitch;         // true if an interchange is done
    int j;
    numpairs = n - 1;
    didSwitch = true;
    while (didSwitch)
        didSwitch = false;
        for ( j = 0; j <  numpairs; j + + )
            if ( E[j] > E[j+1] )
                interchange E[j] and E[j+1];
                didSwitch = true;
          // continue the for loop
        numpairs --;
```

والشكل التالى (شكل 3–42) يوضح تطبيق خوارزمية الترتيب الفقاعي لترتيب عناصر المنظومة E = [8, 3, 4, 9, 7].



شكل 3–42

المطلوب: أ) كم عدد مقارنات المفاتيح (key comparisons) التي تجريها خوارزمية الترتيب الفقاعى في أسوأ حالة (worst case)؟ وما هو ترتيب المفاتيح الابتدائى (initial arrangement of keys) الذى يُعَدُّ أسوأ حالة (a worst case)؟

ب) ما هو ترتيب المفاتيح الابتدائى الذي يُعَدُّ أحسن حالة (a best case) بالنسبه لخوارزمية الترتيب الفقاعى؟ أى ما هى المدخلات (inputs) التى تجعل الخوارزمية تجرى أقل عدد من المقارنات (fewest comparisons)؟ كم عدد المقارنات التي تجريها الخوارزمية في أحسن حاله؟

3–16) نفرض أن لدينا القائمة (المنظومة) التالية:

8 7 6 5 4 3 2 1

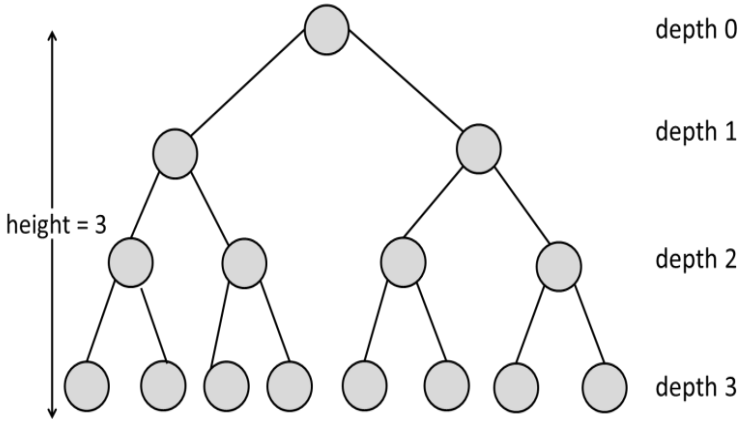
قم بترتيب (sort) عناصر القائمة ترتيبا تصاعديا (ascending order) بتطبيق كل من الخوارزميات التالية، موضحا خطوات الخوارزمية خطوة خطوة:

أ) خوارزمية الترتيب بالدمج MergeSort.

ب) خوارزمية الترتيب الفقاعى BubbleSort.

جـ) خوارزمية الترتيب السريع QuickSort، وافرض أن الوتد pivot هو العنصر الأول "8" في القائمة.

(17–3



أ) ما هو أقل (minimum) وأكبر (maximum) عدد من العناصر فى كومة ارتفاعها h (a heap of height)؟

ب) اثبت أنه إذا كانت لدينا كومة عدد عناصرها n، فإن ارتفاعها يكون $\lfloor \lg n \rfloor$.

3–18–أ) هل القائمة / المنظومة التالية تمثل كومة؟ علل إجابتك. افرض أن عناصر المنظومة معطاة ابتداءً عند المؤشر index 1. اعرض (view) المنظومة فى شكل بنية كومة (a heap structure).

i) القائمة A: List A: 60 90 80 40 70

ii) القائمة B: List B: 25 19 15 7 12 4 13 3 5 10

ب) إذا لم تكن بنية الكومة (the heap structure) كومة (a heap)، فطبق خوارزمية Construct-Heap / Fix-Heap لتحويلها إلى كومة.

3–19) طَبّق خوارزمية Heapsort لترتيب عناصر القائمة (المنظومة) التالية ترتيبا تصاعديا (ascending order). وضح خطوات الحل خطوة خطوة.

6 9 8 4 7

3–20– أ) نفرض أن المنظومة المطلوب ترتيب عناصرها ترتيباً أبجدياً (to be sorted into alphabetical order) بتطبيق خوارزمية Heapsort تحتوى ابتداءً (initially contains) على متتابعة الحروف (sequence of letters) التالية:

COMPLEXITY

وَضّح كيف سيتم ترتيب هذه الحروف فى المنظومة بعد مرحلة بناء الكومة (heap construction phase).

Algorithm ConstructHeap

Input: A heap structure H that does not necessarily have the partial order tree property.

Output: H with the same nodes rearranged to satisfy the partial order tree property.

```
void constructHeap (H)        // OUTLINE
  if (H is not a leaf)
        constructHeap (left subtree of H);
        constructHeap (right subtree of H);
        Element K = root (H);
        fixHeap(H, K);
```

ب) كم عدد المقارنات بين المفاتيح (key comparisons) التى يتم إجراؤها لبناء الكومة (to construct the heap) بهذه المفاتيح؟

3–21) نفرض أن مدخلات جدول 3–1 وهى أعداد المقارنات تمثل مقياسا لدرجة التعقيد الزمنية (measure of the time complexity) للخوارزميات المذكورة فى الجدول.

أ) رتب دوال التعقيد لخوارزميات الترتيب بالإدخال والترتيب السريع والترتيب بالدمج والترتيب بالكومة من أقل رتبة مقاربة (lowest asymptotic order) إلى أعلى رتبة مقاربة (highest asymptotic order).

i) فى أسوأ حالة (in the worst case).

ii) فى المتوسط (on the average).

ملاحظة: إذا تساوت دالتان أو أكثر فى الرتبة المقاربة فبين ذلك.

ب) قارن بين خوارزميات الترتيب الأربع من حيث الحيّز المستخدم(space usage).

3–22) اثبت أنه في أى شجرة فرعية فى كومة تكبيرية (max-heap) يحتوى جذر هذه الشجرة الفرعية على أكبر قيمة موجودة فى هذه الشجرة الفرعية.

3–23) ما الموقع الذى قد يشغله أصغر عنصر فى الكومة التكبيرية بفرض أن جميع العناصر متمايزة (distinct)؟

3–24) هل المتتابعة < 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 > تمثل كومة تكبيرية؟

3–25) نفرض أن عناصر منظومة هى (مبتدئين عند المؤشر1):

25, 19, 15, 5, 12, 4, 13, 3, 7, 10

هل هذه المنظومة تمثل كومة؟ علل إجابتك.

3–26) نفرض أن لدينا منظومة من مفاتيح متمايزة (distinct keys) مرتبة تنازلياً (in decreasing order)، والمطلوب ترتيبها تصاعدياً (to be sorted into increasing order) بخوارزمية الترتيب بالكومة (Heapsort).

أ) كم عدد المقارنات بين المفاتيح التى يتم إجراؤها فى مرحلة بناء الكومة (heap construction phase) [الخوارزمية 3–7] إذا كان عدد العناصر 10؟

ب) كم عدد المقارنات اذا كان عدد العناصر n؟ بين كيف استنتجت إجابتك.

جـ) هل المنظومة المرتبة تنازليا (in decreasing order) تمثل أحسن حالة (a best case)، أم أسوأ حالة (a worst case)، أم حالة وسطية (an intermediate case) بالنسبة للخوارزمية 3–7 ؟

3–27) فى خوارزمية heapSort (الخوارزمية 3–8) يمكننا حذف (eliminating) استدعاء واحد للإجراء fixHeap بتغيير التحكم فى عروة for إلى:

for (heapsize = n; heapsize≥ 3; heapsize - -)

أ) ما العبارة (statement) التى يجب إضافتها بعد عروة for لتأخذ فى الاعتبار حالة بقاء عنصرين فى الكومة (the case when two elements remain in the heap) [إن كانت هناك ضرورة لهذه العبارة]؟

ب) كم عدد المقارنات التى يتم حذفها [إن كانت هناك أى مقارنات تحذف]؟

3–28) نفرض أن S مجموعة عناصر عددها n، والمطلوب إيجاد أكبر k عنصر فى صدارة هذه المجموعة (first k largest elements)، حيث k (عدد تلك العناصر الكبرى الأولى) قيمته صغيرة جداً بالنسبة إلى n (أي أن $k << n$). يمكننا إيجاد تلك العناصر الكبرى الأولى بكفاءة عالية عن طريق بناء كومة (a heap) من عناصر المجموعة S والتي عددها n، ثم تطبيق الخطوات:

$i = n, n-1, n-2, \ldots, n-k+1$ من خوارزمية الترتيب بالكومة. ما أكبر قيمة للعدد الصحيح k بحيث تكون درجة التعقيد خطية في n ؟

3–29– أ) نفرض أن A منظومة عناصر عددها n، والمطلوب إيجاد العنصر الذى ترتيبه k من حيث الصِّغَر (k^{th} smallest element). يمكننا إيجاد هذا العنصر عن طريق ترتيب المنظومة تصاعديا (in ascending order)، ثم إعادة العنصر A[k]. ما درجة تعقيد هذه الطريقة؟

ب) هناك طريقة أكبر كفاءة لإيجاد هذا العنصر، وذلك بتعديل (modifying) خوارزمية الترتيب السريع Q-S بحيث أنها في معظم الحالات (in most cases) تقوم بإجراء عمليات أقل بكثير (much less operations) من تلك التي تجريها في أ). اكتب خوارزمية Q-S مُعَدَّلة، وسَمِّها الخوارزمية find-kth. ما درجة التعقيد المتوسطة لهذه الخوارزمية؟ [ملاحظة: يمكننا الوصول إلى درجة تعقيد O(n) في المتوسط].

الفصل الرابع:

خوارزميات المخططات البيانية

المخطط البياني المُوَجَّه

المخطط البياني غير المُوَجَّه .

المخطط البياني الجزئى .

المخطط البياني المُوَجَّه المتماثل .

المخطط البياني الكامل

علاقة التجاور

المسار فى مخطط بياني .

المخطط البياني المتصل .

الدورة فى مخطط بياني .

المُرَكَّبَة المتصلة

المخطط البياني المُوَزَّن

طرق تمثيل المخططات البيانية

اجتياز المخططات البيانية .

أولا: البحث بالعمق أولا .

ثانيا: البحث بالعرض أولا

مسائل الحلول المُثْلى للمخططات البيانية

أولا: إيجاد الشجرة المُوَلِّدة الصغرى .

1) **خوارزمية "بريم" لإيجاد شجرة مُوَلِّدة صغرى**

2) **خوارزمية "كروسكال" لإيجاد شجرة مُوَلِّدة صغرى**

ثانيا: أقصر مسارات من مصدر أحادى

- **تمرينات الفصل الرابع**

الفصل الرابع

خوارزميات المخططات البيانية Graph Algorithms

تعريف 4-1: **المخطط البياني المُوَجَّه** Directed Graph

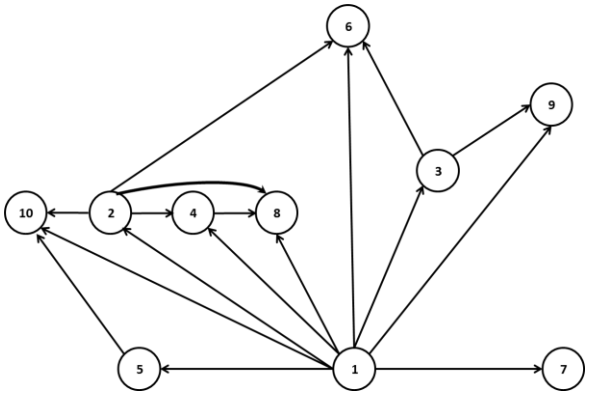
المخطط البياني (أو الرسم البياني) المُوَجَّه (directed graph, or **digraph)** هو زوج $G = (V, E)$ (a pair)، حيث V هى مجموعة عناصرها تدعى "رؤوساً" (vertices) و E هى مجموعة من **أزواج مرتبة (ordered pairs)** من عناصر V. والرؤوس كثيرا ما تدعى أيضاً "عُقَداً" (nodes). وعناصر المجموعة E يطلق عليها "أحرف" (edges) أو "أحرف مُوَجَّهَة" (directed edges) ، أو "أقواس" (arcs). وبالنسبة إلى الحرف المُوجّه (v, w) فى المجموعة E فإن v تُعَدُّ "ذيله" (its tail) و w تُعَدُّ "رأسه" (its head). والزوج (v, w) يُمثَّل فى الرسوم البيانية (diagrams) بالسهم $v \rightarrow w$ (the arrow) ، ونحن هنا سنكتبه ببساطة vw.

مثال 4-1: نفرض أن R هى العلاقة الثنائية (binary relation) المعرفة على المجموعة $S = \{1, 2, \ldots, 10\}$ والمكونة من الأزواج المرتبة (x, y) التى تحقق الشرط أن x هى عامل حقيقى / فعلى من عوامل y (a proper factor of)، أى أن $(x \neq y)$، وباقى (remainder of) (y / x) يساوى صفرا 0. وعادةً نكتب x R y كاصطلاح بديل للانتماء $(x, y) \in R$.

فى المخطط البياني المُوجّه (digraph) التالى (شكل 4–1): النقاط

هى عناصر المجموعة S، وهناك سهم (arrow) من x إلى y إذا وفقط إذا كانت x R y (if and only if). لاحظ أن العلاقة R متُعَدية (transitive): أى أنه اذا كانت x R y & y R z متحققتين فإن x R z تكون أيضا متحققة. ونرمز لهذا المخطط البياني المُوجّه بالزوج G = (V, E)، حيث

$V = \{1, 2, \ldots, 10\}$,
$E = \{(1,2), \ldots, (1,10), (2,4), (2,6), (2,8), (2,10), (3,6), (3,9), (4,8), (5,10)\}$.



شكل 4–1

مخطط بياني موجه يمثل العلاقة الثنائية R: " x عامل فعلى من عوامل y"

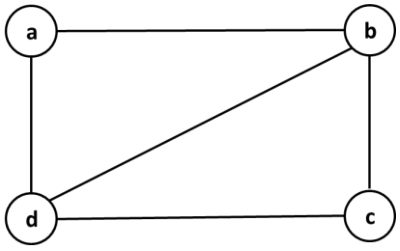
The relation R representing "x is a proper factor of y"

تعريف 4-2: **المخطط البياني غير المُوجّه** Undirected graph

المخطط البياني غير الموجه هو زوج G = (V, E)، حيث V هى مجموعة عناصرها تُدعى "رؤوسا"، و E هى مجموعة من **أزواج غير مرتبة (unordered pairs)** من عناصر متمايزة (distinct elements) من عناصر V. والرؤوس كثيرا ما تدعى أيضاً "عقدا" (nodes). وعناصر E يطلق عليها "أحرف" **أو أحرف غير مُوَجَّهة (undirected edges)** للتأكيد. وأى حرف يمكن اعتباره

مجموعة جزئية من V (subset of) تحتوى على عنصرين، وبالتالى فإن $\{v, w\}$ يرمز إلى حرف غير موجه. وفى الرسوم البيانية فإن هذا الحرف سيكون هو الخط $v - w$ (line) ، ونحن هنا سنكتبه ببساطة vw. وبالطبع فبالنسبة للمخططات البيانيه غير المُوجَّهة فإن $vw = wv$.

مثال 4-2: فى المخطط البياني غير الموجه فى الشكل التالى شكل (4–2):



شكل 4–2

مخطط بياني غير موجه

$V = \{a, b, c, d\}$

$E = \{(a, b), (a, d), (b, c), (b, d), (c, d)\}$

اصطلاحات Notations

- عادةً سنستخدم الاصطلاح (i, j) ليشير إلى الحرف الواصل بين الرأسين i, j بدلا من الاصطلاح $\{i, j\}$.
- سنرمز عادةً لرؤوس المخططات البيانية بالأعداد الطبيعية (natural 1, 2, 3, . . . numbers).
- سنستخدم الحرف n ليشير إلى عدد رؤوس المخطط البياني $n = |v|$، والحرف m ليشير إلى عدد أحرفه $m = |E|$. فمثلا فى المخطط البياني فى شكل 4-2:

عدد الرؤوس: $n = |V| = 4$

عدد الأحرف: $m = |E| = 5$

تعريف4–3–أ) المخطط البياني الجزئي Subgraph

يُقال إن المخطط البياني $G' = (V', E')$ هو مخطط بياني جزئى (subgraph) من المخطط البياني $G = (V, E)$ إذا كان $V' \subseteq V \ \& \ E' \subseteq E$. ومن تعريف المخطط البياني فيجب أيضاً أن يكون $E' \subseteq V' \times V'$.

ب) المخطط البياني المُوجّه المتماثل Symmetric digraph

هو مخطط بياني موجه (directed graph) بحيث أنه لكل حرف vw يوُجدَ ايضاً الحرف المعاكس wv (the reverse edge). وأى مخطط بياني غير موجه (undirected graph) له مخطط بياني موجه متماثل مقابل (corresponding symmetric digraph) وذلك بتفسير (interpreting) أى حرف غير موجه (undirected edge) باعتباره زوجاً من حرفين موجهين (a pair of directed edges) فى إتجاهين متعاكسين (in opposite directions).

جـ) المخطط البياني الكامل Complete graph

هو مخطط بياني (عادةً غير موجه) فيه حرف (an edge) بين كل زوج من الرؤوس (between each pair of vertices).

د) يُقال إن الحرف VW يحدث عند / يقع على (incident upon) الرأسين v,w، وكذلك العكس صحيح، أى يقال إن الرأسين v, w يحدثان عند/ يقعان على الحرف vw.

تعريف 4–4 **علاقة التجاور** "A" Adjacency relation

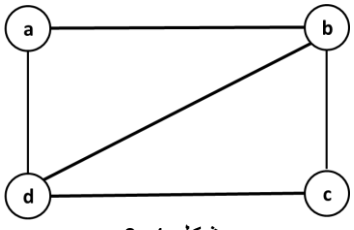
نفرض أن $G = (V, E)$ مخطط بياني (موجه أو غير موجه)، وأن v, w عنصران فى V. يُقال إن "**w مجاورة لـ v**" ("**w is adjacent to v**")، وتكتب **vAw**، إذا وفقط إذا كان $vw \in E$ (if and only if vw is in E). وبأسلوب آخر فإن vAw تعنى أن w يمكن الوصول اليها من v بالتحرك (moving) عبر حرف واحد (along one edge) من حروف G. وإذا كان المخطط G غير موجه فإن العلاقة A تكون متماثلة (symmetric). [أى أن wAv إذا وفقط إذا كان vAw (if and only if)].

تعريف 4-5: **المسار فى مخطط بياني** Path in a graph

المسار من v إلى w فى مخطط بياني $G = (V, E)$ هو متتابعة من الأحرف $v_0v_1, v_1v_2, \dots, v_{k-1}v_k$ (a sequence of edges) بحيث أن $v = v_0$ & $v_k = w$.

وطول المسار (length of the path) هو عدد الأحرف فى المسار أى هو k. وأى رأس v بمفردها (alone) تعتبر مساراً طوله صفر من v إلى نفسها. و"**المسار البسيط**" (a **simple path**) هو مسار بحيث أن الرؤوس $v_0, v_1, \dots, v_k$ جميعها متمايزة (are all distinct). **ويُقال إن رأسا w يمكن الوصول إليها** من الرأس v **(reachable from)** إذا كان هناك مسار من v إلى w.

مثال 4-3: شكل 4-3 يبيّن المسار $(a, b), (b, d), (d, c)$. وعادةً نرمز لأى مسار بذكر متتابعة الرؤوس (sequence of vertices) التى يمر بها (through which it passes). ولكن تذكر أن طول أى مسار هو عدد الأحرف التى اجتازها (number of edges traversed). فالمسار فى شكل 4- 3 هو a, b, d, c وطوله ثلاثة.



شكل 4-3
مسار من a إلى c

تعريف 4-6: **المخطط البياني المُتّصل** Connected graph

أ) يُقال إن المخطط البياني غير المُوجَّه (undirected graph) "**مُتّصل**" **(connected)** إذا وفقط إذا (if and only if) وُجدَ لكل زوج v, w من الرؤوس مسار (a path) من v إلى w.

ب) يُقال إن المخطط البياني المُوجّه (directed graph) قَوِيُّ الاتصال / مُتّصل بقوة **(strongly connected)** إذا وفقط إذا (if and only if) وُجدَ لكل زوج v, w من الرؤوس مسار (a path) من v إلى w.

ملاحظة: السبب فى ذكر تعريفين منفصلين مع أن الشرط المذكور هو نفسه، هو أنه فى حالة المخطط البياني غير المُوجّه: إذا كان هناك مسار من v إلى w، فإنه تلقائياً (automatically) يوُجدَ مسار من w إلى v ، بينما فى حالة المخطط البياني المُوجّه قد لا يكون هذا صحيحا، ولذلك فقد استُخدم القيد "بقوة strongly"

(qualifier) ليشير إلى أن الشرط (condition) أقوي (stronger). وإذا نظرنا إلى المخطط غير المُوجّه على أنه نظام من شوارع ثنائية الاتجاه (a system of two- way streets)، والمخطط المُوجّه على أنه نظام من شوارع أحادية الاتجاه (one-way streets)، فشرط "قوة الاتصال" / "الاتصال القَوِي" (strong connectivity) يعنى أنه يمكننا أن نصل من أى موضع إلى أى موضع (from anywhere to anywhere) بالسير فى الشوارع أحادية الاتجاه فى اتجاهاتها الصحيحة. ومن الواضح ان هذا شرط أكثر صرامة / تشددا / تضييقاً (more stringent condition) مما لو كانت الشوارع ثنائية الاتجاه.

تعريف 4-7: **الدورة فى مخطط بياني** Cycle in a graph

أ) فى المخطط البياني المُوجّه: **الدورة (cycle)** هى مجرد مسار غير خالى (just a nonempty path) بحيث أن الرأس الأولى والرأس الأخيرة (first and last vertices) متطابقتان (identical). **والدورة البسيطة (simple cycle)** هى دورة لا تتكرر فيها أى رأس (no vertex is repeated)، باستثناء أن الرأسين الأولى والأخيرة متطابقتان.

ب) فى المخطط البياني غير المُوجّه: تعريفا الدورة والدورة البسيطة كما هما فى المخطط المُوجّه ولكن مع إضافة الشرط: إذا ظهر أى حرف (edge) أكثر من مرة (appears more than once) فيجب أن يظهر دائما فى التوجيه نفسه (with the same orientation)، بمعنى أنه إذا كانت

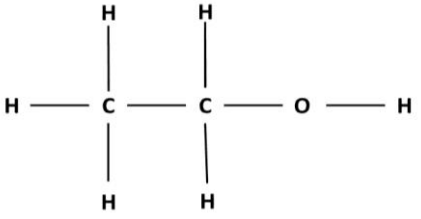
$$v_i = x \ \& \ v_{i+1} = y \qquad \text{for } o \le i < k$$

فلا يمكن أن توجد j بحيث أن $v_j = y \ \& \ v_{j+1} = x$.

جـ) يُقال إن المخطط البياني "**لادَوْرَوِيّ**" (**acyclic**) إذا لم يحتو على أى دورة.

د) المخطط البياني غير المُوجّه اللادوروي (undirected acyclic graph) يطلق عليه "**غابة غير موجهه**" **(undirected forest)**. وإذا كان المخطط البياني مُتّصلاً (connected) أيضاً، فيطلق عليه **شجرة حرة** (a **free tree**) أو "**شجرة غير موجهة**" (undirected tree). [انظر مثلا شكل 4–4].

هـ) غالبًا ما يُختصر اسم المخطط البياني المُوجّه اللادوروي (directed acyclic graph) إلى "**DAG**" [ولا يُفترض أن يحقق المخطط DAG أى شرط اتصالية (any connectivity condition)].



شكل 4-4

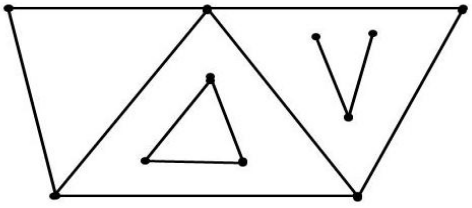
شجره حرة / شجرة غير موجهة (تمثل جزىء الكحول)

تعريف 4-8: **المُرَكِّبَة المتصلة** Connected component

المُرَكِّبَة المتصلة فى مخطط بياني غير موجه G هى مخطط بياني متصل "أَعظَمي" جزئي من G (a maximal connected subgraph of G). ويقال لمخطط بياني إنه "أعظمى" ضمن مجموعة ما من مخططات بيانية إن لم يكن مخططا بيانيا جزئيا فعليا (a proper subgraph) من أى مخطط بياني فى هذه المجموعة.

وأى مخطط بياني غير موجه إن لم يكن متصلا، فإنه يمكن تجزئته

(partitioned) إلى مُرَكِّبَات متصلة منفصلة، (separate connected components)، وهذه التجزئة وحيدة (unique). والشكل التالى (شكل 4-5) يبين مخططا بيانيا له ثلاث مركبات متصلة.



شكل 4-5

مخطط بياني له ثلاث مُرَكِّبَات متصلة

تعريف 4-9: **المخطط البياني المُوَزَّن** Weighted graph

المخطط البياني المُوَزَّن هو ثلاثى (V, E, W) (a triple) حيث (V, E) مخطط بياني (a graph) (موجه أو غير موجه)، وW دالة من E إلى R حيث R هى مجموعة الأعداد الحقيقية (the reals). وبالنسبة لأى حرف e (edge) فإن $W(e)$ يطلق عليها "**وَزْن** e" (**weight** of e).

ملاحظة: عادةً يشير وَزْن أى حرف e إلى خاصية من خصائص (properties) الحرف تبعا للتطبيق (application) المستخدم فيه هذا المخطط البياني. فمثلا إذا كان الحرف e يمثل طريقاً أو شارعاً أو مسافة بين بلدين أو نقطتين، فوَزْنه $W(e)$ قد يشير إلى تكاليف اجتياز هذا الطريق أو سرعة اجتيازه أو طوله أو زمن قطع هذه المسافة. ولذلك قد نستخدم أياً من المصطلحات (terms) التالية: وَزْن (weight)، وتكاليف (cost)، وطول (length) وسعة (capacity) هذا الحرف.

طرق تمثيل المخططات البيانية Graph Representations

رأينا طريقتين لتمثيل أى مخطط بياني على الورق: الأولى برسم صورة (drawing a picture) تمثل فيها الرؤوس بنقاط (points) والأحرف بخطوط (lines) أو أسهم (arrows)، والثانية بذكر عناصر مجموعتى الرؤوس والأحرف (by listing the vertices and edges). وفيما يلى ندرس بإذن الله بنى المعطيات /هياكل البيانات (data structures) التى تفيدنا فى تمثيل المخططات البيانية فى برامج الحاسوب. نفرض أن $G = (V, E)$ مخطط بياني حيث

$$V = \{v_1, v_2, \ldots, v_n\} \text{ ، } m = |E| \text{ ، } n = |V|.$$

(أ) التمثيل بمصفوفة التجاور

Adjacency (Incidence) Matrix Representation

يمكننا تمثيل المخطط G بمصفوفة n x n مربعة $A = (a_{ij})$ تدعى "مصفوفة التجاور" للمخطط G (Adjacency matrix for). وتعرف المصفوفة A هكذا:

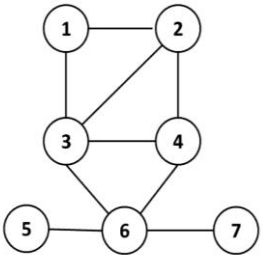
$$a_{ij} = \begin{cases} 1 & \text{if } v_i v_j \in E \quad \text{إذا كان} \\ 0 & \text{otherwise} \quad \text{ما عدا ذلك} \end{cases} \qquad \text{for } 1 \le i, j \le n$$

وبالنسبة للمخطط البياني غير المُوجّه فإن مصفوفة التجاور تكون متماثلة (symmetric) [ولذلك فإننا نحتاج لتخزين (storing) نصفها فقط]. وإذا كان $G = (V, E, W)$ مخططا مُوَزَّنا فإنه يمكن تخزين الأوزان فى مصفوفه التجاور بتعديل تعريفها كما يلي:

$$a_{ij} = \begin{cases} W(v_i v_j) & \text{if } v_i v_j \in E \quad \text{اذا كان} \\ c & \text{otherwise} \quad \text{ما عدا ذلك} \end{cases} \qquad \text{for } 1 \le i, j \le n$$

حيث c ثابت تعتمد قيمته على تفسير الأوزان والمسألة المطلوب حلها. وإذا نظرنا إلى الأوزان على أنها تكاليف (costs)، فقد نختار (∞) (أو أى عدد كبير جداً) كقيمة الثابت c لأن تكلفة اجتياز حرف غير موجود

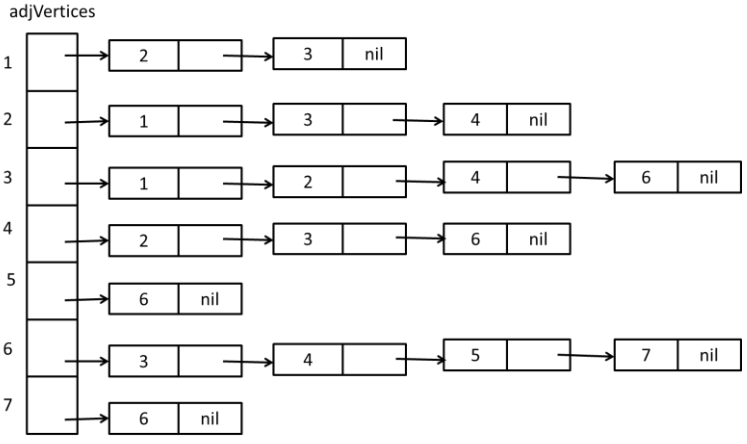
(cost of traversing a nonexistent edge) **تكون عالية جدا. وإذا كانت الأوزان سعات** (capacities)، **فإن اختيار** c = 0 **يكون مناسباً لأنه لا يمكن لأى شىء أن يتحرك عبر حرف غير موجود. انظر مثلا شكلي** 4–6 (أ، ب)، **و** 4–7(أ، ب).



(أ) مخطط بياني غير موجه
An undirected graph

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

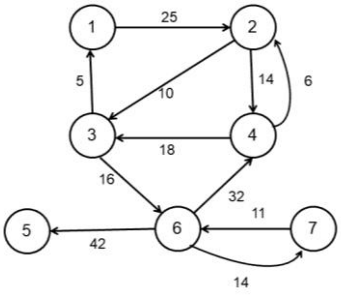
(ب) مصفوفة التجاور للمخطط
its Adjacency (incidence) matrix



(جـ) بنية قائمة التجاور للمخطط its adjacency- list structure

شكل 4–6

تمثيلان لمخطط بياني غير موجَّه بدون أوزان للحروف
(مصفوفة التجاور، و منظومة قوائم التجاور)
ويمكن أيضًا أن يكون مخططًا بيانيًّا مُوجّها متماثلاً



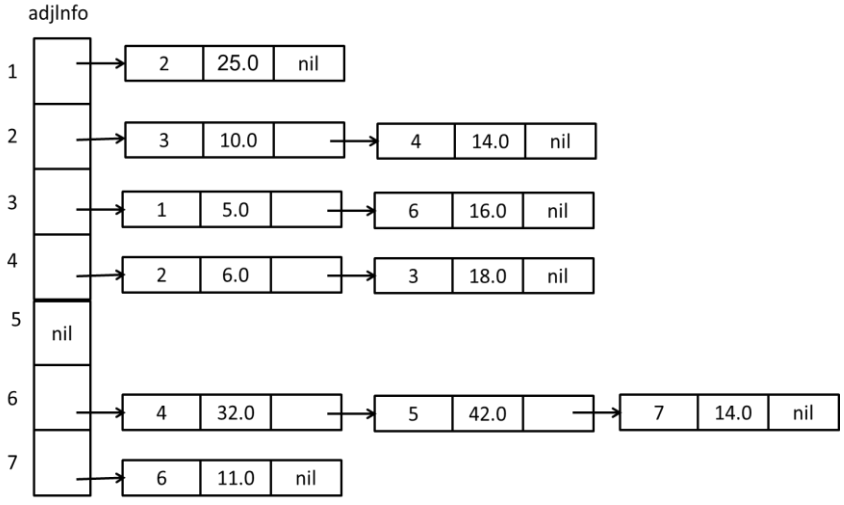
(أ) مخطط بياني مُوجَّه مُوَزَّن

A weighted digraph

$$\begin{pmatrix} 0 & 25.0 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 10.0 & 14.0 & \infty & \infty & \infty \\ 5.0 & \infty & 0 & \infty & \infty & 16.0 & \infty \\ \infty & 6.0 & 18.0 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 32.0 & 42.0 & 0 & 14.0 \\ \infty & \infty & \infty & \infty & \infty & 11.0 & 0 \end{pmatrix}$$

(ب) مصفوفة التجاور للمخطط

Its adjacency matrix



(جـ) بنية قائمة التجاور للمخطط its adjacency- list structure

شكل 4–7

تمثيلان لمخطط بياني مُوجّه مُوَزَّن

وتتطلب خوارزميات حل بعض مسائل المخططات البيانية فحص (examining) وتشغيل (processing) كل حرف (edge) بطريقةٍ ما على الأقل مرة واحدة. وإذا استخدمنا تمثيل المخطط البياني بمصفوفة التجاور

فيمكننا أيضاً أن ننظر إلى المخطط على أنه يشتمل على أحرف بين جميع أزواج الرؤوس المتمايزة (edges between all pairs of distinct vertices)، لأن خوارزميات عديدة تقوم بفحص كل عنصر فى المصفوفة لتحديد أى الأحرف موجودة فعلاً (which edges really exist). ونظرا لأن عدد الأحرف المحتملة هو n^2 فى مخطط بياني موجه، أو $n(n-1)/2$ فى مخطط بياني غير مُوجه، فدرجة تعقيد (complexity) مثل هذه الخوارزميات ستكون فى $\Omega(n^2)$.

(ب) التمثيل بمنظومة قوائم التجاور

Array of Adjacency Lists Representation

يمكننا تمثيل المخطط البياني G بمنظومة (array) مؤشرة برقم الرأس (indexed by vertex number) وتحتوى على قوائم مترابطة (linked lists) تُدعى " قوائم التجاور " (adjacency lists). ولكل رأس v_i (vertex) يحتوى العنصر رقم i فى المنظومة the i-th array (entry) على قائمة (a list) بمعلومات (information) عن جميع أحرف المخطط G التى تغادر v_i (leave). وفى المخطط البياني الموجه (directed graph) هذا يعنى أن v_i هى ذيل (tail) الحرف، بينما فى المخطط غير المُوجّه هذا يعنى أن الحرف يقع على v_i (incident upon). وقائمة v_i (list for) تحتوي على عنصر واحد لكل حرف (one element per edge) . وإذا أطلقنا على هذه المنظومة الاسم adjInfo فيمكننا تعريفها كما يلي:

```
list [ ] adjInfo = new list[n+1];
```

وسنستخدم المؤشرات (indexes) $1, 2, \ldots, n$، وبالتالى سنحجز (allocate) عدد $n + 1$ من المواضع (positions)، ولا نستخدم الموضع رقم

(the 0-th 0 position). وهكذا سيكون adjInfo[i] قائمة بمعلومات عن الأحرف التى تغادر الرأس v_i.

وتتميز بنية قائمة التجاور (adjacency–list structure) بأن الأحرف التى لا توجد فى المخطط G لا توجد أيضاً فى التمثيل. وإذا كان G هشاً / متناثراً (sparse) [أى أن عدد أحرفه أقل بكثير من n^2]، فإنه يمكن معالجته (processed) بسرعة. ويلاحظ أنه إذا كانت العناصر فى أى قائمة تجاور تظهر بترتيب مختلف (appear in a different order)، فإن البنية (structure) ستظل تمثل المخطط نفسه، ولكن أي خوارزمية تستخدم القائمة ستواجه أو ستقابل (encounter) العناصر بترتيب مختلف وقد تتصرف بطريقةٍ ما مختلفة. فأي خوارزمية يجب ألا تفترض أي ترتيب معين [إلا – بالطبع – إذا كانت الخوارزمية نفسها تنشئ (constructs) القائمة بطريقة خاصة].

ويبين شكل 4–6 مثالاً لبنية المعطيات (data structure) هذه لمخطط غير موجه (وممكن أيضا لمخطط موجه متماثل). بينما يبيّن شكل 4–7 مثالاً لهذه البنية لمخطط موجه مُوَزّن.

ويلاحظ أنه فى أى مخطط غير موجه يُمثَّل كل حرف مرتين، فمثلا إذا كان vw حرفا فى المخطط، فسيوجد عنصر للرأس w على قائمة تجاورv، وعنصر للرأس v على قائمة تجاورw. وهكذا يوجد عدد 2m من عناصر قائمة التجاور (2m adjacency-list elements) وعدد n من قوائم التجاور (n adjacency lists). وبالنسبة لأى مخطط موجه فإن كل حرف – كونه موجها – سيُمثَّل مرة واحدة. ولاحظ أن تمثيلات قوائم التجاور للمخططات غير المُوجَّهه، والمخططات المُوجَّهه المتماثلة المقابلة (corresponding symmetric digraphs) متطابقة (identical).

اجتياز المخططات البيانية

Traversing Graphs

تقوم معظم الخوارزميات المستخدمة لحل مسائل المخططات البيانية بفحص أو تشغيل كل رأس (vertex) وكل حرف (edge). وهناك استراتيجيتان لاجتياز المخططات البيانية تقدم كل منهما طريقة ذات كفاءة عالية لزيارة (visiting) كل رأس وكل حرف مرة واحدة بالضبط (exactly once). وهاتان الاستراتيجيتان هما:

- **البحث بالعرض أولاً (breadth-first search)**، وتُدعَى أيضاً **الاجتياز بالعرض أولاً (breadth-first traversal)**.
- **البحث بالعمق أولاً (depth-frist search)**، وتُدعَى أيضاً **الاجتياز بالعمق أولاً (depth-frist traversal)**.

وبناءً عليه فإن خوارزميات عديده – مبنية على أسس هاتين الاستراتيجيتين – يتم تشغيلها فى زمن (run in time) يتزايد خطيًّا (grows linearly) مع حجم مخطط الإدخال (input graph).

أولاً : البحث بالعمق أولاً Depth-First Search (DFS)

نفرض أن $G = (V, E)$ مخطط بياني غير موجَّه (an undirected graph) ، وأننا سنقوم بزيارة رؤوسه بالطريقة التالية:

- نختار أولاً رأس بداية v (select a starting vertex) و"نزورها".
- ثم نختار أى حرف (v, w) (any edge) واقع على v (incident upon) ونزور w .

- وعمومًا إذا فرضنا أن x هى أحدث رأس تمت زيارتها (the most recently visited vertex)، فإن البحث (search) يستمر باختيار حرف ما (x, y) لم يتم استكشافه بعد (some unexplored edge) واقع على x:

(i) فإن كانت y قد تمت زيارتها سابقاً (has been previously visited)، فإننا نبحث عن حرف جديد آخر (another new edge) واقع على x.

(ii) وإن كانت y لم تتم زيارتها من قبل، فإننا نزور y، ونبدأ البحث من جديد مبتدئين عند الرأس y.

- بعد إكمال البحث عبر جميع المسارات (paths) التى تبدأ عند y، فإن البحث يعود إلى x، وهى الرأس التى وصلنا منها إلى y أول مرة.
- تستمر عملية اختيار الأحرف غير المستكشفة (unexplored edges) الواقعة على x، حتى يتم استنفاد (exhausting) قائمة (list) هذه الأحرف .

يطلق على هذه الطريقة لزيارة رؤوس مخطط بياني غير موجَّه "البحث بالعُمق أولاً" (DFS) نظراً لأننا نستمر فى البحث متجهين إلى الأمام (إلى العُمق) (in the forward / deeper direction) قدر الاستطاعة (as long as possible).

ويمكن تطبيق طريقة DFS على المخطط البياني الموجَّه أيضا كما يلى: عند الرأس x نختار فقط الأحرف الموجَّهه (x, y) الخارجة من x (edges directed out of). وبعد استنفاد جميع الأحرف الخارجة من y، فإننا نعود

إلى x حتى لو كانت هناك أحرف أخرى موجَّهة نحو y لم يتم بحثها بعد (have not yet been searched).

وإذا طبقت طريقة DFS على أى مخطط بياني غير موجَّه وكان متصلاً (connected)، فمن السهل إثبات أن كل رأس ستزار، وكل حرف سيفحص (examined). وإن كان المخطط غير متصل، فإنه سيتم البحث فى مُرَكِّبَة متصلة فى المخطط. وعند إكمال (completion) مُرَكِّبَة متصلة نختار رأساً لم تتم زيارتها بعد لتكون رأس البداية الجديدة (the new starting vertex)، ونبدأ بحثاً جديداً.

وعندما تطبق طريقة البحث بالعمق أولاً DFS على أى مخطط بياني غير موجَّه $G = (V, E)$ فإنها تقوم بتجزئة أحرف E الى مجموعتين T, B: حيث نضع أى حرف (v, w) فى المجموعة T إذا كان الرأس w لم تتم زيارته من قبل عندما نكون عند الرأس v ونأخذ فى الاعتبار الحرف (v, w)، بينما نضع الحرف (v, w) فى المجموعة B إذا زرنا w من قبل. وأحرف المجموعة T يطلق عليها "**أحرف شجرة**" (**Tree edges**)، بينما أحرف B يطلق عليها "**أحرف خلفية /عكسية**" **(Back edges)**. والمخطط البياني الجزئى (V, T) هو غابة غير موجَّهة (undirected forest) يطلق عليها "**غابة مُولِّدة للمخطط G بالعمق أولاً**" (a **depth-first spanning forest** for G) .

وفى حالة ما إذا كانت الغابة تتكون من شجرة واحدة (a single tree)، فإن الغابة (V, T) يطلق عليها "**شجرة مُولِّدة بالعمق أولاً**" (a depth-first spanning tree). ويلاحظ أنه إذا كان المخطط G متصلاً (connected) فإن الغابة المُولِّدة بالعمق أولا ستكون شجرة. وفى الغابة

المُولِّدة بالعمق أولاً نعتبر أن جذر أى شجرة فى الغابة هو تلك الرأس التى بدأنا عندها البحث بالعمق أولا فى هذه الشجرة.

تعريف 4-10: **شجرة البحث بالعمق أولاً، غابة البحث بالعمق أولاً**

Depth-first search tree, depth-first search forest

الأحرف التى تؤدى إلى رؤوس "جديدة" لم تتم زيارتها أو اكتشافها من قبل ("new" / unvisited / undiscovered vertices) أثناء البحث بالعمق أولا فى مخطط بياني موجَّه G (a digraph) تُكوِّن شجرة ذات جذر (a rooted tree) يطلق عليها "**شجرة بحث بالعمق أولاً**" **(a depth-first search tree)** أو "**شجرة مُولِّدة بالعمق أولاً**" **(a depth-first spanning tree)**، وتُختصر – فى أى من الحالتين – إلى "**شجرة DFS**" **(DFS tree)**. وإذا لم نستطع الوصول (reaching) إلى جميع الرؤوس من رأس البداية (الجذر)، فإن اجتيازا كاملا للمخطط G سيُجزّئ (partitions) الرؤوس إلى عدة أشجار يُسمّى مجموعها الكلى (entire collection) "**غابة البحث بالعمق أولا**" **(depth-first search forest)** أو "**غابة مُولِّدة بالعمق أولاً**" **(depth-first spanning forest)**، وتُختصر – فى أى من الحالتين – إلى "**غابة DFS**" **(DFS forest)**.

وفيما يلى خوارزمية للبحث بالعمق أولا فى مخطط بياني.

الخوارزمية 4-1: **خوارزمية البحث بالعمق أولا فى مخطط بياني غير موجَّه**.

Depth-first search Algorithm of an undirected graph

المدخلات: مخطط بياني $G = (V, E)$ تمثله قوائم التجاور:

$L[v],\ \text{for}\ v \in V$

المخرجات: تجزئة E الى مجموعتين:

T: مجموعة أحرف شجرة (a set of tree edges)،

B: مجموعة أحرف عكسية / خلفية (a set of back edges).

الإجـــراء:

يقوم الإجراء الارتدادي التالي DFS(v) بإضافة الحرف (v, w) إلى المجموعة T إذا كنا سنصل إلى الرأس w أول مرة أثناء البحث عن طريق حرف من الرأس v. ونفرض أن جميع رؤوس المخطط ستُعطَى فى البداية العلامة "new" (initially marked).

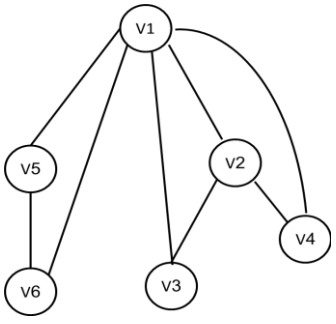
```
void DFS (v)
    {
1.      mark v "old";
2.      for ( each vertex w on L [v] )
3.        if ( w is marked "new" ) {
4.            add (v, w) to T
5.            DFS (w)
          }
    }
```

وفيما يلى الخوارزمية الكلية [البرنامج الرئيسى (main program)]:

```
    {
6.      T = Ø;
7.      for (all v in V ) mark v "new";
8.      while ( there exists a vertex v in V marked "new" )
9.          DFS (v)
    }
```

جميع أحرف E التى لم توضع فى T نعتبرها فى B. ويلاحظ أنه إذا كان الحرف (v,w) فى E، فإن w ستكون على L[v] و v ستكون على L[w]. وبناء عليه فإننا لا نستطيع أن نضع ببساطة الحرف (v, w) في B إذا كنا عند الرأس v وكان الرأس w مُعنوناً "old" نظراً لأن w قد يكون والد v (father of).

مثال 4-4: نفرض أن لدينا المخطط البياني G غير الموجَّه المبين فى شكل 4–8.



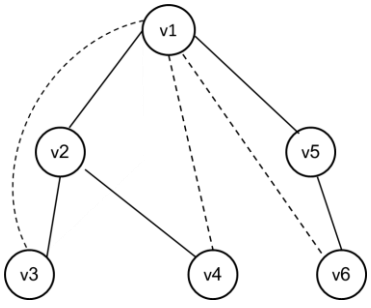
شكل 4–8

مخطط بياني G غير موجَّه

طبّق خوارزمية البحث بالعمق أولا لاجتياز هذا المخطط وتجزئة مجموعة أحرفه E إلى مجموعتين: T مجموعة أحرف شجرة (Tree edges)، و B مجموعة أحرف خلفية (Back edges).

الحـــل: سنستخدم اصطلاح بيان أحرف الشجرة فى T بخطوط متصلة / مصمتة (solid)، والأحرف الخلفية فى B بخطوط متقطعة (dashed). وكذلك سنرسم الشجرة (أو الأشجار) بوضع الجذر

(root) [وهو رأس البداية (starting vertex) الذى تم اختياره فى السطر 8] فى أعلى (top) الشجرة، ورسم أبناء (sons) أى رأس من اليسار إلى اليمين بالترتيب الذى أضيفت به أحرفهم فى السطر 4 فى خوارزمية DFS. بناء على هذا فإن إحدى التجزئات الممكنة للمخطط المعطى G إلى شجرة وأحرف خلفية نتيجة تطبيق بحث بالعمق أولا تظهر فى شكل 4–9.



شكل 4–9
تجزئة المخططG (فى شكل 4–8) إلى T و B بتطبيق DFS

فى البداية تكون جميع الرؤوس "new". نفرض أننا سنختار v_1 عند السطر 8. عندما ننفذ $DFS(v_1)$ فقد نختار $w = v_2$ فى السطر 2. ونظراً لأن v_2 معنونة "new" فإننا نضيف (v_1 , v_2) إلى T ونستدعى $DFS(v_2)$. الإجراء $DFS(v_2)$ قد يختار v_1 من $L[v_2]$، ولكن v_1 قد تم عنونتها "old". فنفرض أننا نختار $w = v_3$. نظراً لأن v_3 معنونة "new"، فإننا نضيف (v_2 , v_3) إلى T ونستدعى $DFS(v_3)$. والآن جميع الرؤوس المجاورة للرأس v_3 معنونة "old"، ولذلك فإننا نرجع إلى $DFS(v_2)$.

وبمتابعة $DFS(v_2)$ نجد الحرف (v_2 , v_4) ونضيفه إلى T، ونستدعى $DFS(v_4)$. ولاحظ أننا رسمنا v_4 إلى يمين v_3 وهو الابن الذى وجدناه سابقاً

للرأس v_2. والآن لا توجد أى رؤوس "new" مجاورة للرأس v_4، وبالتالى نرجع إلى $DFS(v_2)$. وحيث أننا لا نجد الآن أى رؤوس "new" مجاورة للرأس v_2، فإننا نرجع إلى $DFS(v_1)$. وبمواصلة البحث فإن $DFS(v_1)$ يجد v_5، و $DFS(v_5)$ يجد v_6. وهكذا تصبح جميع الرؤوس على الشجرة ومعنونة "old"، وبالتالى تصل الخوارزمية إلى نهايتها. وإذا لم يكن المخطط البياني متصلا فإن عروة السطرين 9 , 8 ستتكرر مرة واحدة لكل مُرَكِّبَة.

تحليل خوارزمية البحث بالعمق أولا Analysis of DFS

نود أن نحصل على درجة تعقيد الخوارزمية فى أسوأ حالة (worst case complexity) كدالة فى"n" عدد رؤوس المخطط البياني و "m" عدد أحرفه. نلاحظ أن **العمليتين الأساسيتين basic operations** فى الخوارزمية هما: اجتياز أحرف المخطط (traversing graph edges) وأوامرالإسناد لمتجه العلامات/العناوين (assignments to the mark vector). وبالنسبة **لعملية إسناد العناوين (mark assignments)** نلاحظ أنه يتم إجراء n عملية (n operations) فى السطر 7 فى البرنامج الرئيسى، وكذلك n عملية فى السطر 1 فى الإجراء DFS. وأما بالنسبة لعملية **اجتياز الأحرف (edge traversals)** نجد أولاً أن الاجراء DFS يُستدعى مرة واحدة (called once) لكل رأس إما من الخوارزمية الرئيسية (main algorithm) أو ارتدادياً (recursively) من الإجراء DFS نفسه. ونجد ثانيا أنه داخل DFS (within) يتم اجتياز قائمة التجاور لأى رأس (adjacency list of a vertex) مرة واحدة بالضبط (exactly once). ومن أولاً وثانيا نجد أن لدينا O(m) عمليات اجتياز كلية [a total of O(m) traversals].

ومما سبق نستنتج أن درجة التعقيد (complexity) هى $O(m + n)$،

والتى تكون عادة $O(m)$ نظراً لأن n تكون عادة $O(m)$. وأما بالنسبة لباقي الخطوات فى الخوارزمية فإنها تتطلب عددا مساوياً أو أقل من العمليات (operations).

ويمكننا أن نلخص ما سبق من تحليل خوارزمية DFS فى النظرية التالية.

نظرية 4-1: الخوارزمية 4-1 (خوارزمية البحث بالعمق أولاً) لاجتياز مخطط بياني غير موجَّه عدد رؤوسه n وعدد أحرفه m تتطلب خطوات (steps) عددها $O(\max\ (n, m))$.

البرهان: إذا عملنا قائمة (list) من الرؤوس ومسحناها (scanned) مرة واحدة، فإن السطر 7 فى الخوارزمية والبحث عن رؤوس "new" يتطلبان $O(n)$ من الخطوات. والزمن الذى نستغرقه فى الإجراء DFS(v) - باستثناء استدعاءاته الارتدادية لنفسه (exclusive of recursive calls to itself) - يتناسب طردياً مع (proportional to) عدد الرؤوس المجاورة للرأس v. والإجراء DFS(v) يتم استدعاؤه مرة واحدة فقط لرأس معطى v، وذلك لأن الرأس v يُعنون "old" حينما نستدعى DFS(v) أول مرة. وهكذا فإن الزمن الكلى الذى نستغرقه فى DFS يساوى $O(\max (n , m))$.

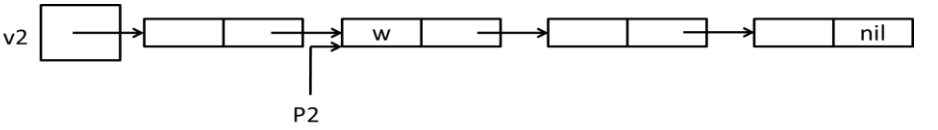
من الواضح أن خوارزمية DFS خوارزمية ذات كفاءة عالية لاستكشاف (exploring) مخطط بياني، حيث أنها تحتاج لرؤية مدخلاته من الرؤوس والأحرف (its input vertices and edges) مرة واحدة فقط.

ملاحظة:

كى يكون تحليلنا السابق صحيحاً (valid)، فإن خوارزميتنا DFS يجب أن تتذكر (remember) من أين تستمر/ تستأنف (where to continue / resume) اجتياز الأحرف فى السطر 2 فى الإجراء:

```
for ( each vertex w on L [v] )
    if ( w is marked "new" ) {
        add (v, w)  to  T
        DFS (w)
    }
```

فمثلا اذا كان لدينا التمثيل البياني التالى (منظومة قوائم التجاور):



المؤشر الحالى
current pointer

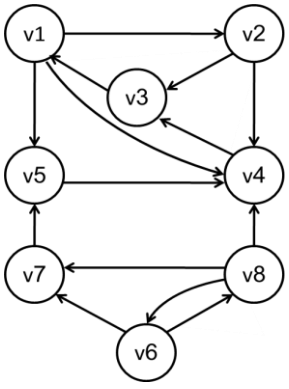
فهذا يعنى اننا نحتاج إلى منظومة مؤشرات (array of pointers): مؤشرا لكل رأس (vertex). غير أن هذا يتحقق لنا تلقائيا (automatically done for us) فى أى نظام (system) يسمح بالارتداد (recursion).

البحث بالعمق أولا فى مخطط بياني موجه

Depth-First Search of a directed graph

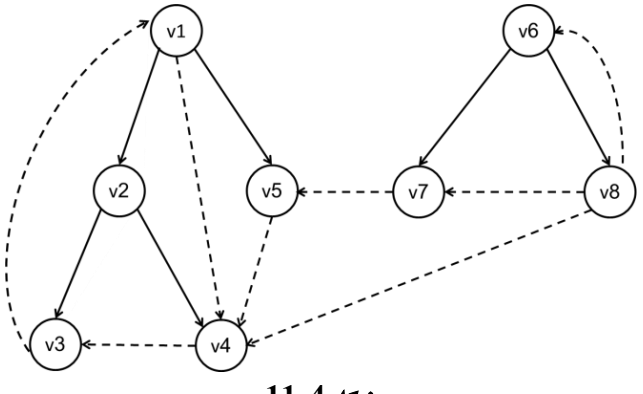
يمكننا تطبيق الخوارزمية 4- 1 أيضاً لإيجاد غابة مُولِّدة موجهة (a directed spanning forest) لمخطط بياني موجه $G = (V, E)$ اذا عَرَّفنا القائمة $L[v]$ وهى قائمة الرؤوس "المجاورة" "adjacent" للرأس v على أنها $\{w \mid (v, w) \text{ is an edge}\}$، أى أن $L(v)$ هى قائمة الرؤوس (vertices) التى أىٌّ منها هو رأس حرف (head of an edge) ذيله v (tail) .

مثال 4-5: نفرض أن لدينا المخطط البياني الموجه G المبين فى شكل 4-10.
طبّق خوارزمية البحث بالعمق أولاً لاجتياز المخطط G، وإيجاد غابة مُولِّدة له.



شكل 4-10
مخطط بياني موجّه G

الحل: باستخدام اصطلاح بيان أحرف الشجرة (tree edges) بخطوط مصمتة وغيرها بخطوط متقطعة - كما فعلنا سابقاً - نحصل على غابة مولّدة للمخطط G مبينة فى شكل 4-11.



شكل 4-11
غابة مولّدة للمخطط G (شكل 4-10)

لإنشاء غابة مولّدة سنبدأ عند الرأس v_1. الإجراء $DFS(v_1)$ يستدعى $DFS(v_2)$ الذى يستدعى $DFS(v_3)$. وهذا الأخير يتوقف دون أي إضافات للشجرة لأن الحرف الوحيد الذى ذيله v_3 يتجه إلى v_1 وهذا قد سبقت عنونته "old". وهكذا نعود إلى $DFS(v_2)$ الذى يضيف عندئذ v_4 كإبنٍ ثانٍ للرأس v_2. الآن $DFS(v_4)$ يتوقف ولا يقوم بأى إضافات إلى الغابة نظراً لأن v_3 قد سبق عنونتها "old". ثم يتوقف $DFS(v_2)$ لأن جميع الأحرف الخارجة من v_2 قد تم أخذها فى الاعتبار. وهكذا نعود أدراجنا إلى v_1 الذى يستدعى $DFS(v_5)$، وهذا يتوقف دون أى إضافات إلى الشجرة، وكذلك لا يستطيع $DFS(v_1)$ إضافة أى جديد.

والآن نختار v_6 كجذر لشجرة مولدة بالعمق أولا جديدة. وإنشاء هذه الشجرة مماثل لما سبق ونترك للقارئ الكريم متابعته. ولاحظ أن الترتيب الذى اخترناه لزيارة الرؤوس هو $v_1, v_2, \ldots, v_8$.

تعريف 4-11: يقال إن رأساً v هو **سَلَف / جَدّ** (**ancestor**) لرأس w فى شجرة ما إذا وقع v على المسار (path) من الجذر (root) إلى w. ويقال إن v **سلف فعلى** (a **proper ancestor**) للرأس w إذا كان v سلفاً للرأس w و$v \neq w$. وأقرب سلف فعلى للرأس v هو **والد** v (**parent of**). وإذا كان v سلفا فعليا للرأس w ، فإن w **سليل / حفيد فعلى من ذرية** v (a **proper descendant** of).

تعريف 4-12: أحـــرف أى مخطـــط بيانـــي موجـــه G تُصنـف / تجزأ

(classified / partitioned) إلى أربع فئات (categories) بالبحث فى G بالعمق أولاً بناءً على كيفية استكشافها (how they are explored) [أى اجتيازها فى اتجاهها للأمام (traversed [in their forward direction)]:

(1) **أحرف شجرة (Tree edges)**: وهى الأحرف التى تؤدى إلى رؤوس جديدة أثناء عملية البحث. أى أنه إذا كان w رأساً جديداً عند استكشاف الحرف vw، فإن vw يطلق عليه حرف شجرة، ويصبح v والد w (parent of) .

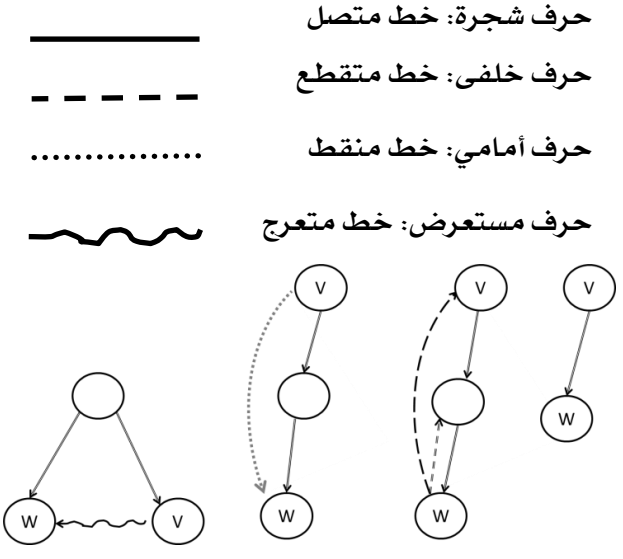
(2) **أحرف خلفية / عكسية (Back edges)**: وهى الأحرف التى تتجه من السلالة / الذرية من الأبناء والأحفاد (descendants) إلى السلف من الآباء والأجداد (ancestors) [مع احتمال اتجاه الحرف من رأس إلى نفسه]. أى أنه إذا كان w سلفا للرأس v، فإن v w يسمى حرفاً خلفياً (وذلك يشمل أيضا v v).

(3) **أحرف أمامية/متجهة إلى الأمام (Forward edges)**: وهى الأحرف التى تتجه من السلف من الآباء والأجداد (ancestors) إلى السلالة الفعلية / الذرية الفعلية (proper descendants) ولكنها ليست أحرف شجرة. أى أنه إذا كان w من سلالة / ذرية v، ولكن w تم اكتشافه قبل (discovered earlier) وقت استكشاف vw (exploring) فإن vw يسمى حرفا أمامياً. والحرف الأمامى يسمى أيضا "**حرف نازل / هابط / منحدر من سلف**" (a **descendant edge**).

(4) **أحرف مستعرضة / متعارضة / متقاطعة (Cross**

(edges): وهى الأحرف التى تصل بين رؤوس ليس بينها علاقة آباء/ أبناء (no ancaster / descendant relationship). أى أنه إذا لم يكن w سلفاً ولا سليلا للرأس v، فإن vw يسمى حرفاً مستعرضا.

الشكل التالى (شكل 4-12) يوضح الاصطلاحات التى نستخدمها لهذه الأنواع الأربعة من الأحرف.

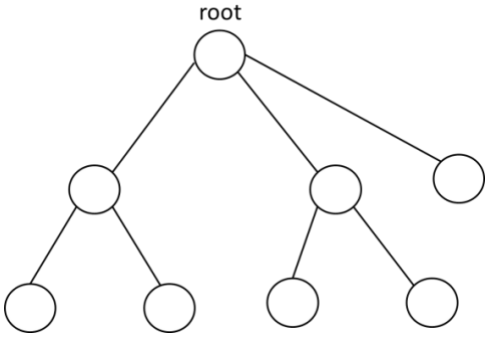


شكل 4-12

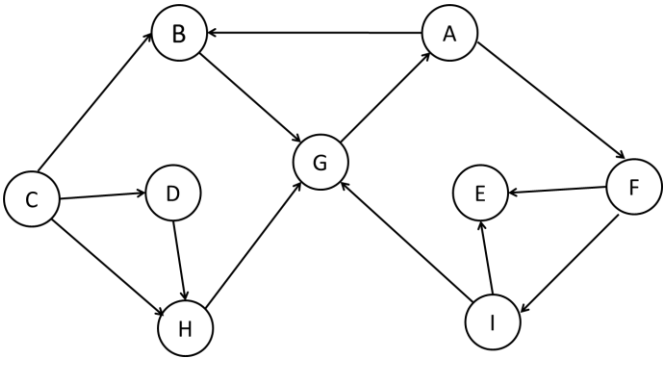
اصطلاحات الأحرف

والمثالان التاليان يوضحان هذه الأنواع من الأحرف [أحرف شجرة (tree edges)، وأحرف خلفية (back edges)، وأحرف أمامية (forward edges) وهى المتجهة من رأس إلى واحد من ذريتها ليس ابناً (not a child)، وأحرف مستعرضة (cross edges) وهى الأحرف بين رأسين ليس أى منهما من ذرية الآخر]. ولاحظ أن رأس وذيل (head and tail) الحرف المستعرض قد يكونان فى شجرتين مختلفتين.

ولتبسيط تصنيف الأحرف (edge classification) سنتبع اصطلاح رسم الأشجار كما هو مبين بالشكل التالى حيث الجذر (root) هو أعلى رأس وأبناء (children) أى رأس يظهرون فى المستوى الأدنى الذى يلى مستواه بترتيب زيارتهم (in order of visiting them) من اليسارإلى اليمين.



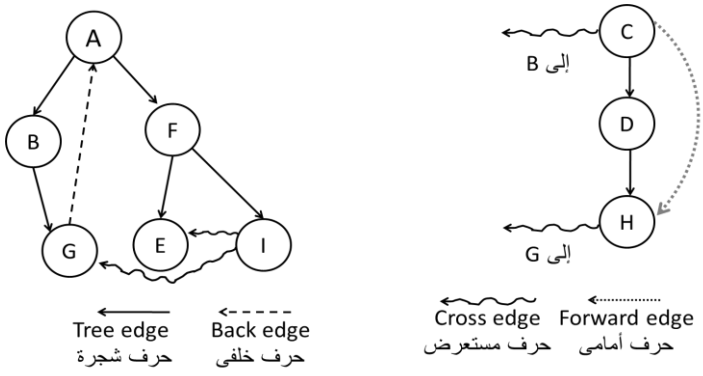
مثال 4-6: طبّق خوارزمية البحث بالعمق أولاً لاجتياز المخطط البياني الموجه G المبين فى شكل 4-13 وبيان أشجار البحث بالعمق أولا مع تصنيف أحرف المخطط.



شكل 4-13

مخطط بياني موجه G

الحـــل:

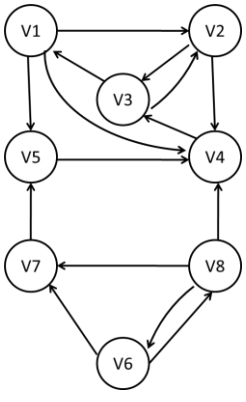


شكل 4-14

أشجار البحث بالعمق أولا للمخطط البياني الموجَّه G

Depth-first search trees for the digraph G

مثال 4-7: طبّق خوارزمية البحث بالعمق أولاً لاجتياز المخطط البياني الموجَّه G المبين فى شكل 4-15 وبيان الغابة المولدة له مع تصنيف أحرف المخطط. وضح أيضاً أرقام / ترتيب زيارة رؤوس المخطط أول مرة بهذه الخوارزمية.



شكل 4-15

مخطط بياني موجَّه G

A digraph G

الحـــل: فيما يلى قائمة بأسماء رؤوس المخطط، وأمام كل رأس نذكر أسماء الرؤوس التى تتجه إليها أحرف من هذا الرأس.

$V_1 : V_2, V_4, V_5$

$V_2 : V_3, V_4$

$V_3 : V_1, V_2$

$V_4 : V_3$

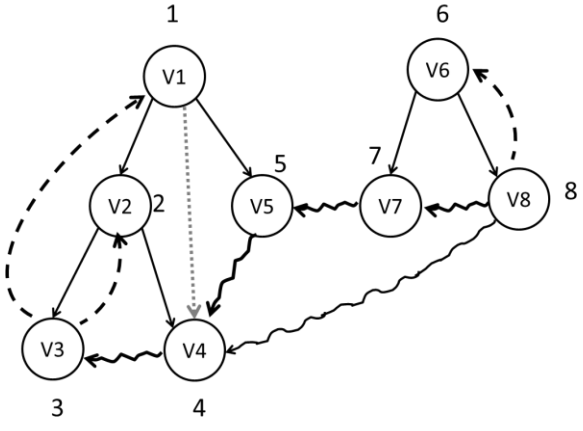
$V_5 : V_4$

$V_6 : V_7, V_8$

$V_7 : V_5$

$V_8 : V_4, V_6, V_7$

ويوضح الشكل التالى (شكل 4-16) الغابة المولّدة لهذا المخطط مع بيان تصنيف أحرفه، وأرقام ترتيب زيارة رؤوسه. وقد استخدمنا الاصطلاحات نفسها التى ذكرناها سابقا لأنواع الأحرف الأربعة.



شكل 4-16

غابة مولدة للمخطط البياني الموجَّه G (شكل 4–15)

Spanning forest for the digraph G

ملاحظتان حول الأشجار Remarks regarding trees

ملاحظة 1: أى مخطط بياني غير موجه $G = (V, E)$ (an undirected

(graph) يُعد شجرة إذا كان متصلا (connected) ولا يحتوى على أى دورة (cycle).

ملاحظة 2: العبارات التالية جميعها متكافئة (equivalent) بفرض أن G مخطط بياني عدد أحرفه m ، وعدد رؤوسه n:

(1) المخطط G شجرة.

(2) المخطط G غير موجه، ومتصل، ولا يحتوى على أى دورة.

(3) المخطط G لا يحتوى على أى دورة، وعدد أحرفه $n-1$.

(4) المخطط G متصل، وعدد أحرفه n-1.

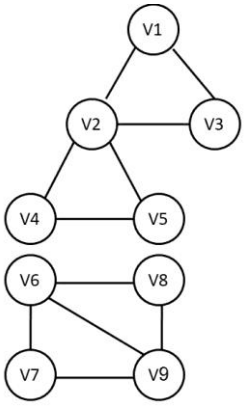
(5) المخطط G متصل، ولكن إذا حذف منه حرف (an edge is deleted) يصبح غير متصل (disconnected).

(6) المخطط G لا يحتوى على أى دورة، ولكن إذا أضيف إليه حرف (an edge is added) فإن دورة وحيدة تتكون (a unique cycle is formed).

(7) أى رأسين v , w فى المخطط G متصلان بمسار وحيد (a unique path).

المثال التالي يوضح **تطبيق خوارزمية البحث بالعمق أولا على مخطط بياني غير موجه وغير متصل.**

مثال 4-8: طبّق خوارزمية البحث بالعمق أولا لاجتياز المخطط البياني G غير الموجه وغير المتصل المبين فى شكل 4-17، والحصول على غابة مولّدة للمخطط، مع بيان أرقام ترتيب زيارة رؤوسه.



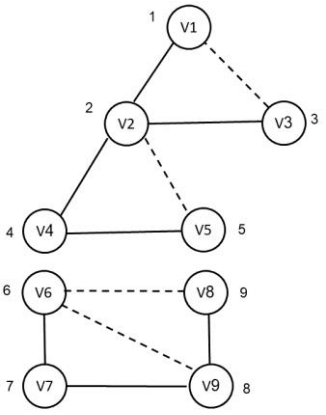
شكل 4-17

مخطط بياني G غير موجَّه وغير متصل

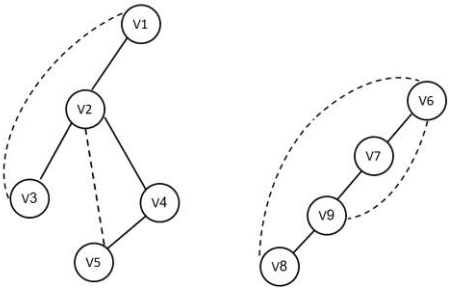
A disconnected undirected graph G

الحـــل: سنبدأ بإذن الله بإعطاء جميع رؤوس المخطط العلامة (mark) "new". وسنختار v_1 كرأس بداية (a starting vertex). ثم نتابع تطبيق خطوات الخوارزمية 4-1، والشكل التالى (شكل 4-18) يوضح نتيجة تطبيق هذه الخطوات، مع ملاحظة أن كلا من الشكلين (4-18- أ) و(4-18- ب) يعطى الغابة المولدة التى حصلنا عليها، ولكن الشكل (4-18- ب) يتبع اصطلاح رسم الأشجار الذى ذكرناه سابقاً حيث الجذر هو أعلى رأس، وأبناء أى رأس يظهرون فى المستوى الأدنى الذى يلى مستواه بترتيب زيارتهم من اليسار إلى اليمين. وتظهر فى الشكل (4-18- أ) أرقام ترتيب زيارة رؤوس المخطط G. وفى الغابة المولّدة التى حصلنا عليها نجد أن

$T = \{ (v_1,v_2), (v_2,v_3), (v_2,v_4), (v_4,v_5), (v_6,v_7), (v_7,v_9), (v_9,v_8) \}$



(أ)



(ب)

شكل 4-18

غابة مولّدة للمخطط البياني G غير الموجَّه وغير المتصل (شكل 4-17)

spanning forest for the disconnected undirected graph G

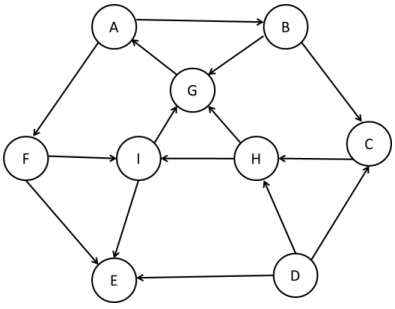
ثانياً: البحث بالعرض أولاً Breadth-First Search (BFS)

رأينا سابقا فى طريقة البحث بالعمق أولا لاجتياز مخطط بياني أنه عند زيارة رأس جديدة "new" ما v ، فإننا نستمر فى البحث متجهين إلى العمق (إلى الأمام) قدر الاستطاعة، ثم نرجع (back up) عبر آخر حرف تم اجتيازه

لنتفرع (branch out) فى اتجاه آخر. وهذا يؤدى إلى زيارة جميع الرؤوس الموجودة فى مخطط بياني جزئي مجاور للرأس v (subgraph adjacent to) قبل الذهاب إلى مخطط بياني جزئي جديد (new subgraph) مجاور للرأس v، وهذا يماثل الاجتياز سابق الترتيب للأشجار (preorder traversal of trees) الذى يقوم بزيارة جميع الرؤوس فى شجرة فرعية ما (one subtree) قبل الذهاب إلى الشجرة الفرعية التالية. ولذلك فإن البحث بالعمق أولا يُعدَ تعميما (generalization) للاجتياز سابق الترتيب للأشجار.

أما فى طريقة البحث بالعرض أولا فإننا نزور الرؤوس بترتيب المسافات المتزايدة (in order of increasing distance) من نقطة البداية v (starting point) مثلاً، حيث المسافة ببساطة هى عدد الأحرف فى أقصر مسار (number of edges in a shortest path). ونقصد بأقصر مسار من رأس v إلى رأس w يمكن الوصول إليه من v (reachable from) فى مخطط G: مساراً (a path) يحتوى على أقل عدد من الأحرف (smallest number of edges). أى أن الخوارزمية تكتشف (discovers) / تزور (visits) جميع الرؤوس التى تبعد مسافة تساوى d من الرأس v قبل اكتشاف / زيارة أى رؤوس تبعد مسافة تساوى d+1 من v. فالخطوة الأساسية/ المركزيه (central step) فى طريقة البحث بالعرض أولاً - ابتداءً بمسافة $d = 0$ وتكراريا (repeated) حتى لا نجد أى رؤوس جديدة - هى أن نأخذ فى الاعتبار بالترتيب (consider in turn) كل رأس "x" واقعة على بُعد d من رأس البداية v، وبفحص (examining) جميع الأحرف الواقعة على x (all edges incident with) ثم نجد ونزور / نشغِّل (find and process) جميع الرؤوس الواقعة على بُعد $d + 1$ من v.

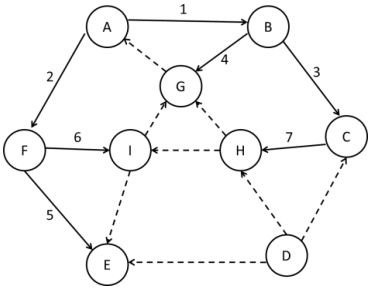
مثال 4-9: طبّق كلاً من خوارزمية البحث بالعرض أولا و خوارزمية البحث بالعمق أولا لاجتياز المخطط البياني الموجه G المعطى فى شكل 4-19، مبتدئاً عند الرأس A.



شكل 4-19

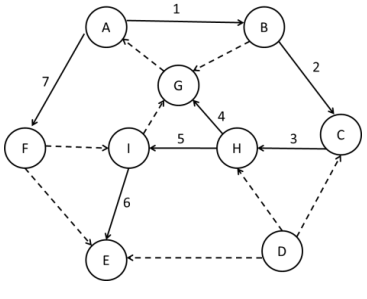
مخطط بياني موَجَّه A digraph G

الحـــل: الشكل التالى (شكل 4-20) يعطى نتيجة اجتياز المخطط G بتطبيق كل من خوارزميتي البحث بالعرض أولا والبحث بالعمق أولاً، وقد رُقّمت الأحرف فى الشكل بترتيب اجتيازها (edges are numbered in the order traversed).



(أ) البحث بالعرض أولاً BFS

Breadth-First search



(ب) البحث بالعمق أولاً DFS

Depth-First search

شكل 4-20

ونلاحظ من الشكل أن ترتيب زيارة الرؤوس (order in which vertices are visited) عند البحث بالعرض أولا (شكل 4-20- أ) هو: ABFCGEIH، بينما ترتيب زيارة الرؤوس عند البحث بالعمق أولاً (شكل 4-20- ب) هو: ABCHGIEF .

ملاحظة: رأينا سابقاً فى (تعريف 4- 12) أن أحرف أى مخطط بياني موجَّه G (a digraph) تصنف إلى أربع فئات بالبحث فى G بالعمق أولا. أما إذا كان G مخططاً بيانيا غير موجه (an undirected graph) فإن:

i) خوارزمية البحث بالعرض أولا BFS تُنتج (produces):

أحرف شجرة (tree edges)، وأحرفاً مستعرضة (cross edges) ولا تُنتج أى أحرف خلفية (back edges) أو أحرف أمامية (forward edges).

ii) خوارزمية البحث بالعمق أولا DFS تُنتج:

أحرف شجرة، وأحرفاً خلفية، ولا تُنتج أى أحرف مستعرضة أو أحرف أمامية.

مسائل الحلول المثلى للمخططات البيانية

Graph Optimization Problems

ندرس فيما يلى بإذن الله خوارزميتين لإيجاد شجرة مولِّدة صغرى MST (a minimum spanning tree) لمخطط بياني غير موجَّه (an undirected graph): خوارزمية "بريم" (Prim) وخوارزمية "كروسكال" (Kruskal)، وخوارزمية لإيجاد أقصر مسار ذى مصدر أحادى (single-source shortest path) فى مخططات بيانية موجهة وغير موجهة (directed and undirected graphs): خوارزمية "ديجسترا" (Dijkstra). وهذه الخوارزميات الثلاث تستخدم طابور أولوية (a priority queue) لاختيار (selecting) أفضل اختيار حالى (best current choice) من بين مجموعة اختيارات مُرشحه متاحة (a set of candidate choices).

أولاً : إيجاد الشجرة المُوَلِّدة الصغرى

1) خوارزمية "بريم" لإيجاد شجرة مولِّدة صغرى

Prim's Minimum Spanning Tree Algorithm

سندرس بإذن الله مسألة إيجاد شجرة مولِّدة صغرى لمخطط بياني غير موجَّه وموزن ومتصل (a connected, weighted, undirected graph). وبالنسبة للمخططات البيانية غير المتصلة (disconnected graphs) فإن الامتداد الطبيعى للمسألة هو إيجاد شجرة مولِّدة صغرى لكل مُرَكِّبَة متصلة (each connected component). ومن المعلوم أنه يمكن إيجاد المُرَكِّبَات المتصلة (connected components) فى زمن خطي (in linear time).

والأشجار المولّدة الصغرى لها معنى (meaningful) فقط بالنسبة للمخططات البيانية غير الموجهة ذوات أوزان للأحرف (with edge weights)، ولذلك فكل إشارة فيما يلى إلى"مخطط بياني" (graph) ستعني "مخططا بيانيا غير موجه" (undirected graph)، "والأوزان" (weights) ستعني دائما "أوزان أحرف" (edge weights). وتذكر أن الاصطلاح $G = (V, E, W)$ يعنى أن W هى دالة (a function) تسند (assigns) وزنا لكل حرف فى E. وهذا هو الوصف الرياضي. وأما فى التنفيذ (implementation) فلا توجد عادةً "دالة"، وإنما وزن كل حرف يُخزن (stored) ببساطة فى بنية معطيات هذا الحرف (the data structure for that edge).

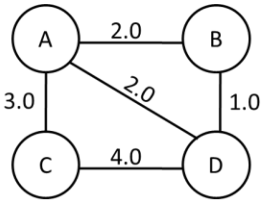
تعريف 4-13: **الشجرة المولِّدة الصغرى** Minimum Spanning Tree

نفرض أن $G = (V, E)$ مخطط بياني غير موجَّه ومتصل (a connected, undirected graph). يقال إن المخطط البياني G_1 هو **شجرة مولّدة (a spanning tree)** للمخطط G إذا كان G_1 مخططا بيانيا جزئيا من (a subgraph of) G عبارة عن شجرة غير موجَّهة (an undirected tree) تحتوى على جميع رؤوس G (all the vertices of). وفى أى مخطط مُوزَّن $G = (V, E, W)$ (a weighted graph) **وزن أى مخطط بياني جزئي w (the weight of a subgraph)** هو مجموع أوزان الأحرف فى هذا المخطط الجزئي. **والشجرة المولّدة الصغرى MST (a minimum spanning tree)** لمخطط بياني موزن هى شجرة مولّدة ذات أقل وزن (a spanning tree with minimum weight).

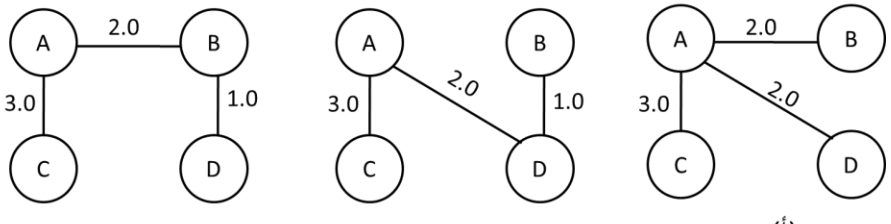
وهناك عدة تطبيقات عملية نحتاج فيها لإيجاد أشجار مولّدة صغرى. فمثلا لإيجاد أقل الطرق تكلفة (cheapest way) لتوصيل مجموعة من الوحدات الطرفية (to connect a set of terminals) اسواء كانت هذه الوحدات مُدُنا (cities) أو طرفيات كهربائية (electrical terminals) أو حواسيب

(computers) أومصانع (factories)؛ باستخدام طرق (roads) أو أسلاك (wires) أو خطوط تليفونات (telephone lines)، فإن الحل هو عبارة عن شجرة مولّدة صغرى (a MST) للمخطط البياني تحتوي على حرف (an edge) لكل توصيلة محتملة (for each possible connection) وزنه هو تكلفة (cost) هذه التوصيلة. وإيجاد أشجار مولّدة صغرى يُعد أيضا من المسائل الفرعية الهامة فى خوارزميات التوجيه (routing algorithms)، وهى الخوارزميات التى تبحث عن مسارات ذوات كفاءة عالية (efficient paths) عبر مخطط بياني بحيث تزور (visit) كل رأس (أو كل حرف) فى المخطط.

مثال 4-10: الشكل التالى (شكل 4-21) يبين مخططا بيانيا مُوزنا G، وثلاث اشجار مولّدة (أ)، (ب)، (جـ).



مخطط بيانى موزن G



شكل 4-21

ثلاث أشجار مولّدة للمخطط البياني المُوَزَّن G

ونلاحظ أن الشجرة المولّدة (أ) وزنها $w_1 = 7.0$ بينما الشجرتان المولدتان (ب)، (جـ) وزناهما $w_2 = w_3 = 6.0$. بمعنى أن كلا من الشجرة (ب) والشجرة (جـ) شجرة مولّدة صغرى. أى أن المخطط البياني الموزن يمكن أن يكون له أكثر من شجرة مولّدة صغرى واحدة.

تبدأ خوارزمية بريم باختيار رأس بداية اختيارية (arbitrary starting vertex)، ثم تتفرع (branches out) من جزء الشجرة الذى تم بناؤه / إنشاؤه حتى اللحظة باختيار رأس جديدة (a new vertex) وحرف عند كل تكرار (at each iteration). الحرف الجديد (the new edge) يصل الرأس الجديدة بالشجرة السابقة. وأثناء تنفيذ خطوات الخوارزمية يمكننا أن ننظر إلى الرؤوس (vertices) على أنها مقسمة (divided) إلى ثلاث فئات متباعدة (disjoint categories) كما يلى:

1) **رؤوس شجرة (tree vertices)**: وهى الرؤوس الموجودة فى الشجرة التى تم بناؤها حتى اللحظة (in the tree constructed so far).
2) **رؤوس هُدَّاب / حاشية / شراشيب / حافة (fringe vertices)**: وهى رؤوس ليست فى الشجرة، ولكنها مجاورة لرأس ما (adjacent to some vertex) فى الشجرة .
3) **رؤوس غير مرئية (unseen vertices)**: وهى جميع الرؤوس الأخرى (all other vertices).

والخطوة المفتاح (key step) فى الخوارزمية هى اختيار رأس من الحاشية/ الهُدَّاب (fringe) وحرف واقع (an incident edge) على الرأس. ونظراً لأن الأوزان (weights) موجودة على الأحرف ففعلياً (actually) يكون التركيز فى الاختيار على الحرف وليس على الرأس. وخوارزمية بريم تختار دائما حرفا ذا أقل وزن (an edge of minimum weight) من رأس شجرة (a tree vertex) إلى رأس هُدَّاب (a fringe vertex). وفيما يلى وَصف للخوارزمية:

الخوارزمية 4-2: **البنية العامة لخوارزمية "بريم"**

General structure of Prim's algorithm

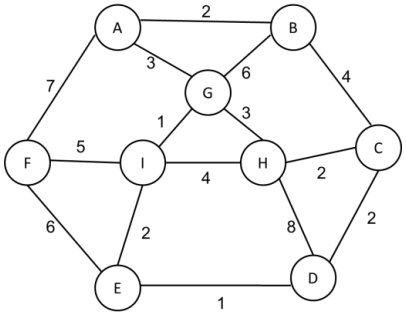
```
void primMST (G, n)          //OUTLINE
{
    Initialize all vertices as unseen.
    Select an arbitrary vertex s to start the tree; reclassify it as tree.
    Reclassify all vertices adjacent to s as fringe.
    while ( there are fringe vertices ) {
       Select an edge of minimum weight between a tree vertex t and
            a fringe vertex v;
       Reclassify v as tree; add the edge tv to the tree;
       Reclassify all unseen vertices adjacent to v as fringe.
    }
}
```

مثال 4-11: **تكرير واحد من خوارزمية "بريم"**

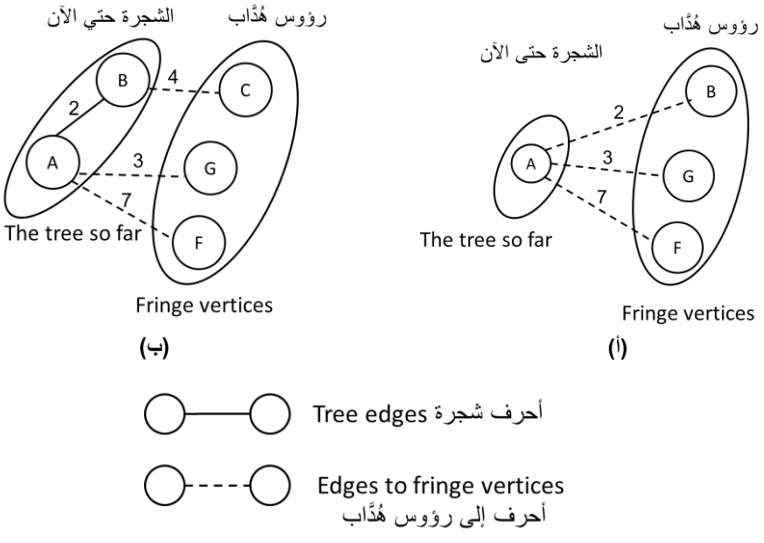
One iteration of Prim's algorithm

نفـرض أن لدينا المخطط البياني المـوزن المبين فى الشكل التالى (شـكل 4-22). ونفرض أننا سنختار الـرأس A لتكون رأس البدايـة (starting vertex). الـرؤوس المجـاورة للـرأس A (adjacent to) هى B, G, F، ولذلك فهى تشكل رؤوس الهُدَّاب (fringe vertices)، كما هو واضح فى الشكل (4-22- أ) الذى تؤدى إليه خطوات الخوارزمية المكتوبة قبل عروة while. وفى أول تكرير (iteration) لهذه العروة نجد أن الحرف AB هو الحرف ذو أقل وزن (minimum weight edge) إلى رأس هُدَّاب (to a fringe vertex). ولذلك فإن الرأس B يضاف إلى الشجرة، والرؤوس غير المرئية (the unseen vertices) المجاورة للرأس B (adjacent to) تدخل الهُدَّاب (the fringe)، فيؤدى ذلك إلى الشكل (4-22- ب). ويلاحظ أن

الحرف BG لم يظهرفى هذا الشكل نظراً لأن الحرف AG هو اختيار أفضل (a better choice) للوصول إلى الرأس G (to reach) لأن وزنه أقل. كما نلاحظ فى هذا الشكل أن الخطوط المتصلة (المصمتة) (solid lines) هى أحرف شجرة (tree edges) بينما الخطوط المتقطعة (dashed lines) هى أحرف إلى رؤوس هُدَّاب (fringe vertices).



مخطط بيانى موزَّن

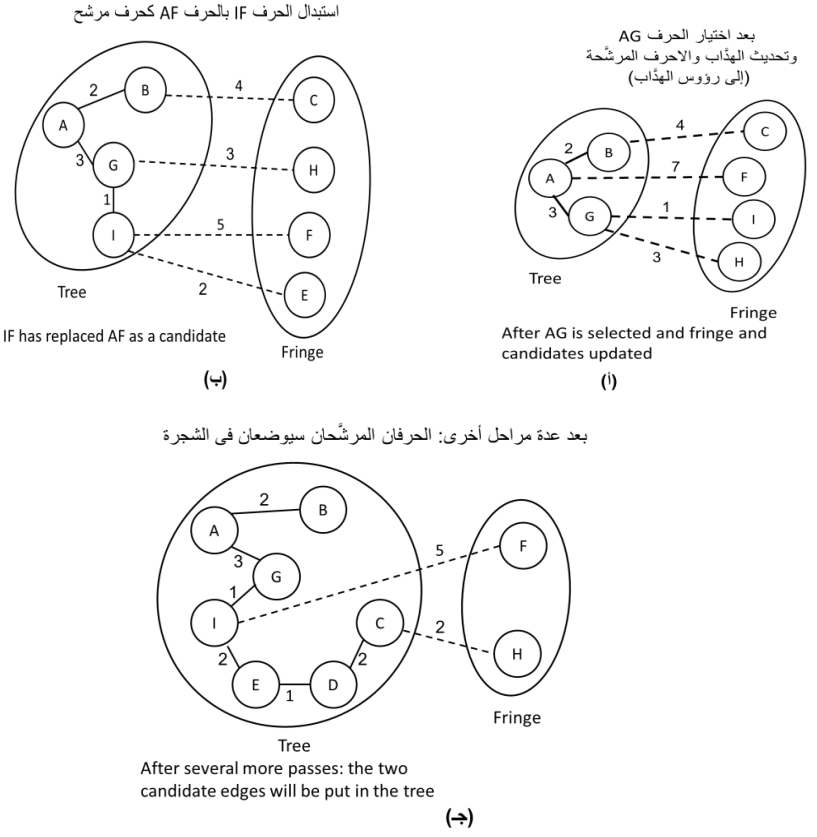


شكل 4-22

تنفيذ تكرير واحد من عروة خوارزمية "بريم"

مثال 4-12: **طبِّق خوارزمية "بريم" لإيجاد شجرة مولدة صغرى للمخطط البياني الموزن المبين فى شكل** 4-22.

الحـــل: **بعد تنفيذ التكرير الأول من خوارزمية "بريم" الموضح فى المثال السابق (مثال** 4-11) **نستكمل تطبيق خطوات التكريرات التالية فى الخوارزمية لنحصل على النتائج المبينة فى الشكل التالى (شكل**4-23).



شكل 4-23

مثال لخوارزمية "بريم" لإيجاد شجرة مولدة صغرى

يمكننا أن نلخص خطوات خوارزمية "بريم" وفكرتها الأساسية فيما يلى.

أخوارزميث 4-3: مخطط / شبث كود خوارزميث "بريم"

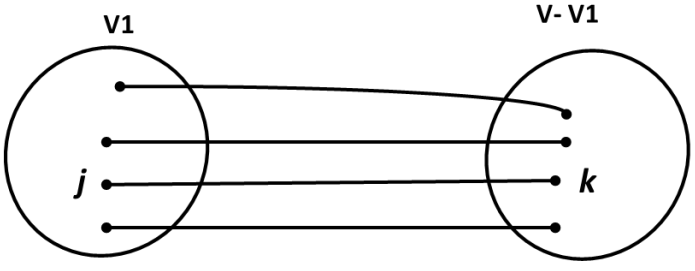
Prim's algorithm outline

المعطيات: نفرض أن $G = (V, E)$ مخطط بياني متصل موزن.

المطلوب: إيجاد شجرة مولّدة صغرى للمخطط G.

الفكرة الأساسية: نختار رأساً جديدة j (a new vertex) فى كل خطوة من خطوات الخوارزمية ونضيفها إلى مجموعة من الرؤوس V1 (a set of vertices)[انظر شكل 4-24]، ثم نختار حرفاً (k, j) (an edge) ذا أقل وزن (of minimum weight) بحيث أن $k \in V - V1$، ونضيف هذا الحرف إلى مجموعة من الأحرف T (a set of edges).

المخرجات: المجموعة T هى شجرة مولّدة صغرى (a minimum spanning tree) مخزونة (stored) كقائمة من الأحرف (a list of edges).



شكل 4-24

مخطط أخوارزميث algorithm outline

1- $V1 = \{1\};\ T = \Phi;$

2- **while** ($V \neq V1$) {

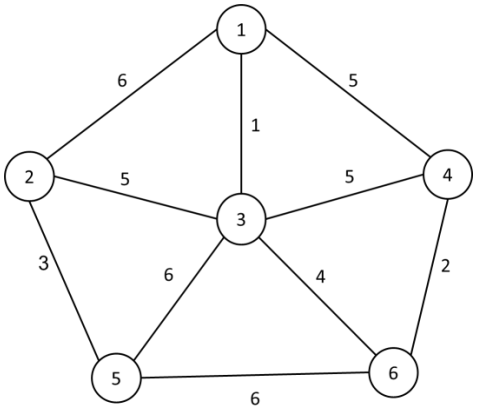
• let (k, j) be an edge of minimum weight such that $k \in V - V1$ & $j \in V1$

3- • $V1 = V1 \cup \{k\}$

4- • $T = T \cup \{(k, j)\}$

}

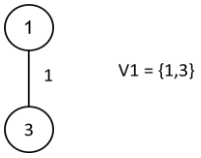
مثال 4-13: طبِّق خوارزمية "بريم" لإيجاد شجرة مولّدة صغرى للمخطط البياني المتصل المُوَزَّن G المبين فى شكل 4-25.



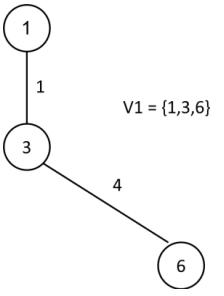
شكل 4-25

الحـــل:

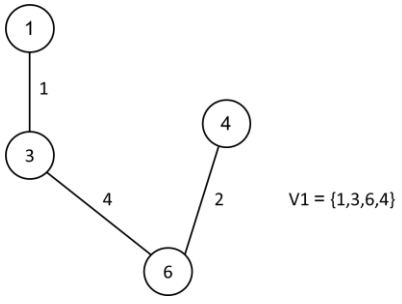
(i) سنختار الرأس 1 لتكون رأس البداية، أى أن $V1 = \{1\}$. ثم نختار الرأس 3 نظراً لأن الحرف (1, 3) هوالحرف ذو أقل وزن:
$3 \in V - V1$ & $1 \in V1$.



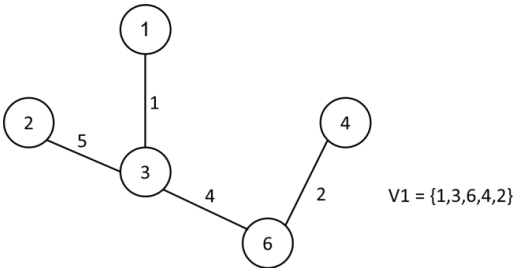
(ii) نختار الحرف (3, 6) ذا أقل وزن، بحيث أن $6 \in V - V1$ & $3 \in V1$.



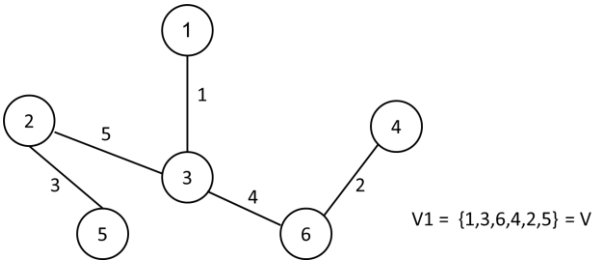
(iii) نختار الحرف (6, 4) ذا أقل وزن، بحيث أن $4 \in V - V1$ & $6 \in V1$.



(iv) نختار الحرف (3, 2) ذا أقل وزن، بحيث أن $2 \in V - V1$ & $3 \in V1$.



(v) نختار الحرف (2, 5).



شكل 4-26

تكلفة (cost) / وزن (weight) الشجرة المولّدة الصغرى T يساوى

$$\text{cost of } T = 1 + 4 + 2 + 5 + 3 = 15$$

صحة خوارزمية "بريم" لإيجاد شجرة مولّدة صغرى

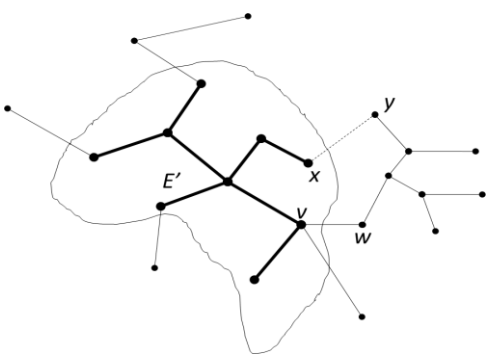
Correctness of Prim's MST Algorithm

خوارزمية "بريم" تُعدُ مثالا لما يطلق عليه خوارزمية "جَشِعَة / طَمّاعة / شَرِهَة" (a ''greedy'' algorithm). والخوارزميات الجشعة هى خوارزميات لمسائل الحلول المُثلى (optimization problems) [أى التى نود فيها أن تكون قيمةُ كميةٍ ما (some quantity) قيمةً صُغرى أو قيمةً كُبرى (minimized or maximized)]، وتقوم الخوارزمية بإجراء اختيارات مُثلى محليا (make locally optimal choices) فى كل خطوة من خطوات الخوارزمية على أمل أن تؤدى هذه الاختيارات إلى حل أمثل شامل (a global optimal solution). وهذه الطريقة تنجح فى كثير من الحالات ولكن ليس فى جميع المسائل. وخوارزمية "بريم" هى من الحالات التى تنجح فيها هذه الاستراتيجية، ويمكن برهنة ذلك بسهولة بالاستقراء (induction). ونظراً لأن الخوارزمية تبدأ بدون اختيار أى أحرف، فإن الأحرف التى سيتم اختيارها فى البداية (the edges selected initially) تُكوِّن (form) مجموعة جزئية من الأحرف الموجودة فى شجرة مولّدة صغرى ما. ثم نثبت أنه بعد إضافة حرف أصغر (a minimum edge) [أى حرف ذى أقل وزن] من الشجرة إلى رأس

هُدَّاب (a fringe vertex)، فإن مجموعة الأحرف المختارة تظل محتواة (contained) فى شجرة مولّدة صغرى.

نظرية 4-2: نفرض أن $G = (V, E, W)$ مخطط بياني متصل ومُوزن (a connected, weighted graph)، وأن E' - حيث $E' \subseteq E$ - مجموعة جزئية من الأحرف فى شجرة مولّدة صغرى ما $T = (V, E_T)$ للمخطط البياني G. ونفرض أن V' هى مجموعة الرؤوس الواقعة على الأحرف فى (incident with edges in) E'. إذا كان xy حرفا ذا أقل وزن بحيث أن $x \in V'$ & $y \notin V'$، فإن $E' \cup \{xy\}$ هى مجموعة جزئية من شجرة مولّدة صغرى.

البرهان: اذا كان الحرف xy موجوداً فى المجموعة E_T فإن المطلوب ينتج مباشرة. نفرض أن xy ليس فى E_T. يوجد مسار (a path) من x إلى y فى الشجرة T لأن الاشجار متصلة. نفرض أن vw هو أول حرف فى هذا المسار، حيث توجد رأس واحدة بالضبط (exactly one vertex) - من هذا الحرف- فى مجموعة الرؤوس V'، ولتكن هذه الرأس هى v. [انظر الشكل 4- 27].



شكل 4- 27

شجرة مولّدة صغرى T لنظرية 4-2

الأحرف المبينة كخطوط مصمتة (solid lines) تقع فى الشجرة T

الأحرف السميكة (heavy edges) تقع أيضا فى المجموعة E'

قد تكون هناك أحرف كثيرة أخرى فى المخطط لا تظهر فى الشكل

نفرض أن: $E_{T'} = E_T - \{vw\} \cup \{xy\}$. يمكن للقارئ الكريم أن يتحقق من أن $E' \cup \{xy\} \subseteq E_{T'}$ وأن T' [أى $(V, E_{T'})$] شجرة مولّدة. ونظراً لأن v تقع فى V' و w ليست فى V' [باختيارنا الحرف xy]، فلذلك $W(xy) \le W(vw)$، وبالتالى فإن $W(T') \le W(T)$، و T' هى شجرة مولّدة صغرى، ويتم إثبات النظرية.

❖ ❖ ❖

نلاحظ أنه بعد كل تكرير من عروة الخوارزمية قد تكون هناك رؤوس هُدَّاب جديدة، ومجموعة الأحرف التى من بينها يتم الاختيار التالى ستتغير. ومن شكل (4-22- ب) نرى أننا لا نحتاج لأن نأخذ فى الاعتبار جميع الأحرف بين رؤوس الشجرة ورؤوس الهُدَّاب. بعد اختيار الحرف AB، أصبح الحرف BG مُرشَّحاً للاختيار، ولكننا نتجاهله لأن الحرف AG وزنه أقل، وبالتالى يصبح اختيارا أفضل للوصول إلى الرأس G. وإذا كان وزن BG أقل من وزن AG لتجاهلنا AG. ولكل رأس هُدَّاب نحتاج لأن نتتبع حرفاً واحدا فقط إليها من الشجرة، وهو الحرف ذو أقل وزن. وسنطلق على هذه الأحرف "الأحرف المرشحه" (candidate edges).

ويمكننا الآن أن نُمَدِّد / نُوَسِّع مخطط الخوارزمية بما يأخذ فى الاعتبار الملاحظات السابقة.

الخوارزمية4 -4: المخطط (المُوسَّع) لخوارزمية "بريم"

Prim's algorithm (expanded) outline

Input: G = (V, E, W), a weighted graph.

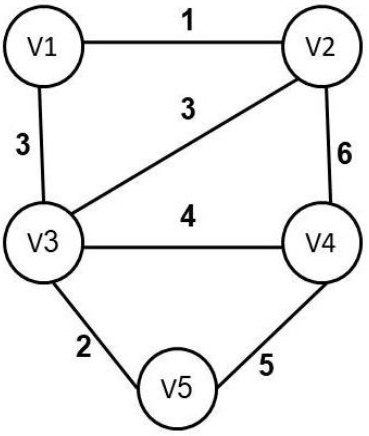
Output: the edges in a minimum spanning tree.

```
1-  // initialization
      Let x be an arbitrary vertex.
      VT = {x};  ET = φ;
      stuck = false;
2-  // Main loop; x has just been brought into the tree.
    // Update fringe and candidates.Then add one vertex and edge.
      while ( VT ≠ V and not stuck ) {
3-  // replace some candidate edges.
      for  ( each fringe vertex y adjacent to x )
           if ( W(xy) < W(the candidate edge e incident with y))
          xy replaces e as the candidate edge for y;
4-  // find new fringe vertices and candidate edges.
      for ( each unseen y adjacent to x ) {
           y is now a fringe vertex;
           xy is now a candidate;
           } // end for
5-  // ready to choose next edge.
           if ( there are no candidates )
           stuck = true;  //  no spanning tree
            else {
6-  // choose next edge.
      Find a candidate edge, e, with minimum weight;
      x = the fringe vertex incident with e;
      Add x and e to the tree;
           // x and e are no longer fringe and candidate.
   }// end if
}// end while
```

ينتهى تنفيذ الخوارزمية عندما $V_T = V$، والأحرف فى المجموعة E_T تُكوِّن شجرة مولدة صغرى إذا وفقط إذا كان المخطط البياني G متصلا (connected). وإذا انتهى تنفيذ الخوارزمية لنقص فى الأحرف المرشحة (for lack of candidate edges) - أى أن stuck = true - فإن الأحرف فى المجموعة E_T تُكَوِّن شجرة مولّدة صغرى لمُرَكِّبَة متصلة فى المخطط G (a connected component of). وبتعديل بسيط يمكننا "إعادة بدء" إنشاء الشجرة ("restarting" the tree construction) لإيجاد غابة (a forest) من أشجار مولدة صغرى، واحدة لكل مُرَكِّبَة متصلة فى المخطط G. ويوضح مثال 4-12 (شكل 4-22، وشكل 4-23) مثالا لتطبيق هذه الخوارزمية.

❖ ❖ ❖

رأينا أن خوارزمية "بريم" تبدأ بمجموعة جزئية خاوية من الأحرف E_T (an empty subset of edges)، ومجموعة جزئية من الرؤوس V_T تبدأ برأس اختيارية $\{x\}$. ونقول إن رأسا تعد "الأقرب" ("nearest") للمجموعة V_T هى رأس فى المجموعة V-V_T مجاورة (adjacent) لرأس فى V_T بحرف ذى أقل وزن. [تذكَّر أننا نستخدم تبادليا (interchangeably) المصطلحين: الوزن والمسافة (weight and distance) بالنسبة للمخططات البيانية الموزنة.] ففى الشكل التالى (شكل 4- 28) الرأس v_2 تعد الأقرب إلى V_T عندما تكون $V_T = \{v_1\}$. و الرأس الأقرب إلى V_T تضاف إلى V_T والحرف يضاف إلى E_T.



شكل 4-28

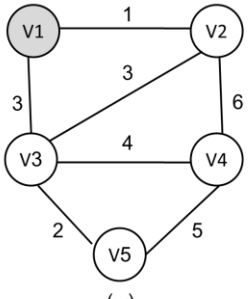
مخطط بياني متصل موزَّن غير موجَّه G

A connected weighted undirected graph G

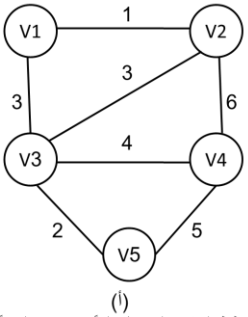
وفى حالة تساوى أوزان الأحرف يكون ترتيب اختبارها اختياريا (ties are broken arbitrarily). وفى هذه الحالة v_2 تضاف إلى V_T، و (v_1, v_2) يضاف إلى E_T. وتستمر عملية إضافة الرؤوس الأقرب إلى أن يصبح $V_T = V$. ويتم إجراء اختيار الرؤوس (selection procedure) والتحقق من شرط الأقرب (feasibility check) معاً لأن أَخْذَ الرأس الجديدة من المجموعة $V - V_T$ يضمن عدم إنشاء دورة (a cycle is not created). وفى كل خطوة فى الشكل التالى (شكل4- 29) الذى يوضح تطبيق خوارزمية "بريم" المجموعة V_T تحتوى على الرؤوس المظللة، و المجموعة E_T تحتوى على الأحرف المزدوجة.

مثال 4-14: طبِّق خوارزمية "بريم" لإيجاد شجرة مولّدة صغرى للمخطط البياني المتصل الموزن غير الموجه G المبين فى شكل 4-28.

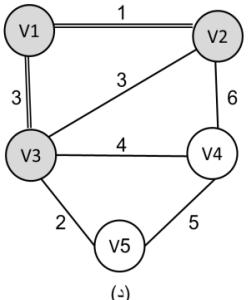
الحـــل:



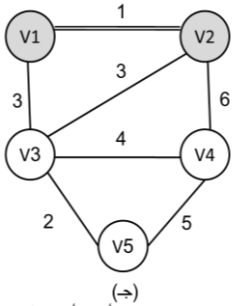
(ب)
الرأس V1 يتم اختيارها أولاً



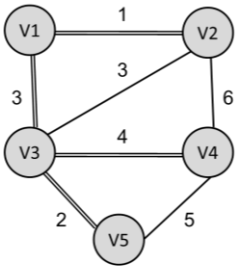
(أ)
المخطط البياني المتصل المؤَّزن غير الموجَّه G



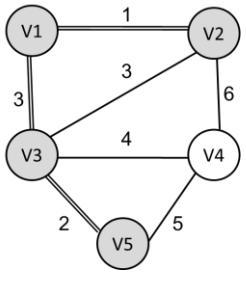
(د)
الرأس V3 يتم اختيارها لأنها الاقرب إلى {V1,V2}



(جـ)
الرأس V2 يتم اختيارها لأنها الأقرب إلى {V1}



(و)
الرأس V4 يتم اختيارها



(هـ)
الرأس V5 يتم اختيارها لأنها الأقرب إلى {V1,V2,V3}

شكل 4-29

خطوات تطبيق خوارزمية "بريم" على المخطط البياني G (شكل 4-28)

ولكتابة خوارزمية يمكن تنفيذها (implemented) بلغة من لغات الحاسوب نحتاج إلى وصف إجراء (a procedure) خطوةً خطوة. ومن أجل ذلك سنمثل أى مخطط بياني موزن بمصفوفة التجاور (its adjacency matrix) أى أننا سنمثله بمصفوفة أعداد W ثنائية البعد n×n بحيث أن

$$W[i][j] = \begin{cases} weight\ on\ edge & \text{إذا كان هناك حرف بين } v_i \text{ و } v_j \\ \infty & \text{إذا لم يكن هناك حرف بين } v_i \text{ و } v_j \\ 0 & \text{إذا كانت } i = j \end{cases}$$

فمثلا المخطط البياني G الذى يظهر فى شكل 4- 28 يمثل بمصفوفة التجاور W التى تظهر فى الشكل التالى (شكل 4- 30).

$$\begin{pmatrix} 0 & 1 & 3 & \infty & \infty \\ 1 & 0 & 3 & 6 & \infty \\ 3 & 3 & 0 & 4 & 2 \\ \infty & 6 & 4 & 0 & 5 \\ \infty & \infty & 2 & 5 & 0 \end{pmatrix}$$

(شكل 4-30)

تمثيل المخطط G (شكل 4–28) بمنظومة W

وسنرمز لمجموعة الرؤوس V_T بالرمز Y، ولمجموعة الأحرف E_T بالرمز F . وسنحتفظ بمنظومتين "nearest" و"distance"، حيث: للقيم $i = 2, 3, \dots, n$

nearest[i] = v_i مؤشر (index) الرأس الموجودة فى Y والتى تعد الأقرب إلى الرأس

distance[i] = وزن الحرف بين الرأس v_i والرأس التى مؤشرها nearest[i]

ونظراً لأنه فى البداية تكون $Y = \{v_1\}$ فلذلك يُعطَى المؤشر nearest[i] القيمة الابتدائية 1، وتكون القيمة الابتدائية للوزن distance[i] هى وزن الحرف الواصل بين الرأس v_1 والرأس v_i. ومع إضافة الرؤوس إلى Y فإنه يتم تحديث (updating) المنظومتين distance و nearest للإشارة إلى (reference) الرأس الجديدة فى Y الأقرب إلى كل رأس خارج Y. ولتحديد الرأس التى يتم إضافتها إلى Y، ففى كل تكرير (iteration) نحسب المؤشر (index) الذى يكون عنده الوزن distance[i] أقل قيمة (the smallest)، وسنطلق على هذا المؤشر "vnear". والرأس التى مؤشرها vnear (indexed by) ستضاف إلى Y بجعل distance[vnear] تساوى 1-. والخوارزمية التالية تقوم بتنفيذ (implementing) هذا الإجراء.

الخوارزمية 4-5: **إجراء "بريم" لإيجاد شجرة مولدة صغرى**

Prim's Minimum Spanning Tree procedure

المدخلات: عدد صحيح $n \geq 2$، ومخطط بياني متصل موزن غير موجه عدد رؤوسه n. والمخطط تمثله منظومة W ثنائية البعد، مؤشرات كل من صفوفها وأعمدتها تقع فى المدى من 1 إلى n، حيث W[i][j] هو وزن الحرف الذى يصل بين الرأس رقم i والرأس رقم j.

المخرجات: مجموعة أحرف F فى شجرة مولّدة صغرى للمخطط البياني.

الإجـــراء:

```
void prim ( int n, const number W[ ][ ], set_of_ edges & F )
{
    index i , vnear;
    number min;
    edge e;
    index nearest [2..n];
    number distance [2..n];

    F = φ ;
    for (i = 2; i <= n; i+ +)  {
       nearest [i]=1              // for all vertices, initialize v1 to
       distance[i] = W[1][i];     // be the nearest vertex in Y and
    }                             // initialize the distance from Y to
                                  // be the weight on the edge to
    repeat ( n - 1 times )  {
                                   // add all n-1 vertices to Y.
       min = ∞;
       for ( i = 2; i <= n; i++)              // check each vertex for
          if ( 0 ≤ distance [i] <  min ) {  // being nearest to Y.
             min = distance [i];
             vnear = i ;
          }
       e = edge connecting vertices indexed by vnear and
       nearest[vnear];
       add e to F;
       distance [vnear] = -1;              // add vertex indexed by
       for ( i = 2; i <= n; i++)            // vnear to Y.
          if ( W[i][vnear] < distance[i] ) { // for each vertex not in Y,
             distance[i] = W[i] [vnear]; // update its   distance from Y.
             nearest[i] = vnear;
          }
    }
}
```

درجة التعقيد الزمنية لإجراء "بريم"

Time Complexity of Prim's procedure

العملية الأساسية (Basic operation): نلاحظ أنه توجد عروتان (two loops) منفصلتان تحتوى كل منهما على تكريرات (iterations) عددها $(n-1)$ وذلك داخل عروة (repeat). ويمكن اعتبار أن تنفيذ الأوامر/ التعليمات (executing the instructions) داخل كل منهما يعد تنفيذاً للعملية الأساسية مرة واحدة (doing the basic operation once).

حجم المدخلات (input size): n، وهو عدد الرؤوس.

ونظراً لأن عروة repeat تشتمل على تكريرات عددها $(n-1)$، فلذلك تُعطَى درجة التعقيد الزمنية (time complexity) بالعلاقة

$$T(n) = 2(n-1)(n-1) \in \Theta(n^2)$$

متطلبات التخزين لإجراء "بريم"

Storage requirement of Prim's Procedure

يتطلب إجراء "بريم" تخزين عدد 2n من عناصر المنظومتين nearest, distance بالإضافة إلى تخزين المخطط G. وهذا يعد قدراً بسيطاً من التخزين الإضافى (extra storage).

2) **خوارزمية "كروسكال" لإيجاد شجرة مولدة صغرى**

Kruskal's Minimum Spanning Tree Algorithm

نفرض أن $G = (V, E, W)$ مخطط بياني موزن غير موجه. درسنا فيما سبق خوارزمية "بريم" لإيجاد شجرة مولدة صغرى للمخطط G [بشرط أن يكون المخطط متصلا (connected)]. وقد بدأت الخوارزمية عند رأس اختيارية ثم تفرعت للخارج (branched out) من هذه الرأس باختيار أحرف ذوات أوزان

صغيرة "بطريقة جشعة" (greedy). وفى أى وقت فإن الأحرف المختارة (chosen edges) كونت شجرة (formed a tree). وفيما يلى ندرس خوارزمية تستخدم استراتيجية أكثر جشعاً (a greedier strategy). وفيما يلى ستكون كل المخططات غير موجهة.

الخوارزمية 4-6: البنية العامة لخوارزمية "كروسكال"

General structure of Kruskal's algorithm

الفكرة العامة فى خوارزمية "كروسكال" أنها فى كل خطوة تختار الحرف المتبقى ذا أقل وزن (lowest–weighted remaining edge) من أى موضع فى المخطط البياني، ولكنها تجتنب أى حرف سيؤدى اختياره إلى تكوين دورة (would form a cycle) مع الأحرف التى تم اختيارها فعلاً (already chosen). وفى أى وقت فإن الأحرف التى تم اختيارها حتى اللحظة ستُكَوِّن غابة (will form a forest) ولكن ليس بالضرورة شجرة واحدة. وينتهى تنفيذ الخوارزمية عندما يتم تشغيل (processing) جميع الأحرف.

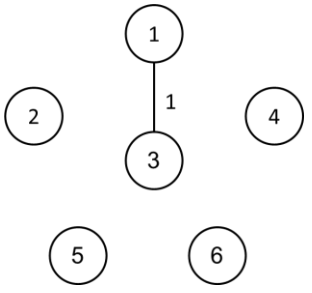
```
void kruskalMST (G, n)     // OUTLINE
{
      R = E;       // R is remaining edges.
      F = φ;       // F is forest edges.
      while ( R is not empty ) {
            Remove the lowest–weighted/shortest edge, vw, from R;
            if ( vw does not make a cycle in F )
                Add vw to F;
   }
}
```

مثال 4-15: طبِّق خوارزمية "كروسكال" لإيجاد شجرة مولّدة صغرى للمخطط البياني المتصل الموزن G المبين فى شكل 4-25.

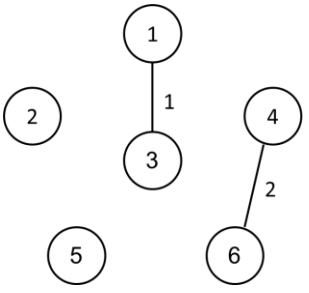
الحـــل: سنرتب (sort) أولاً أحرف المخطط البياني G تصاعدياً (in ascending order) حسب أوزانها (weights)/ أطوالها (lengths)/ تكلفتها (costs).

edge	(1,3)	(4,6)	(2,5)	(3,6)	(1,4)	(3,4)	(2,3)	(1,2)	(3,5)	(5,6)
weight	1	2	3	4	5	5	5	6	6	6

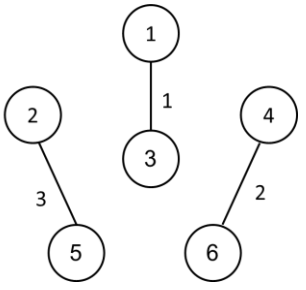
(i) نختار الحرف (1,3) ذا أقل وزن.



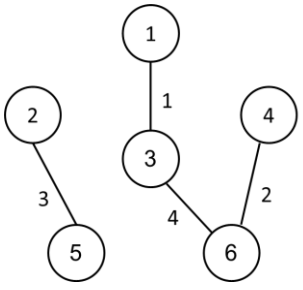
(ii) نختار الحرف المتبقي (4,6) ذا أقل وزن.



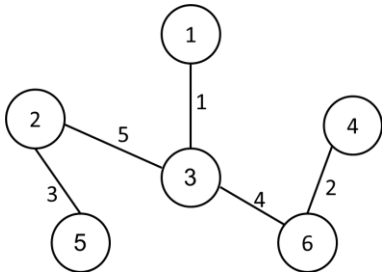
(iii) نختار الحرف المتبقي (2,5) ذا أقل وزن



(iv) نختار الحرف المتبقي (3,6) ذا أقل وزن.



(v) إذا اخترنا الحرف المتبقي (1,4) ذا أقل وزن فإنه سيُكوِّن دورة، ولذلك نجتنبه. وكذلك الحرف (3,4) نجتنبه للسبب نفسه. ونختار الحرف المتبقي (2,3) ذا أقل وزن.



(vi) الأحرف المتبقية جميعها تؤدي إلى تكوين دورة، ولذلك نجتنبها، وينتهي تنفيذ الخوارزمية.

شكل 4-31

خطوات تطبيق خوارزمية "كروسكال" على المخطط البياني G (شكل 4-25)

نلاحظ أن وزن المخطط البياني G يساوي $W(G) = 43$، ووزن الغابة الناتجة F يساوى $W(F) = 15$. كما نلاحظ أن الغابة الناتجة فى مثالنا هذا عبارة عن شجرة من 6 رؤوس و 5 أحرف. وعموما أى شجرة من n رأس و $(n - 1)$ حرف إذا أضيف لها أى حرف فإنه سيؤدي إلى تكوين دورة (cycle).

وعموما تبدأ خوارزمية "كروسكال" لإيجاد شجرة مولّدة صغرى بإنشاء مجموعات جزئية متباعدة من مجموعة الرؤوس V (disjoint subsets of)، واحدة لكل رأس (one for each vertex) وتحتوى المجموعة الجزئية على هذه الرأس فقط. ثم تختبر الخوارزمية الأحرف تباعاً بناءً على أوزانها غير التناقصية (nondecreasing weights)، وبالنسبة للأحرف متساوية الاوزان (ties) يكون ترتيب اختبارها اختياريا (arbitrary). وإذا كان لدينا أى حرف يقع على (incident upon) رأسين فى مجموعتين جزئيتين متباعدتين، فإن هذا الحرف يُضاف إلى مجموعة الأحرف F، وهاتين المجموعتين الجزئيتين تُدمجان (merged) فى مجموعة واحدة. ويستمر تكرير (repeating) هذه العملية (process) حتى يتم دمج جميع المجموعات الجزئية فى مجموعة واحدة. والخوارزمية التالية عالية المستوى (high level algorithm) توضح هذه الخطوات.

الخوارزمية 4-7: **خوارزمية "كروسكال" لإيجاد شجرة مولّدة صغرى**
Kruskal's Minimum Spanning Tree algorithm

```
F = φ;                          // initialize set of edges to empty
create disjoint subsets of  V, one for each vertex
and containing only that vertex;
sort the edges in E in nondecreasing order;
 while ( the instance is not solved ) {
    select next edge ;              // selection procedure
    if ( the edge connects two vertices in disjoint subsets ) {
                                          // feasibility check
            merge the subsets;
            add the edge to F;
     } // end if
     if ( all the subsets are merged )      // solution check
            the instance is solved
} // end while
```

ويوضح المثال التالى (مثال 4–16) تطبيق خوارزمية "كروسكال".

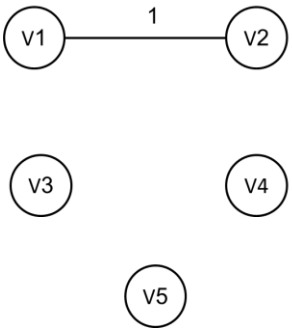
مثال 4-16: طبِّق خوارزمية "كروسكال" لإيجاد شجرة مولّدة صغرى للمخطط البياني الموزن G المبين فى شكل 4- 28.

الحـــل:

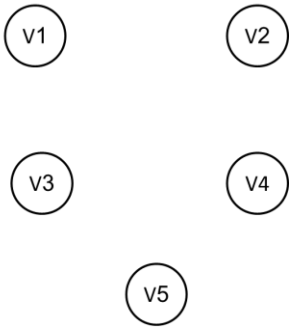
(1) نبدأ بترتيب أحرف المخطط G ترتيباً غير تنازلي بناءً على أوزانها.

edge	(v1,v2)	(v3,v5)	(v1,v3)	(v2,v3)	(v3,v4)	(v4,v5)	(v2,v4)
weight	1	2	3	3	4	5	6

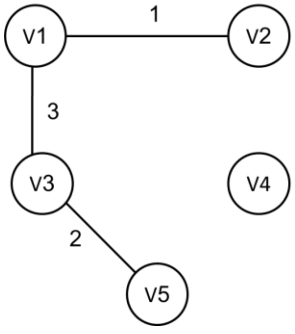
(2) ننشئ المجموعات الجزئية المتباعدة (3) نختار الحرف (v1,v2)

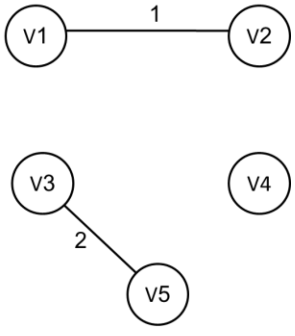


(5) نختار الحرف (v1, v3)

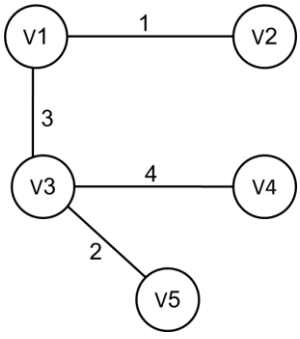


(4) نختار الحرف (v3, v5)





(6) نختار الحرف (v3, v4)



شكل 4-32

خطوات تطبيق خوارزمية "كروسكال" على المخطط البياني G (شكل 4-28)

ولكتابة صيغة رسمية (a formal version) تنفيذية لخوارزمية "كروسكال" نحتاج إلى نوع بيانات مجرد للمجموعات المتباعدة (disjoint sets abstract data type). وإذا كان تنفيذ (implementation) هذا النوع من البيانات هو لمجموعات جزئية متباعدة من المؤشرات (for disjoint subsets of indices)، فإننا نحتاج فقط للإشارة (refer) إلى الرؤوس بالمؤشرات (by index) لاستخدام التنفيذ. ويتكون نوع البيانات المجرد للمجموعات المتباعدة من نَوْعَى البيانات index, set_pointer والبرامج الروتينية (الفرعية) initial, find, merge, equal بحيث أنه إذا أعلنا (declared):

```
index i;
set_ pointer p, q;
```

فـإن:

- initial (n): يُعطِى قيماً ابتدائية (initializes) لمجموعات جزئية متباعدة عددها n، ويحتوى كل منها بالضبط (exactly) على واحد من المؤشرات (indices) بين 1 و n.
- p = find (i): يجعل المؤشر p (pointer) يشير إلى (point to) المجموعة التى تحتوى على المؤشر i (index).
- merge (p, q): يدمج المجموعتين اللتين يشير إليهما p, q فى مجموعة واحدة.
- equal (p, q): يعيد القيمة true إذا كان المؤشران p, q يشيران معاً إلى المجموعة نفسها (same set).

الخوارزمية 4-8: **خوارزمية "كروسكال" الرسمية/التنفيذية لإيجاد شجرة مولّدة صغرى**

Formal Kruskal's Minimum Spanning Tree algorithm

المدخلات: عدد صحيح $n \geq 2$، وعدد صحيح موجب m، ومخطط بياني متصل موزن غير موجه يحتوى على رؤوس عددها n وأحرف عددها m. والمخطط يُمثَّل (represented) بمجموعة أحرف E تحتوى على أحرف المخطط مع أوزانها.

المخرجات: مجموعة أحرف F فى شجرة مولّدة صغرى.

```
void kruskal ( int n, int m, set_of_edges E, set_of_edges& F)
{
    index i, j;
    set_pointer p,q;
    edge e;

    sort the m edges in E by weight in nondecreasing order;
    F = φ;
    initial (n);                    // initialize n disjoint subsets.
    while ( number of edges in F is less than n-1 ) {
        e= edge with least weight not yet considered ;
        i, j = indices of vertices incident upon e ;
        p = find(i) ;
        q = find(j) ;
        if ( ! equal ( p, q ) ) {
            merge ( p, q);
            add e to F;
        } // end if
    } // end while
}
```

يلاحظ أن عروة while يتم الخروج منها عندما يصل عدد الأحرف فى المجموعة F إلى $n-1$، وذلك لأن عدد أحرف أى شجرة مولدة هو $n-1$.

درجة التعقيد الزمنية لخوارزمية "كروسكال" فى أسوأ حالة
Kruskal's Algorithm Worst-Case Time Complexity

العملية الاساسية: تعليمة مقارنة (a comparison instruction).

حجم المدخلات: n وهو عدد الرؤوس، و m هو عدد الأحرف.

هناك ثلاث نقاط فى هذه الخوارزمية نأخذها فى الاعتبار:

i) **الزمن اللازم لترتيب الأحرف (sorting the edges)**. وقد حصلنا فى الفصل السابق على خوارزمية ترتيب (خوارزمية الترتيب بالدمج Mergesort) درجة تعقيدها فى أسوأ حالة $\Theta(m\lg m)$. كما رأينا أيضا فى الفصل السابق أنه لا يمكن تحسين هذا الأداء (performance) بالنسبة للخوارزميات التى تجرى الترتيب (sorting) عن طريق المقارنة بين المفاتيح (comparison of keys). ولذلك فإن درجة التعقيد الزمنية لترتيب الأحرف تعطى بالعلاقة

$$W(m) \in \Theta(m\lg m)$$

ii) **الزمن المستغرق فى عروة (while)**. الزمن اللازم لمعالجة المجموعات المتباعدة هو الغالب (dominant) فى هذة العروة [لأن كل ما عدا ذلك هو ثابت (constant)]. فى أسوأ حالة يتم اخذ جميع الأحرف فى الاعتبار قبل الخروج من عروة (while)، وهذا يعنى أن عدد مرات اجتياز/ مرور عبر العروة m (passes).وباستخدام تنفيذ بنية بيانات المجموعات المتباعدة فإن درجة التعقيد الزمنية لعدد m من الاستدعاءات (calls) للبرنامجين

الفرعيين find, equal، وعدد $n-1$ من الاستدعاءات للبرنامج الفرعي merge يُعطَى بالعلاقة

$$W(m,n) \in \Theta(m + n\lg n)$$

حيث العملية الأساسية هى تعليمة مقارنة.

iii) **الزمن اللازم لإعطاء قيم ابتدائية (initialize) لمجموعات متباعدة عددها n.** باستخدام تنفيذ بنية بيانات المجموعات المتباعدة، فإن درجة التعقيد الزمنية لهذه البُداءة (initialization) تُعطَى بالعلاقة

$$T(n) \in \Theta(n)$$

ونظراً لأن $m \geq n-1$ فإن عملية الترتيب (sorting) ومعالجات (manipulation) المجموعات المتباعدة تغلب (dominate) زمن البُداءة (initialization)، مما يعنى أن

$$W(m,n) \in \Theta(m + n\lg n + m\lg m) = \Theta(m\lg m)$$

قد يبدو أن أسوأ حالة ليس لها أى اعتماد على n. إلا أنه فى أسوأ حالة فإن كل رأس (vertex) يمكن أن تكون متصلة بكل رأس أخرى، مما يعنى أن

$$m = \frac{n(n-1)}{2} \in \Theta(n^2)$$

ولهذا فإنه يمكننا أيضا أن نكتب اسوأ حالة كما يلى:

$$W(m,n) \in \Theta(n^2 \lg n^2) = \Theta(n^2 2\lg n) = \Theta(n^2 \lg n)$$

ومن المفيد أن نستخدم التعبيرين لأسوأ حالة حين المقارنة بين خوارزمية "كروسكال" و خوارزمية "بريم".

مقارنة بين خوارزمية "كروسكال" و خوارزمية "بريم" .

حصلنا على درجتى التعقيد الزمنيتين التاليتين:

خوارزمية "بريم": $T(n) \in \Theta(n^2)$

خوارزمية "كروسكال": $W(m,n) \in \Theta(m \lg m)$

وفى أى مخطط بياني متصل تتحقق دائما العلاقة

$$n-1 \le m \le \frac{n(n-1)}{2}$$

وبالتالي فبالنسبة لأى مخطط عدد أحرفه m قريب من الحد الأدنى [أى أن المخطط متناثر/متفرق/هش جدا (very sparse)، أى أن $m \cong c\,n$] فإن درجة تعقيد خوارزمية "كروسكال" تكون $\Theta(n \lg n)$ ، مما يعنى أنها تصبح أسرع من خوارزمية "بريم". وأما إن كان عدد أحرف المخطط قريبا من الحد الأعلى [أى أن المخطط متصل بدرجة عاليه (highly connected)، أى أن $m \cong c' n^2$] فإن درجة تعقيد خوارزمية "كروسكال" تكون $\Theta(n^2 \lg n)$ ، مما يعنى أن خوارزمية"بريم" تصبح أسرع.

ثانياً: أقصر مسارات من مصدر أحادى Single – Source Shortest Paths

فى كثير من مسائل التطبيقات العملية للمخططات البيانية نحتاج لإيجاد المسارات ذوات أقل وزن (minimum–weight path) من رأس مصدر معين (a specified source vertex) إلى كل رأس أخرى فى مخطط بياني موزن موجه أوغير موجه، ووزن [أو طول (length) او كلفة (cost)] أى مسار هو مجموع أوزان الأحرف فى هذا المسار.

فإذا كان لدينا المخطط البياني: $G = (V, E)$

ودالة الوزن: weight function $w : E \to R$

فإن وزن المسار $\textbf{weight of path } p = \langle v_0, v_1, \ldots, v_k \rangle$

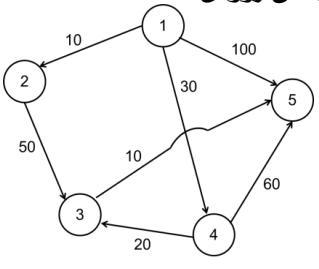
$$= \sum_{i=1}^{k} w\,(v_{i-1}, v_i)$$

وإذا فُسِّر الوزن على أنه المسافة، فإن مساراً ذا أقل وزن يطلق عليه "أقصر مسار" (a shortest path)، وهذا هو الاسم المستخدم غالباً. وإذا لم يوجد مسار من رأس لرأس أخرى فإننا فرضاً نقول إن وزن هذا المسار يساوى ∞، وعليه فإننا نُعَرِّف وزن أقصر مسار من الرأس u إلى الرأس v (shortest–path weight u to v) بأنه:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{if there exists a path } u \to v \\ \infty & \text{otherwise} \end{cases}$$

إذا وُجِدَ مسار
ما عدا ذلك

مثال 4-17: فى المخطط البياني التالى (شكل 4-33) نفرض أن الرأس 1 هى المصدر (source). أوجد أوزان أقصر مسارات من هذا المصدر إلى كل رأس أخرى من رؤوس المخطط.



شكل4-33

الحـــل:

Paths	shortest path	Weight
1 ⇝ 2	1,2	10
1 ⇝ 3	1,4,3	50
1 ⇝ 4	1,4	30
1 ⇝ 5	1,4,3,5	60

خوارزمية "ديجسترا" لإيجاد أقصر مسارات من مصدر أحادى

Dijkstra's Algorithm for Single-Source Shortest Paths

نستخدم فيما يلى الأسلوب الجشع (the greedy approach) للوصول إلى خوارزمية درجة تعقيدها $\Theta(n^2)$ لمسألة إيجاد أقصر مسارات من مصدر أحادى. وسوف نعرض أولا الخوارزمية بفرض أنه يوجد مسار من الرأس المصدر إلى كل من الرؤوس الأخرى. وإذا لم يتحقق هذا الفرض فيمكن بتعديل بسيط معالجة المسألة.

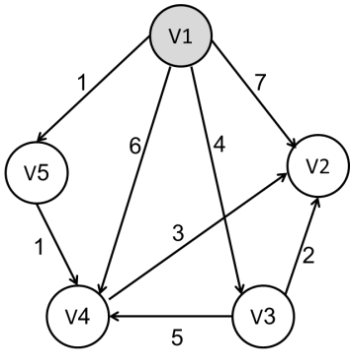
وهذه الخوارزمية شبيهة بخوارزمية "بريم" لإيجاد شجرة مولدة صغرى، حيث نبدأ بمجموعة رؤوس Y تحتوى ابتداءً على الرأس المصدر – ولتكن v_1 - فقط. ونستخدم مجموعة أحرف F تكون ابتداءً فارغة (empty). ثم نبدأ باختيار رأس v تكون أقرب رأس إلى المصدر v_1، ونضيف v إلى المجموعة Y، ونضيف الحرف $<v_1, v>$ إلي المجموعة F. [ونعنى بالمصطلح $<v_1, v>$ الحرف الموجه من v_1 إلي v]. وواضح أن هذا الحرف هو أقصر مسار من المصدر v_1 إلى v. ثم نختبر المسارات من v_1 إلى جميع الرؤوس فى المجموعة $V-Y$ (حيث V هى مجموعة رؤوس المخطط البياني المعطى G) والتى تمر فقط عبر رؤوس فى المجموعة Y كرؤوس وسيطه (intermediate vertices). وأقصر هذه المسارات هو أقصر مسار (a shortest path) [وهذا سنثبته بإذن الله فيما بعد]. والرأس الموجودة فى نهاية هذا المسار تضاف إلى المجموعة Y، والحرف الموجود فى آخر هذا المسار ويلامس (touches) هذه الرأس يضاف إلى المجموعة F. ونستمر فى هذه العملية/ الإجراء (procedure) إلى أن تصبح المجموعة Y مساوية للمجموعة V. وعند هذه اللحظة تصبح المجموعة F محتوية على الأحرف المطلوبة فى أقصر مسارات.

الخوارزمية4-9: خوارزمية ديجسترا عالية المستوى لإيجاد أقصر مسارات من مصدر أحادى

High-Level Dijkstra's Algorithm for Single-Source Shortest Paths

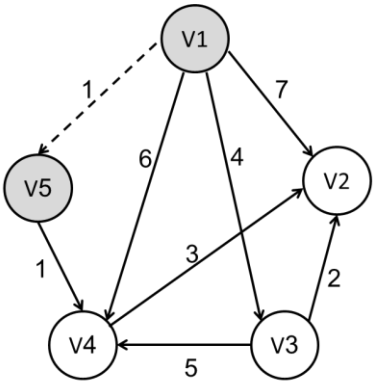
$Y = \{v_1\}$;
$F = \emptyset$;
while (the instance is not solved) {
  select a vertex v from $V - Y$, that has a // selection
  shortest path form v_1, using only vertices // procedure and
  in Y as intermediates; // feasibility check
  add the new vertex v to Y;
  add the edge (on the shortest path) that touches v to F;
  if ($Y == V$) // solution check
    the instance is solved;
}

مثال 4-18: نفرض أن لدينا المخطط البياني الموجة الموزن G المبين فى شكل 4-34. طبِّق خوارزمية "ديجسترا" لإيجاد أقصر مسارات من المصدر v_1 إلى جميع الرؤوس الأخرى فى هذا المخطط.

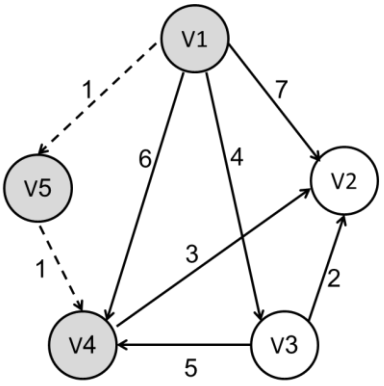


شكل 4–34
المخطط البياني G

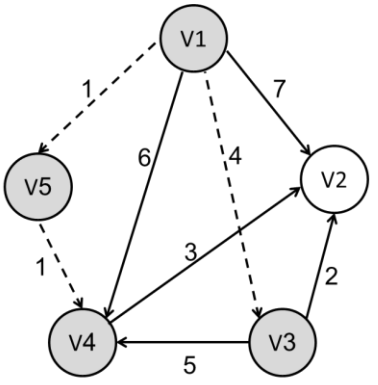
الحـــل :



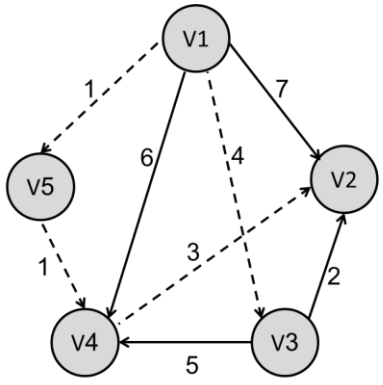
أ) نختار الرأس V5 لأنها الأقرب إلى V1



ب) نختار الرأس V4 لأن لها أقصر مسار من V1 يمر فقط عبر رؤوس في {V5} كرؤوس وسيطة



جـ) نختار الرأس V3 لأن لها أقصر مسار من V1 يمر
فقط عبر رؤوس في {V4,V5} كرؤوس وسيطة



د) أقصر مسار من V1 إلى V2 هو
⟨V1, V5, V4, V2⟩

شكل 4-35

خطوات خوارزمية "ديجسترا" للمخطط G

فى كل خطوة: الرؤوس في Y مظللة، والأحرف فى F خطوط مقطعة

الخوارزمية السابقة عالية المستوى تصلح فقط لشخص يحل المسألة بالنظر وفحص مخطط بياني صغير. وفيما يلي نعطي خوارزمية مفصلة (detailed algorithm) وفيها نمثل المخطط البياني الموزن بمنظومة ثنائية البعد. وهذه الخوارزمية شبيهة جدا بالخوارزمية 4- 5 [إجراء "بريم"]، والفارق بينهما أننا هنا بدلا من المنظومتين nearest, distance نستخدم المنظومتين touch, length بحيث أنه للقيم i=2,3,...,n:

touch[i] = مؤشر (index) الرأس v فى المجموعة Y بحيث أن الحرف $\langle v, v_i \rangle$ هو آخر حرف (last edge) فى أقصر مسار حالي (current shortest path) من v_1 إلى v_i يمر فقط عبر رؤوس فى Y كرؤوس وسيطة.

length[i] = طول أقصر مسار حالي من v_1 إلى v_i يمر فقط عبر رؤوس فى Y كرؤوس وسيطة.

الخوارزمية 4-10: إجراء "ديجسترا" Dijkstra's procedure

المطلوب: إيجاد أقصرمسارات من v_1 إلى جميع الرؤوس الأخرى فى مخطط بياني موجه موزن.

المدخلات: عدد صحيح $n \geq 2$، ومخطط بياني موجه موزن متصل يحتوي على رؤوس عددها n. والمخطط ممثل بمنظومة ثنائية البعد W مؤشرات كل من صفوفها وأعمدتها تقع فى المدى من 1 إلى n، حيث W [i][j] هو وزن الحرف المتجه من الرأس رقم i (i-th vertex) إلى الرأس رقم j (j-th vertex).

المخرجات: مجموعة أحرف F تحتوى على الأحرف فى أقصر مسارات.

الإجراء:

```
void dijkstra ( int n, const number W[ ][ ], set_of_edges& F )
{
    index i, vnear;
    edge e ;
    index touch[2..n];
    number length[2..n];

    F = ∅;
    for ( i = 2; i <= n; i+ + ) {          // for all vertices, initialize v1
        touch[i] = 1;                      // to be the last vertex on the
        length[i] = W [1][i];              // current shortest path from
    }                                      // v1, and initialize length of
                                           // that path to be the weight
                                           // on the edge from v1.
    repeat ( n – 1 ) times ) {             // add all n–1 vertices to Y.
        min = ∞ ;
        for ( i = 2; i <= n; i++ )         // check each vertex for
            if ( 0 ≤ length[i] < min)  {   // having shortest path.
                min = length[i];
                vnear = i;
            } // end if
        e = edge from vertex indexed by touch[vnear]
            to vertex indexed by vnear;
        add e to F;
```

```
    for ( i = 2; i <= n; i+ + )
      if ( length[vnear] + W[vnear][i] < length[i] ) {
          length[i]= length[vnear] + W [vnear][i];
          touch[i] = vnear;          // for each vertex not in Y,
      } // end if                    // update its shortest path.
    length [vnear] = -1;             // add vertex indexed by
                                     // vnear to Y
  } // end repeat
}
```

ونظراً لأننا فرضنا أنه يوجد مسار من v_1 الي كل رأس أخرى فإن المتغير vnear تكون له قيمة جديدة فى كل تكرير فى عروة repeat. وإذا لم يكن هذا هو الواقع فإن الخوارزمية كما هي مكتوبة ستنتهي بإضافة الحرف الأخير مرة تلو الأخرى(over and over) حتى يكتمل عدد $n-1$ من تكريرات عروة repeat.

والخوارزمية 4-10 تُوجِد فقط الأحرف فى أقصرمسارات، ولا تعطى أطوال هذه المسارات. وهذه الأطوال يمكن الحصول عليها من الأحرف. ويمكننا بتعديل بسيط للخوارزمية حساب الأطوال وتخزينها فى منظومة أيضا.

والتحكم (control) فى الخوارزمية 4-10 مطابق لنظيره فى الخوارزمية 4-5، ولذلك فمن تحليل خوارزمية 4-5 نعلم أنه بالنسبة للخوارزمية 4-10 درجه التعقيد الزمنية (time complexity) تعطى بالعلاقة:

$$T(n) = 2(n-1)^2 \in \Theta(n^2)$$

ومن الممكن إثبات أن الخوارزمية 4-10 صحيحة (correct)، أى أنها تنتج دائما أقصر مسارات. والبرهان يستخدم أسلوب الاستقراء (induction argument) شبيهاً بالأسلوب المستخدم لإثبات أن خوارزمية "بريم" (الخوارزمية 4-5) تنتج دائما شجرة مولدة صغرى.

وكما هو الحال بالنسبة لخوارزمية "بريم" فإن خوارزمية "ديجسترا" يمكن تنفيذها (implemented) باستخدام كومة (a heap). ودرجة تعقيد تنفيذ الكومة $\Theta(m \lg m)$، حيث m هوعدد الأحرف .

ويلاحظ أنه إذا كانت جميع أوزان/ تكاليف الأحرف غير سالبه (nonnegative costs) فإنه يمكننا التأكد من أن المسار (path) من المصدر إلى الرأس ذات أقصر مسار من المصدر يمر فقط عبر الرؤوس فى المجموعة Y. وبالتالى فمن الضروري فقط أن نسجل (record) لكل رأس v أقصر مسافة (shortest distance) من المصدر إلى v عبر مسار يمر فقط عبر رؤوس في Y. وفيما يلي الخوارزمية الرسمية(formal algorithm).

الخوارزمية 4-11: خوارزمية "ديجسترا" لإيجاد أقصر مسارات من مصدر أحادى

Single-Source Shortest Paths Dijkstra's Algorithm

المدخلات: مخطط بياني موجه $G = (V, E)$، ومصدر $v_0 \in V$، ودالة ℓ من الأحرف إلى الأعداد الحقيقيه غيرالسالبة (nonnegative reals). وسنعتبر أن $\ell(v_i, v_j)$ يساوى $+\infty$ إذا لم يكن (v_i, v_j) حرفا، وأن $v_i \neq v_j$، وأن $\ell(v, v) = 0$.

المخرجات: لكل رأس $v \in V$ توجد الخوارزمية أصغر قيمة لمجموع (أوزان / أطوال) قيم الدالة ℓ لأحرف مسار P من v_0 إلى v، وذلك من بين جميع المسارات P المختلفة.

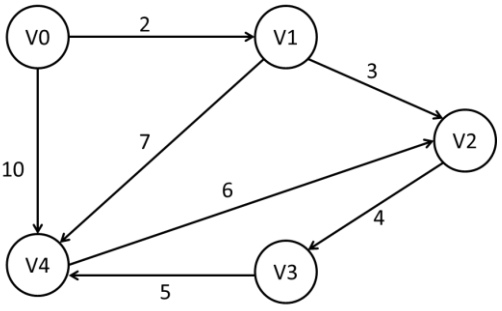
(the minimum –over all paths P form v_0 to v– of the sum of the labels / weights of the edges of P).

الطريقة: ننشئ مجموعه $Y \subseteq V$ بحيث أن أقصر مسار من المصدرالى أي رأس v فى Y يقع كلية فى Y. والمنظومة D[v] ستحتوي على وزن / طول / كلفة (cost) أقصر مسار حالى (current shortest path) من v_0 إلى v والذى يمر فقط عبر رؤوس Y.

الخوارزمية:

```
1. Y = {v0};
2. D[v0] = 0;
3. for ( each v in V − {v0} ) { D[v] = ℓ (v0, v); }
4. while ( Y ≠ V ) {
5. choose a vertex w in V − Y such that D[w] is a minimum;
6. add w to Y;
7. for ( each v in V − Y )
8. D[v] = MIN( D[v], D[w] + ℓ (w, v) );
   }
```

مثال4–19: نفرض أن لدينا المخطط البياني الموجه الموزن G المبين فى شكل 4-36. طبِّق خوارزمية "ديجسترا" لإيجاد أقصر مسارات من المصدر v_0 إلى جميع الرؤوس الأخرى فى هذا المخطط، وكذلك أطوال هذه المسارات الأقصر.



شكل 4-36

المخطط البياني G

الحـــل:

ابتداءً $Y = \{v_0\},\ D[v_0] = 0,$

وللقيم $i = 1, 2, 3, 4$ $D[v_i] = 2, +\infty, +\infty, 10;$

وفى أول تكرير لعروة while فى السطور $4 \rightarrow 8$ نختار $w = v1$ لأن $D[v1] = 2$ هو أصغر قيمة (minimum) فى المنظومة D. وبعد ذلك نحصل على القيمتين / المسافتين:

$$D[v2] = MIN(+\infty, 2 + 3) = 5$$

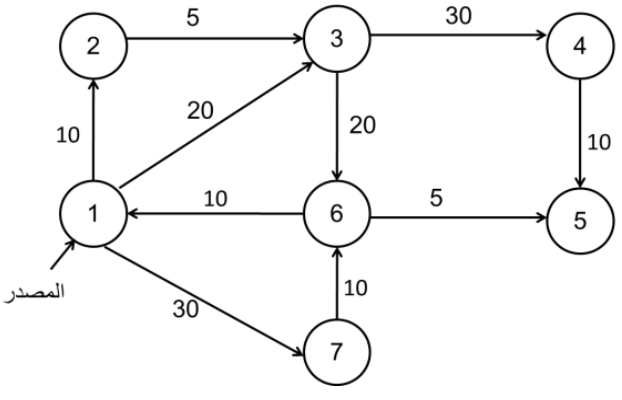
$$D[v4] = MIN(10, 2 + 7) = 9$$

والجدول التالى يلخص تتابع قيم منظومة المسافات D التى نحصل عليها وكذلك الحسابات الأخرى فى الخوارزمية 4- 11.

Iteration	Y	W	D[w]	D[v_1]	D[v_2]	D[v_3]	D[v_4]
Initial	$\{v_0\}$	-	-	2	$+\infty$	$+\infty$	10
1	$\{v_0,v_1\}$	v_1	2	2	5	$+\infty$	9
2	$\{v_0,v_1,v_2\}$	v_2	5	2	5	9	9
3	$\{v_0,v_1,v_2,v_3\}$	v_3	9	2	5	9	9
4	$\{v_0,v_1,v_2,v_3,v_4\}$	v_4	9	2	5	9	9

مثال 4-20: أعد حل المثال السابق (مثال 4- 19) بالنسبة:

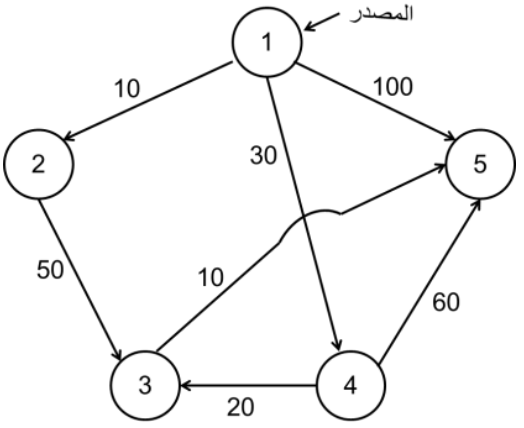
أ) للمخطط البياني G1 (شكل 4-37).



شكل 4-37

المخطط البياني G1

ب) للمخطط البياني G2 (شكل 4-38).



شكل 4-38

المخطط البياني G2

الحـــل: أ)

Y	w	D[w]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
{1}	-	-	0	10	20	+∞	+∞	+∞	30
{1,2}	2	10			15	+∞	+∞	+∞	30
{1,2,3}	3	15				45	+∞	35	30
{1,2,3,7}	7	30				45	+∞	35	
{1,2,3,7,6}	6	35				45	40		
{1,2,3,7,6,5}	5	40				45			
V	4	45							

ب)

Y	w	D[w]	D[1]	D[2]	D[3]	D[4]	D[5]
{1}	-	-	0	10	+∞	30	100
{1,2}	2	10			60	30	100
{1,2,4}	4	30			50		90
{1,2,4,3}	3	50					60
{1,2,4,3, 5} = V	5	60					

نظرية 4-3: صحة ودرجة تعقيد خوارزمية "ديجسترا"

Correctness and Complexity of Dijkstra's Algorithm

الخوارزمية 4-11 تحسب طول / وزن / تكلفة (cost) أقصر مسار من المصدر v_0 إلى كل رأس أخرى فى المخطط البياني، وتتطلب زمناً قدره $O(n^2)$.

البرهان: (i) درجة تعقيد الخوارزمية:

عروة **for** فى السطرين 7, 8 تتطلب خطوات (steps) $O(n)$، كما يتطلب اختيار الرأس w فى السطر 5. وهذه هي التكاليف الغالبة (dominant costs) فى عروة while فى السطور $4 \rightarrow 8$. وعروة while بدورها تُنفَّذ عدداً من المرات (times) قدره $O(n)$، وبالتالى تكون التكلفة الكلية قدرها $O(n^2)$. وأما السطور $1 \rightarrow 3$ فواضح أنها تتطلب زمناً قدره $O(n)$.

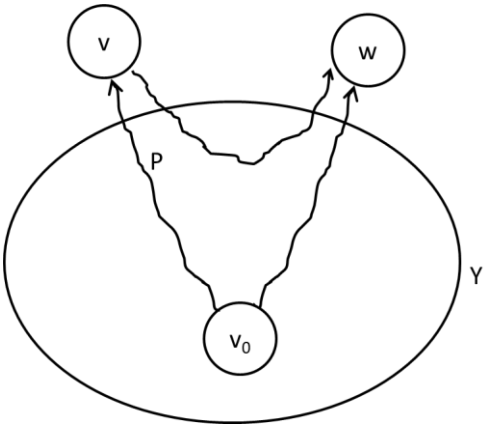
(ii) صحة الخوارزمية:

بالنسبة لصحة الخوارزمية علينا أن نثبت بالاستقراء (induction)

على حجم / سعة المجموعة Y (size of) أنه لكل رأس v فى Y فإن D[v] يساوي طول أقصر مسار (a shortest path) من v_0 إلى v. وبالإضافة إلى ذلك فإنه لكل رأس $v \in V - Y$ فإن D[v] هو طول أقصر مسار من v_0 إلى v يقع كلية فى Y (lies wholly within)، ما عدا v نفسها.

الأساس (Basis): $\|Y\| = 1$. أقصر مسار من v_0 إلى نفسها طوله 0، وأى مسار من v_0 إلى v - يقع كلية فى Y ما عدا v- يتكون من الحرف الوحيد (v_0, v) (single edge). وبالتالى فإن السطرين 2, 3 يعطيان القيمة الصحيحة الابتدائية للمنظومة D (correctly initialize).

الخطوة الاستقرائية (Inductive step): نفرض أننا اخترنا الرأس w فى السطر 5. إذا لم يكن D[w] هو طول أقصر مسار (a shortest path) من v_0 إلى w، فيجب أن يكون هناك مسار أقصر P. هذا المسار P يجب أن يحتوى على رأس ما (some vertex) غير w ليست فى Y. نفرض أن v هى أول هكذا رأس (first such vertex) على المسار P. ولكن عندئذ ستكون المسافة من v_0 إلى v أقصر من D[w]، وبالإضافة إلى ذلك فإن أقصر مسار إلى v يقع كلية فى Y، ماعدا v نفسها. [انظر شكل 4- 39]. وهكذا بالفرض الاستقرائى (inductive hypothesis) وصلنا إلى أن $D[v] < D[w]$ عندما اخترنا w، وهذا يناقض (contradicts) أن D[w] قيمة صغرى (a minimum). ونستنتج من ذلك أن المسار P لا وجود له، وأن D[w] هو طول أقصر مسار من v_0 إلى w.



شكل 4-39

مسارات إلى الرأس v

والجزء الثانى من الفرض الاستقرائي -أن D[w] يبقى صحيحا (remains correct) - واضح بسبب السطر 8 فى الخوارزمية.

ملاحظة: إذا تم تخزين (storing) المنظومة D باعتبارها كومة (a heap) فإنه يمكننا الحصول على درجة تعقيد $O(m\ lg\ n)$ حيث m هو عدد الأحرف. وهذه الدرجة تكون أفضل من الدرجة $O(n^2)$ التى حصلنا عليها إذا كان $m\ lg\ n < n^2$، أى إذا كان $m < n^2 / lg\ n$، وهذه هى الحالة المعتادة (usual case).

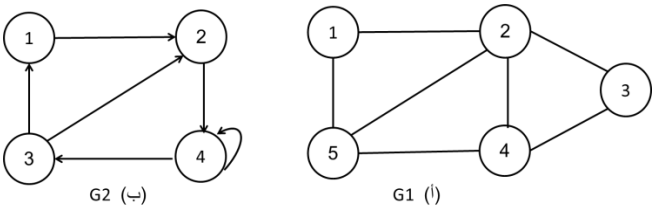
تمرينات الفصل الرابع

تمرينات الفصل الرابع

4-1) وَضِّح تمثيل كل من المخططين البيانيين التاليين G1 (شكل 4-40- أ) و G2 (شكل 4-40- ب)

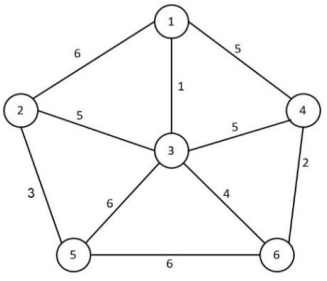
i) باستخدام مصفوفة التجاور.

ii) باستخدام منظومة قائمة التجاور.



شكل 4-40

4-2) وَضِّح تمثيل المخطط البياني المُوَجَّه المُوَزَّن G التالى (شكل 4-41) باستخدام منظومة قوائم التجاور.



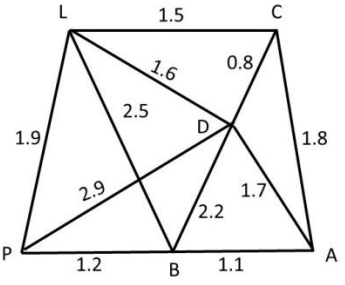
شكل 4-41

3-4) يراد إنشاء خط أنابيب (pipeline) يربط ست مدن: A, B , C, D, L, P. وتكلفة إنشاء أى خط رابط محتمل بين مدينتين (بملايين الدنانير) يعتمد على كل من المسافة (distance) بينهما و تضاريس الأرض (terrain)، و هذه التكلفة مبينة فى المخطط البياني المُوَزَّن الذى يظهر فى الشكل التالي (شكل 4-42). أوجد نظاما لخطوط الأنابيب (a system of pipelines) بحيث يصل بين جميع المدن (connect all the cities) وتكون تكلفته الإجمالية أقل ما يمكن (minimize the total cost).

أ) استخدم خوارزمية " كروسكال" لإيجاد شجرة مولِّدة صغرى (MST) للمخطط البياني المُوَزَّن G المبيَّن فى شكل 4-42. وَضِّح تنفيذ خطوات الخوارزمية خطوةً خطوة [أي بَيّن ترتيب (order) الأحرف التى تضاف لتكوين الشجرة (MST)].

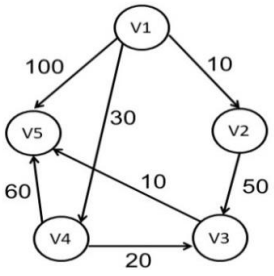
ب) كم قيمة الوزن الكلى (total weight) لأحرف الشجرة (MST) التى حصلت عليها فى أ).

جـ) هل هذه الشجرة (MST) وحيدة (unique)؟ عَلِّل إجابتك.



شكل 4-42

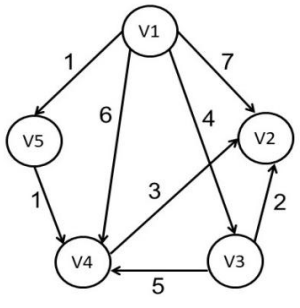
4-4-أ) طَبِّق خوارزمية "ديجسترا" لإيجاد أقصر مسارات من الرأس v1 إلى جميع الرؤوس الأخرى فى المخطط البياني G المبيَّن فى الشكل التالي (شكل 4-43). وَضِّح تنفيذ خطوات الخوارزمية خطوةً خطوة بإكمال الجدول التالي.



شكل 4-43

Iteration	Y	w	D[w]	D[v2]	D[v3]	D[v4]	D[v5]
Initial							
.							
.							
.							

ب) أعد حل السؤال في أ) بالنسبة للمخطط البياني G المبيَّن فى الشكل التالى (شكل 4-44).

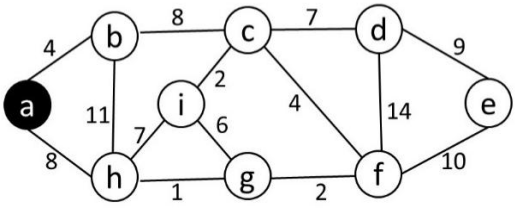


شكل 4-44

4-5-أ) وَضِّح خطوات تطبيق خوارزمية "كروسكال" خطوةً خطوة لإيجاد شجرة مولِّدة صغرى (MST) للمخطط البياني المُوَزَّن G المبيَّن فى الشكل التالى (شكل 4-45).

ب) كم قيمة الوزن الكلى لأحرف الشجرة (MST) التى حصلت عليها فى أ)؟

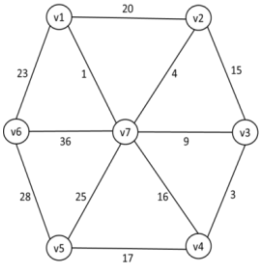
جـ) هل هذه الشجرة (MST) وحيدة؟ عَلِّل إجابتك.



شكل 4-45

6-4) وَضِّح خطوات تطبيق خوارزمية "بريم" خطوةً خطوة لإيجاد شجرة مولِّدة صغرى (MST) للمخطط البياني المُوَزَّن G المبيَّن فى شكل 4-45، وذلك باعتبار أن الرأس a هى رأس البداية (starting vertex).

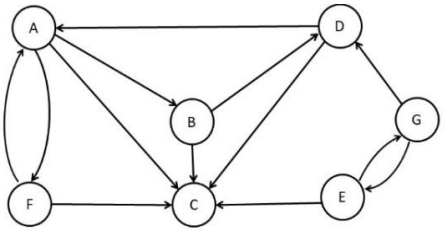
7-4) نفرض أن لدينا المخطط البياني المُوَزَّن G المبيَّن فى الشكل التالي (شكل 4-46). وَضِّح خطوات تطبيق خوارزمية "كروسكال" لإيجاد شجرة مولِّدة صغرى (MST) للمخطط G.



شكل 4-46

8-4) نفرض أن مخططا بيانيا موجَّهاً G يمثل علاقة ثنائية R. اذكر شرطاً على المخطط G يتحقق إذا وفقط إذا كانت العلاقة R متعدية (transitive).

9-4) نفرض أن لدينا المخطط البياني المُوَجَّه المبيَّن فى الشكل التالى (شكل 4-47).



مخطط بيانى موجه A digraph

شكل 4-47

استخدم استراتيجية البحث بالعمق أولا (DFS) لاجتياز المخطط - مبتدئا بزيارة الرأس A - والحصول على غابة مولِّدة للمخطط بالعمق أولا.

i-10-4) أوجد شجرة البحث بالعمق أولا للمخطط البياني المبيَّن فى شكل 4-47 باعتبار أن رأس البداية هى G، وبفرض تحقق الشرط التالى عن ترتيب قائمة التجاور(the adjacency list order):

أ) أيُّ قائمة تجاور تكون مرتبة ترتيبا أبجديا (each adjacency list is in alphabetical order).

ب) أيُّ قائمة تجاور تكون مرتبة ترتيبا أبجديا عكسيا (in reverse alphabetical order).

ii) أعد حل الجزء i) من السؤال، ولكن بإيجاد شجرة البحث بالعرض أولاً بدلا من شجرة البحث بالعمق أولاً.

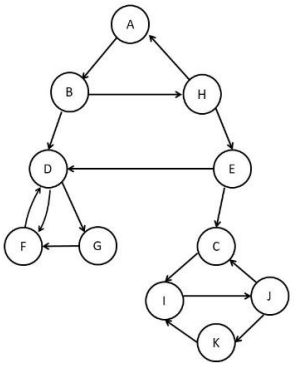
4-11- i) المطلوب تطبيق خوارزمية DFS على المخطط البياني المُوَجَّه المبيَّن فى الشكل التالى (شكل 4-48) وتصنيف جميع أحرفه أى بيان أنواعها، وذلك بفرض أن:

أ) رؤوس المخطط مؤشَّرة بترتيب أبجدى (indexed in alphabetical order) فى المنظومة adjVertices، وأى قائمة تجاور تكون مرتبة ترتيبا أبجديا.

ب) رؤوس المخطط مؤشرة بترتيب أبجدي عكسي (in reverse alphabetical order) فى المنظومة adjVertices، وأى قائمة تجاور تكون مرتبة ترتيبا أبجديا.

جـ) رؤوس المخطط مؤشرة بترتيب أبجدي فى المنظومة adjVertices، وأي قائمة تجاور تكون مرتبة ترتيبا أبجديا عكسيا.

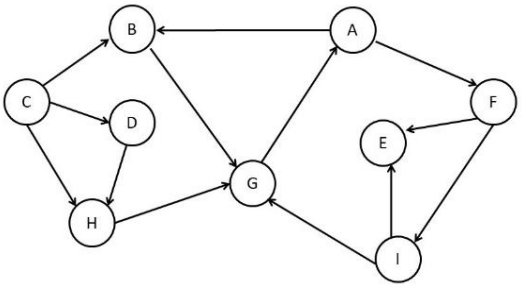
د) رؤوس المخطط مؤشرة بترتيب أبجدى عكسي فى المنظومة adjVertices، وأي قائمة تجاور تكون مرتبة ترتيبا أبجديا عكسيا.



شكل 4-48

مخطط بيانى موجه A digraph

ii) أعد حل الجزء i) من السؤال على المخطط البياني المُوَجَّه المبيَّن فى الشكل التالى (شكل4-49).



شكل 4-49

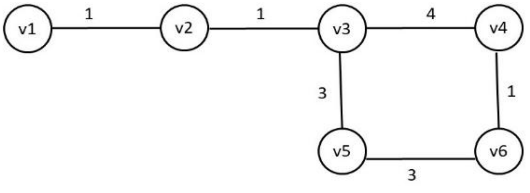
مخطط بيانى موجه A digraph

12-4) اعط مثالا لمخطط بيانى متصل مُوَزَّن غير موجه (connected, weighted, undirected graph) ورأس بداية (a start vertex) بحيث أن أياً من شجرة البحث بالعمق أولاً وشجرة البحث بالعرض أولاً ليست شجرة مولِّدة صغرى (neither the DFS tree nor the BFS tree is an MST) بغض النظر عن كيفية ترتيب قوائم التجاور (regardless of how the adjacency lists are ordered).

13-4) اعط مثالا لمخطط بيانى موجه مُوَزَّن (a weighted directed graph) ورأس مصدر (a source vertex) بحيث أن أياً من شجرة البحث بالعمق أولا وشجرة البحث بالعرض أولا ليست شجرة أقصر مسار (a shortest-path tree) بغض النظر عن كيفية ترتيب قوائم التجاور.

14-4) هل خوارزمية ديجسترا لإيجاد أقصر مسار تعمل بصورة صحيحة (work correctly) فى حالة احتمال وجود أوزان سالبة (if weights may be negative)؟ عَلِّل إجاباتك ببرهان أو بإعطاء مثال معاكس (by an argument or a counter example).

15-4) بالنسبة للمخطط البياني المبيَّن فى الشكل التالى (شكل 4- 50) وَضِّح ما هى الأحرف التى ستظهر فى الشجرة المولِّدة الصغرى MST التى تنشئها خوارزمية بريم باعتبار أن الرأس V_1 هى المصدر، وما هى الأحرف التى تنشئها خوارزمية ديجسترا لإيجاد أقصر مسار من الرأس V_1 إلى الرأس V_6.



شكل 4-50

16-4) فيما يلي قوائم التجاور (adjacency lists) لأحد المخططات البيانية المُوَجَّهة [حيث أوزان الأحرف مكتوبة بين أقواس]. و للإيضاح فإن المخطط مبين أيضا فى الشكل التالي (شكل 4-51).

A: B(4), F(2)

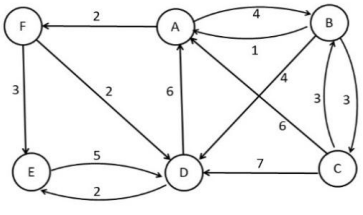
B: A(1), C(3), D(4)

C: A(6), B(3), D(7)

D: A(6), E(2)

E: D(5)

F: D(2), E(3)



شكل 4-51

مخطط بيانى موجه

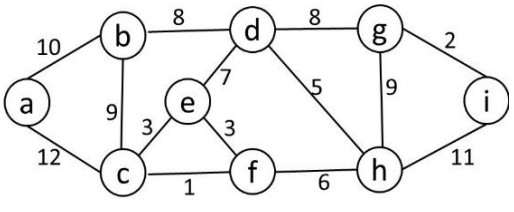
أ) هذا المخطط البياني المُوَجَّه يحتوى على ثلاثة أقصر مسارات(three shortest paths) من الرأس C إلى الرأس E [أى أنها جميعا لها الوزن الكلى نفسه (same total weight)]. أوجد هذه المسارات الثلاثة. [اكتب قائمة متتابعة الرؤوس (list the sequence of vertices) فى كل مسار].

ب) أى هذه المسارات هو المسار الذى تنشئه خوارزمية ديجسترا لإيجاد أقصر مسار باعتبار أن الرأس C هى المصدر s = C؟ (عَلِّل إجابتك بشرح السبب، أو ببيان الخطوات الرئيسية فى الخوارزمية).

4-17-أ) هل صحيح (true) دائما أم أحيانا أم غير صحيح إطلاقا أن الترتيب (order) الذى تزار به رؤوس أى مخطط بيانى لأول مرة (vertices are first visited) [أى ترتيب إضافتها الى الشجرة] فى خوارزمية بريم لإيجاد الشجرة المولِّدة الصغرى هو نفسه الترتيب الذى تزار به الرؤوس فى خوارزمية البحث بالعمق أولا أو خوارزمية البحث بالعرض أولا؟ عَلِّل إجابتك بإعطاء أمثلة أو بإعطاء برهان.

ب) أعد حل السؤال أ) بالنسبة لخوارزمية ديجسترا لإيجاد أقصر مسار.

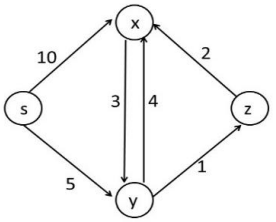
4-18) أوجد شجرة مولِّدة صغرى (MST) للمخطط البياني المُوَزَّن المبيَّن فى شكل 4-52.



شكل 4-52

مخطط بيانى مُوَزَّن

19-4) اذكر ترتيب إضافة رؤوس من المخطط البياني المُوَجَّه المُوَزَّن G التالى (شكل4- 53) إلى الشجرة المولِّدة الصغرى التى تنشئها خوارزمية ديجسترا لإيجاد أقصر مسار من الرأس s الى الرأس x.



شكل 4-53

مخطط بيانى موجه مُوَزَّن G

20-4) نفرض أننا قد أُعطينا تمثيلا لمخطط بيانى موجه بقائمة تجاور. كم من الوقت يستغرق حساب درجة الخروج لكل رأس (the out-degree of every vertex) فى المخطط؟ وكم يستغرق حساب درجات الدخول (the in-degrees)؟

21-4) مدوَّر (transpose) أى مخطط بيانى موجه G=(V,E) هو المخطط البياني $G^T = (V, E^T)$، حيث E^T={ (v, u) : (u, v) ∈ E }. أى أن المخطط G^T هو المخطط G مع عَكْس (reversing) جميع أحرفه. اكتب خوارزمية ذات كفاءة عالية (an efficient algorithm) لحساب المخطط G^T من المخطط G بفرض أن المخطط G معطى بالتمثيل:

ا) بقائمة تجاور.

ب) بمصفوفة تجاور. وقم بتحليل (analyzing) وقت التشغيل

(running time) بالنسبة لكل من الخوارزميتين.

22-4) مُرَبَّع (square) أى مخطط بيانى مُوَجَّه $G = (V, E)$ هو المخطط البياني $G^2 = (V, E^2)$، بحيث أن:

$(u,w) \in E^2$ iff for some $v \in V$ $[(u,v) \in E \ \& \ (v,w) \in E]$.

أى أن المخطط G^2 يحتوى على حرف بين u, w كلما احتوى المخطط G على مسار ذى حرفين بالضبط (a path with exactly two edges) بين u, w. اكتب خوارزمية ذات كفاءة عالية (an efficient algorithm) لحساب المخطط G^2 من المخطط G بفرض أن المخطط G معطى بالتمثيل:

أ) بقائمة تجاور.

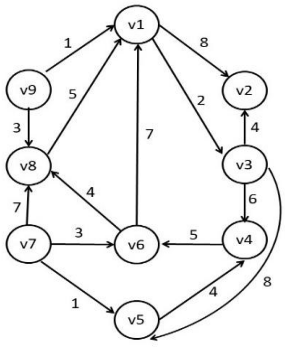
ب) بمصفوفة تجاور. وقم بتحليل وقت التشغيل بالنسبة لكل من الخوارزميتين.

23-4) كم يساوى زمن تشغيل خوارزمية BFS (running time of) إذا كان مخطط الإدخال (input graph) ممثلا بمصفوفة تجاور، والخوارزمية قد تم تعديلها لتتعامل مع هذه الصيغة من المدخلات(modified to handle this form of input)؟

24-4) خوارزمية كروسكال يمكن أن تعيد أشجاراً مولِّدةً مختلفة (different spanning trees) للمخطط البياني المدخل نفسه(for the same input G graph)، وذلك بناءً على كيفية معالجة الحالات المتساوية (depending on how ties are broken) عند ترتيب الأحرف (when the edges are sorted). اثبت أنه لأى شجرة مولِّدة صغرى T للمخطط G، توجد طريقة

لترتيب (sorting) أحرف المخطط G فى خوارزمية كروسكال حتى تعيد الخوارزمية الشجرة T.

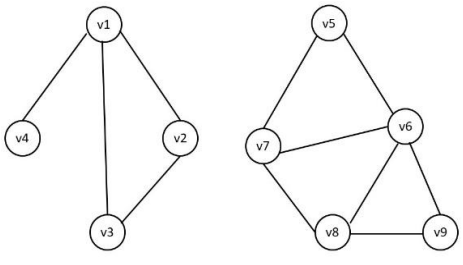
4-25-أ) أوجد غابة مولِّدة بالعمق أولا للمخطط البياني المُوَجَّه التالى (شكل 4-54).



شكل 4-54

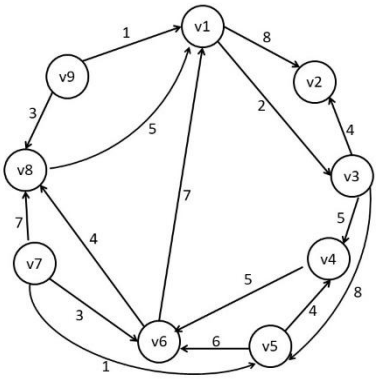
ب) أوجد الشجرة المولِّدة الصغرى للمخطط البياني المُوَجَّه فى شكل 4-54 بفرض أن الأحرف المبيَّنة فى الشكل غير موجهة (undirected).

4-26) أوجد غابة مولِّدة بالعمق أولا للمخطط البياني غير المُوَجَّه المبيَّن فى شكل 4-55. اختر أى رأس تشاء لتبدأ أى شجرة.



شكل 4-55

4-27) بتطبيق خوارزمية ديجسترا أوجد أقصر مسافة من الرأس v_6 إلى كل رأس v فى المخطط البياني G المبيَّن فى شكل 4- 56.



شكل 4-56

4-28) اثبت أن أي خوارزمية لحل مسألة إيجاد شجرة مولِّدة صغرى في مخطط بياني G يلزمها على الأقل: $\Omega(m+n)$ من العمليات في أسوأ حالة، حيث n هي عدد الرؤوس و m هي عدد الأحرف في المخطط البياني.

❖❖❖

أجوبة تمرينات الفصل الثانى

أجوبة تمرينات الفصل الثالث

أجوبة تمرينات الفصل الرابع

أجوبة تمرينات الفصل الثانى

1-2) عدد مقارنات K مع عناصر المنظومة E: n

عدد المقارنات مع المتغير index: n+1

عدد عمليات الجمع: n

عدد عمليات الإسناد للمتغير index: n+1

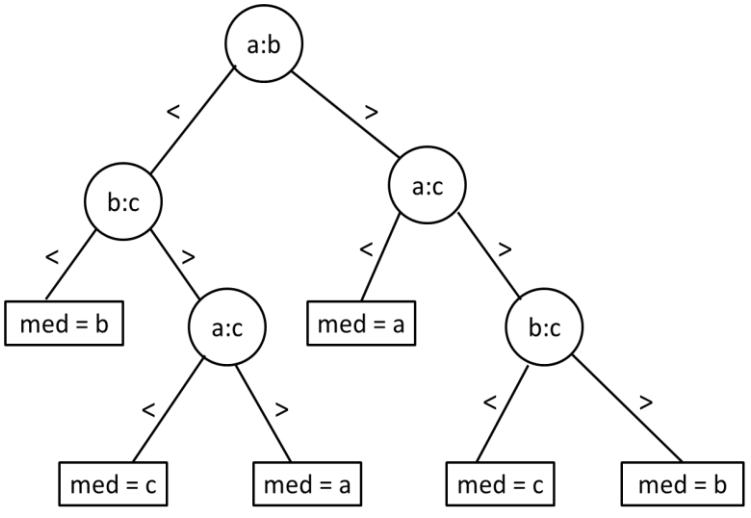
وبالتالى يكون العدد الإجمالى للعمليات فى أسوأ حالة هو: $4n + 2$

2-2- أ)

```
if ( a < b )
    if ( b < c )
        median = b;
    else if ( a < c )
        median = c;
    else
        median = a;
else if ( a < c )
    median = a;
else if ( b < c )
    median = c;
else
    median = b;
```

ب) D: مجموعة مدخلات الخوارزمية هى مجموعة تباديل (set of permutations) ثلاثة عناصر.

جـ) شجرة قرار الخوارزمية:



د) عدد المقارنات فى أسوأ حالة = 3

هـ) نحتاج إلى ثلاث مقارنات فى أسوأ حالة، وهو ارتفاع شجرة القرار.

و) عدد المقارنات فى المتوسط $= \frac{2+3+3+2+3+3}{6} = 2\frac{2}{3}$

3-2) خذ عناصر المجموعة زوجاً زوجاً (pair up the entries) وأوجد أكبر العنصرين فى كل زوج. وإذا كانت n عدداً فرديا فسيبقى عنصر واحد لم يُختبر (عدد المقارنات $\lfloor n/2 \rfloor$ مقارنة). ثم أوجد أكبر عنصر بين هذه العناصر الكبرى بتطبيق خوارزمية findMax (الخوارزمية 2- 3) بما فى ذلك (including) العنصر غير المُختبَر إذا كانت n فردية (عدد المقارنات $\lfloor (n+1)/2 \rfloor$ - 1 مقارنة). فهذا هو أكبر عنصر فى المجموعة. ثم أوجد أصغر عنصر بين العناصر الصغرى بتطبيق التعديل المناسب للخوارزمية 3-2 بما فى ذلك أيضا العنصر غير المختبر إذا كانت n فردية (عدد المقارنات $\lfloor (n+1)/2 \rfloor - 1$ مقارنة). وهذا هو أصغر عنصر فى المجموعة. فسواءً كانت n فردية أو زوجية فإن العدد الإجمالى (total)

للمقارنات هو $\lfloor 3(n-1)/2 \rfloor$. الخوارزمية التالية تطبق الخطوات السابقة.

```
/* precondition:   n > 0.          */
if (odd(n))
      min = E[n − 1];
      max = E[n − 1];
else if ( E[n − 2] < E[n − 1] )
      min = E[n − 2];
      max = E[n − 1];
else
      max = E[n − 2];
      min = E[n − 1];
for (i = 0; i <= n − 3; i = i + 2)
      if ( E[i] < E[i + 1] )
          if ( E[i] < min )          min = E[i];
          if ( E[i + 1] > max )      max = E[i + 1];
      else
          if ( E[i] > max )          max = E[i];
          if ( E[i + 1] < min )      min = E[i + 1];
```

$$\lim_{n\to\infty} \frac{p(n)}{n^k} = \lim_{n\to\infty} \left(a_k + \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k}\right) = a_k > o \quad 4\text{-}2$$

5-2) الحل التالى يشمل الجزئين أ) و ب). الدوال التى تقع على السطر نفسه تتساوى رتبها المقاربة.

$\lg \lg n$

$\lg n$, $\ln n$

$(\lg n)^2$

$\sqrt{n}$

n

$n \lg n$

$n^{1+\epsilon}$

$n^2,\quad n^2 + \lg n$
n^3
$n - n^3 + 7n^5$
$2^{n-1},\quad 2^n$
e^n
$n!$

6-2) لإثبات أن n^k تقع فى $o(2^n)$، نثبت أن $\lim_{n\to\infty} \frac{n^k}{2^n} = 0$. ولإيجاد هذه النهاية نحتاج لتطبيق قاعدة هوبيتال عدة مرات. وبملاحظة أن

$2^n = e^{\ln 2^n} = e^{n \ln 2}$، فإن

$$\frac{d}{dn} 2^n = e^{n \ln 2} \ln 2 = \ln 2\ 2^n \approx 0.7\ 2^n = c\ 2^n$$

حيث $c \approx 0.7$

$$\Rightarrow \lim_{n\to\infty} \frac{n^k}{2^n} = \lim_{n\to\infty} \frac{k\ n^{k-1}}{c\ 2^n} = \lim_{n\to\infty} \frac{k(k-1)\ n^{k-2}}{c^2\ 2^n}$$
$$= \lim_{n\to\infty} \frac{k(k-1)(k-2)\ n^{k-3}}{c^3 \quad 2^n} = \dots$$
$$= \lim_{n\to\infty} \frac{k(k-1)(k-2)\dots\ (k-(k-1))}{c^k \qquad 2^n}$$
$$= \lim_{n\to\infty} \frac{k!}{c^k\ \ 2^n} = 0 \qquad \text{(لأن } k \text{ ثابت)}$$

7-2)

$$f(n) = \begin{cases} 1 & \text{for odd } n \\ n & \text{for even } n \end{cases}$$
$$g(n) = \begin{cases} n & \text{for odd } n \\ 1 & \text{for even } n \end{cases}$$

وتوجد أيضاً أمثلة باستخدام دوال متصلة على الأعداد الحقيقية (continuous functions on the reals)، وكذلك أمثلة باستخدام دوال رتيبة (monotonic functions).

8-2) الخوارزمية المعدلة هي:

```
int binarySearch (int [ ] E, int first, int last, int K)
1.  if (last < first)
2.          index = −1;
3.  else if (last == first)
4.          if (K == E[first])
5.              index = first;
6.          else
7.              index = −1;
8.  else
9.          int mid = (first + last)/2;
10.         if (K ≤ E[mid])
11.             index = binarySearch (E, first, mid, K);
12.         else
13.             index = binarySearch(E, mid + 1, last, K);
14. return index;
```

مقارنـــة مع خوارزميـــة البحث الثنائى الأصلية (الخوارزمية 2-4) فإن الخوارزمية المعدلة تجمع اختبارى الخوارزمية الأصلية الموجودين فى السطرين 5,7 وهما:

if (k == E[mid]) & **else if** (k < E [mid])

فى اختبار واحد فى السطر 10، والمدى الجزئى الأيسر(left subrange) تمت زيادته من mid-1 إلى mid، وذلك لأن E[mid] قد يحتوى على المفتاح key الذى نبحث عنه. كما احتجنا إلى حالة أساسية إضافية (an extra base case) فى السطور 3-7 وذلك لتختبر (test for) التطابق (exact equality) عندما يتقلص المدى إلى عنصر وحيد (range shrinks to a single entry).

وفى الحقيقة إذا أمكننا افتراض الشرط السابق $first \le last$ (the precondition)، فإنه يمكننا الاستغناء عن السطرين 2 ,1. والخوارزمية الحالية تجعل هذا الشرط السابق ينتشر (propagates) فى استدعاءات ارتدادية (recursive calls) بينما الخوارزمية الأصلية لا تحقق ذلك فى حالات معينة (certain cases).

2-9) الترتيب بالإدخال يتفوق على الترتيب بالدمج عندما يتحقق الشرط:

$$8\,n^2 < 64\,n \lg n \;\Rightarrow\; n < 8 \lg n \;\Rightarrow\; \frac{n}{8} < \lg n \;\Rightarrow\; 2^{n/8} < n$$

$$\Rightarrow \quad 2 \le n \le 43 \qquad \text{(باستخدام آله حاسبة)}$$

وبناءً عليه فلتحسين زمن التشغيل (running time) نعيد كتابة خوارزمية الترتيب بالدمج بحيث تستخدم الترتيب بالإدخال للمدخلات ذات السعة 43 أو أقل.

2-10) القيمة المطلوبة هى أصغر قيمة للمتغير n تحقق الشرط:

$$100\,n^2 < 2^n$$

$n = 13$: $16900 > \approx 8000$ $(2^{10} = 1024 \approx 1000)$

$n = 14$: $19600 > \approx 16000 \rightarrow 16384$

$n = 15$: $22500 < \approx 32000 \rightarrow 32768$

أى أن القيمة المطلوبة هي: $n = 15$.

11-2) نفرض أن جميع الشهور ثلاثون يوماً وأن جميع السنوات 365 يوماً.

t / f(n)	ثانية	دقيقة	ساعة	شهر	سنة
$\lg n$	2^{10^6}				
$\sqrt{n}$	10^{12}				
n	10^6				
$n \lg n$	62746				
n^2	10^3				
n^3	10^2				
2^n	19				
$n!$	9				

12-2) $n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^3)$

13-2) نفرض أن الإجراء FIND-MIN (A, r, s) يعيد مؤشر(index) أصغر عنصر فى المنظومة A بين المؤشرين r, s. من الواضح أن هذا يمكن تنفيذه (implemented) فى زمن $O(s - r)$ إذا كانت $r \leq s$.

Algorithm $SELCTION - SORT(A, n)$

input: $A = [a_1, a_2, \ldots, a_n]$

output: sorted A.

for $i = 1$ **to** $n - 1$ **do**

$j = FIND - MIN\ (A, i, n)$

$A[j] \leftrightarrow A[i]$

end for

استدعاءات الإجراء FIND-MIN والتى عددها n (n calls) تُعطِي الحد (bound) التالى على درجة التعقيد الزمنية (time complexity):

$$\Theta(\sum_{i=1}^{n-1}(n-i)) = \Theta(n^2)$$

وهذا يتحقق بالنسبة لكل من زمن التشغيل فى أحسن حالة وفى أسوأ حالة (best and worst- case running time).

2-15) يمكننا تعديل أى خوارزمية حتى يصبح زمن تشغيلها جيداً فى أحسن حالة عن طريق جعلها متخصصه لتعالج أى مدخلات أحسن حالة بكفاءة عالية (by specializing it to handle a best-case input efficiently).

2-16) فيما يلى صيغة ارتدادية للبحث الثنائى فى منظومة.

```
Algorithm BINARY − SEARCH(A, k, p, r)
    input: a sorted array A and a value k.
    Output: index i such that k = A[i] or − 1.
    if (p ≥ r and k ≠ A[p]) then
        return − 1
    end if
    j = A[(r + p)/2]
    if (k = A[j]) then
        return j
    else
        if (k < A[j]) then
            return BINARY − SEARCH(A, K, p, j − 1)
        else
            return BINARY − SEARCH(A, k, j + 1, r)
        end if
    end if
```

زمن التشغيل نحصل عليه من حل العلاقة:

$$T(n) = 1 + T(n/2), T(1) = 1$$

وهو $\Theta(\lg n)$.

2-17- أ) نعم، وذلك لأن $2^{n+1} = 2.2^n \leq 2.2^n$

ب) لا، وذلك لأن $2^{2n} = (2^n)^2$

ولا يوجد أى ثابت c بحيث أن $(2^n)^2$ يمكن أن تكون مُقَارَبَةً (asymptotically) أقل من أو تساوى $c\ 2^n$

$$\Rightarrow \quad 2^{2n} \neq O(2^n)$$

2-18) نفرض أن $f(n),\ g(n)$ دالتان موجبتان مُقَارَبَةً.

أ) خطأ.

$$f(n) = O(g(n)) \quad \not\Rightarrow \quad g(n) = O(f(n))$$

من الواضح مثلاً أن: $n = O(n^2)$ ولكن $n^2 \neq O(n)$.

ب) خطأ.

$$f(n) + g(n) \neq \Theta(\min(f(n), g(n)))$$

مثلاً لاحظ أن: $n + 1 \neq \Theta(\min(n, 1)) = \Theta(1)$

جـ) صح.

$$f(n) = O\big(g(n)\big) \Rightarrow \lg\big(f(n)\big) = O\left(\lg\big(g(n)\big)\right)$$

بشرط أن

$$f(n) \geq 1 \quad \& \quad \lg(g(n)) \geq 1$$

وذلك لأن

$$f(n) \leq c\ g(n) \Rightarrow \lg(f(n)) \leq \lg\big(c\ g(n)\big) = \lg c + \lg(g(n))$$

ولإثبات أن هذا المقدار أصغر من $b\ \lg(g(n))$ حيث b ثابت ما فإننا نفرض أن

$$\lg c + \lg(g(n)) = b \lg(g(n))$$

وبقسمة الطرفين على $\lg(g(n))$ فإننا نحصل علي:

$$b = \frac{\lg c + \lg(g(n))}{\lg(g(n))} = \frac{\lg c}{\lg(g(n))} + 1 \leq \lg c + 1$$

وذلك لأن: $\lg(g(n)) \geq 1$.

د) خطأ.

إذا كانت $f(n) = 2n \quad \& \quad g(n) = n$

فإن $2n \leq 2.n$

ولكن الشرط $2^{2n} \leq c\, 2^n$ لا يتحقق لأى ثابت c (من السؤال 2-17- ب).

هـ) صح وخطأ.

إذا كانت $f(n) < 1$ لقيم n الكبيرة، فإن $(f(n))^2 < f(n)$ والحد الأعلى (upper bound) سوف لا يتحقق. وما عدا ذلك $f(n) > 1$ والعبارة تكون صحيحة ببساطة (trivially true).

و) صح.

$f(n) \leq c\, g(n)$ لقيم c الموجبة.

وبالتالي $1/c.\, f(n) \leq g(n)$.

ز) خطأ.

من الواضح أن $2^n \not\leq c\, 2^{n/2} = c\sqrt{2^n}$

لأى ثابت c إذا كانت n كبيرة كبراً كافيا (sufficiently large).

2-19) عدد عمليات الإسناد للمتغير index: n

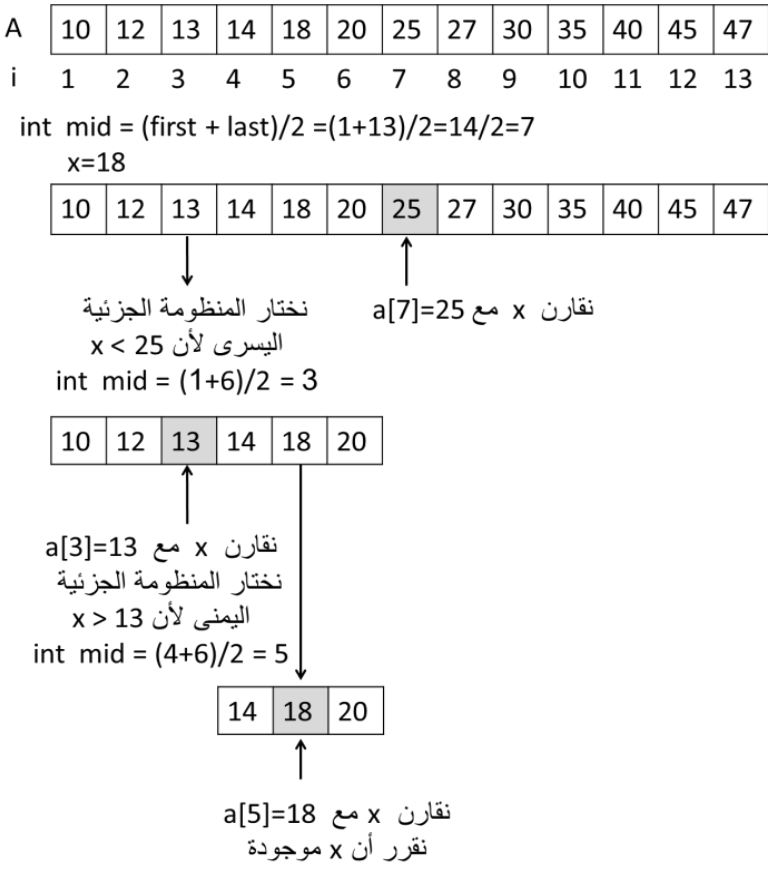
عدد عمليات الجمع: n-1

عدد مقارنات max مع عناصر المنظومة: n-1

عدد المقارنات مع المتغير index: n

وبالتالى فإن العدد الإجمالى للعمليات السابقة فى أسوأ حالة هو 4n-2. وإذا أضفنا عدد عمليات الإسناد فى السطر رقم 4 فى الخوارزمية وهو: $n - 1$، يصبح العدد الإجمالى 5n-3.

(20-2



$\Rightarrow$ found
index $= 5$

النتيجة: العدد الصحيح $x = 18$ موجود فى المنظومة المعطاة A.

المخرجات: $i = 5$، حيث $x = 18 = A[5]$.

أجوبة تمرينات الفصل الثالث

1-3) 1, n, n-1, n-2, …, 3, 2

و

n, n-1, n-2, …, 3, 1, 2

2-3) عدد مقارنات المفاتيح (key comparisons) n(n-1)/2. وعدد تحريكات العناصر (element movements) 2(n-1) حيث يتم تحريك E[first] إلى pivotElement، ثم يعاد تحريكه لموضعه فى كل مرة يتم فيها استدعاء الإجراء partition.

3-3- أ) بعد الاستدعاء الأول: [10 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,1] 10 كان هو الوتد (pivot)]. وتم إجراء تحريك واحد (one move) فى الإجراء partition.

بعد الاستدعاء الثانى: الترتيب نفسه [1 كان هو الوتد] لم يتم إجراء أى تحريك فى الإجراء partition.

ب) العدد الكلى لتحريكات العناصر: n/2 فى استدعاءات partition، بالإضافة إلى 2(n-1) لتحريكات الأوتاد فى خوارزمية quickSort نفسها.

4-3) إذا كانت المفاتيح مرتبة فعلا فى البداية، فإن عملية دمج النصفين (merging the two halves) ستحتاج إلى n/2 مقارنة فقط. وبالتالى فإن العلاقة الارتدادية (recurrence relation) ستصبح:

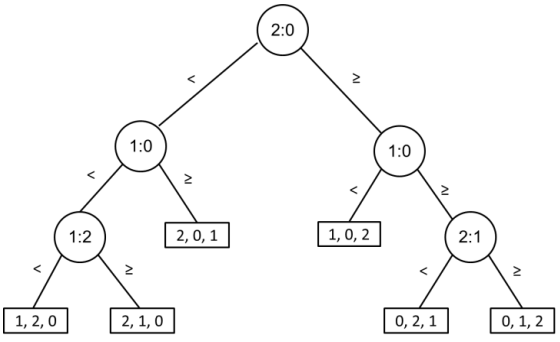
$$W(n) = W(\lceil n/2 \rceil) + W(\lfloor n/2 \rfloor) + \lceil n/2 \rceil \;\&\; W(1) = 0$$

وللتبسيط (for simplicity) نفرض أن n هى إحدى قوى (power of) 2. وبناءً عليه إذا قمنا بفك بعض الحدود (expanding a few terms) فمن السهل أن نستنتج أن

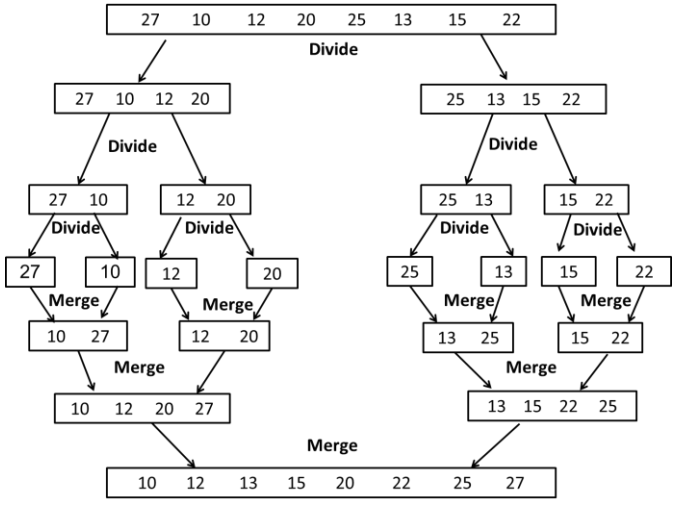
$$W(n) = 2^k\, W(n/2^k) + \sum_{i=1}^{k} \frac{n}{2}$$

وإذا فرضنا أن k = lg n، فإننا نصل الى النتيجة $W(n) = \frac{n}{2}\lg n$.

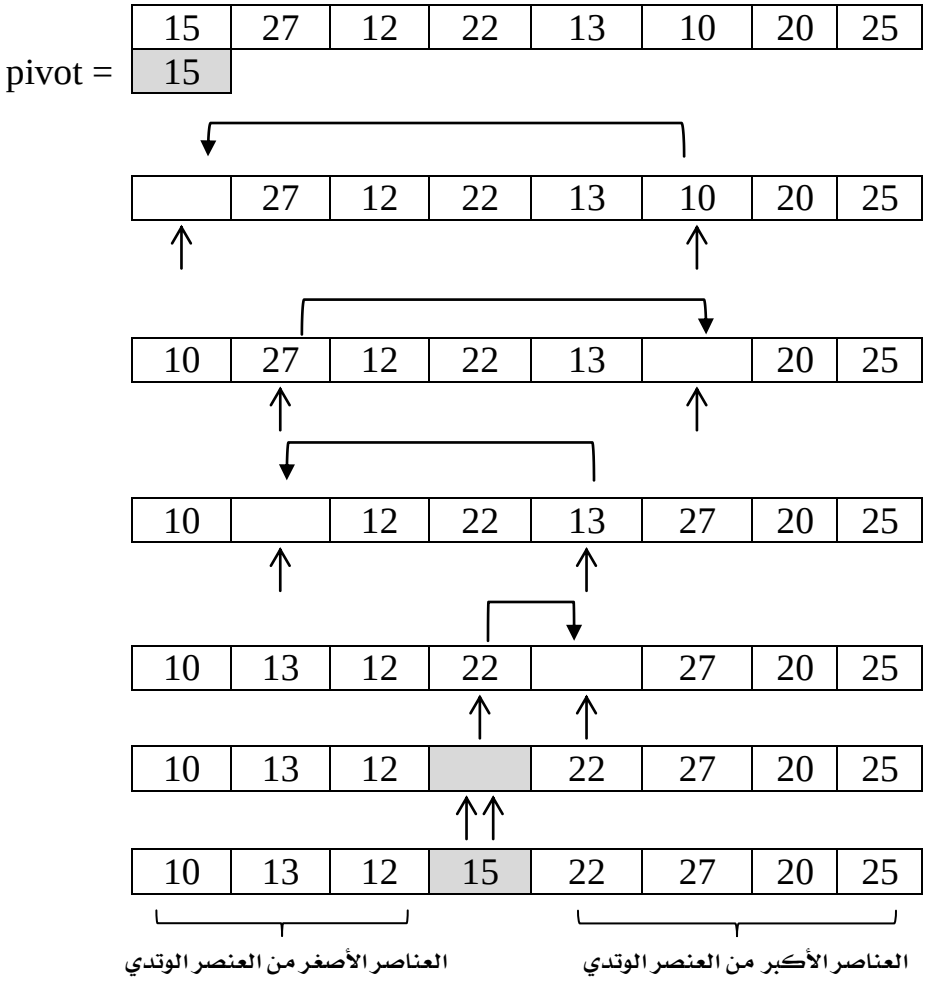
5-3) يتم اجراء المقارنات فى خوارزمية التجزئة partition الخوارزمية 3-3، وفعليا فى برامجها الجزئية الروتينية (its subroutines). وفى كل عقدة داخلية (internal node) فى شجرة القرار التالية المؤشر الثانى (second index) هو مؤشر العنصر الوتدى.



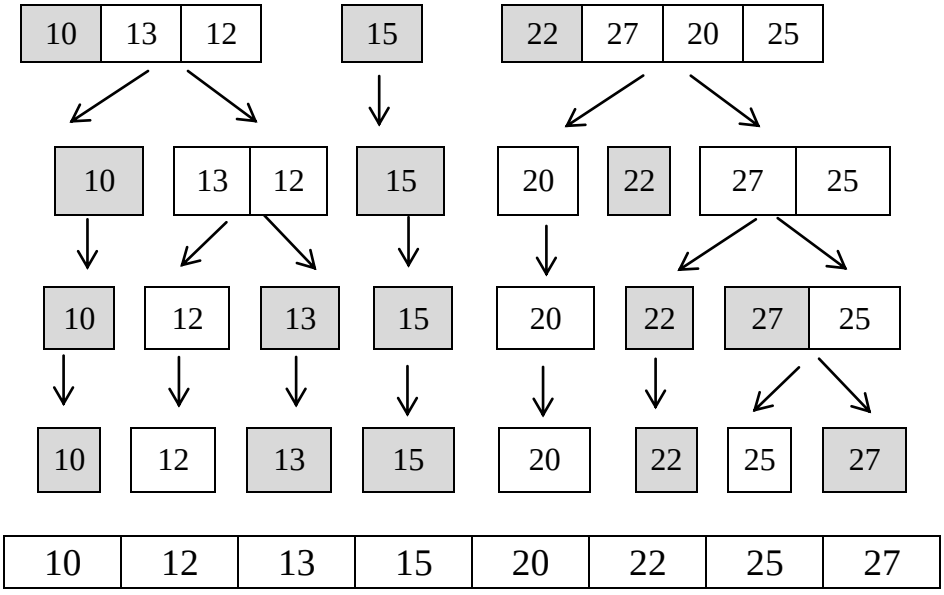
(6-3)



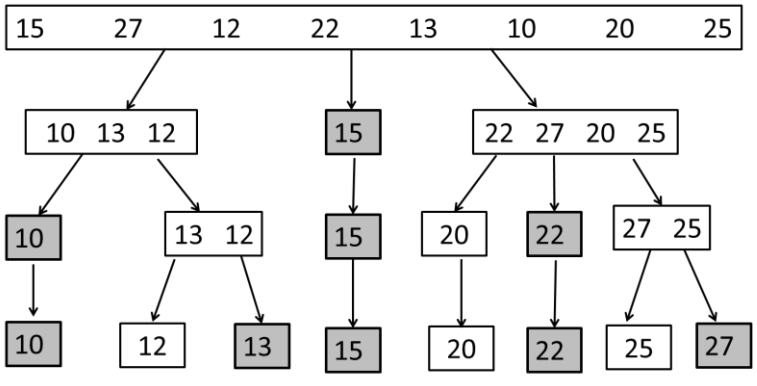
7-3) نجزئ (partition) المنظومة بحيث أن جميع العناصر الأصغر من العنصر الوتدى تكون إلى يساره، وجميع العناصر الأكبر منه تكون إلى يمينه:



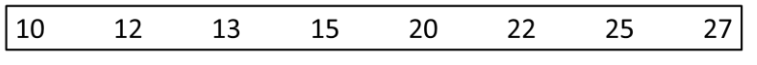
ثم نقوم بترتيب (sorting) المنظومتين الجزئيتين بالطريقة نفسها، ونترك للقارئ الكريم تتبع باقي الخطوات خطوة خطوة لنصل إلى:



أى أننا حصلنا على الترتيبات التالية للعناصر، حيث المنظومات الجزئية (subarrays) محاطة بمستطيلات / بمربعات، بينما العناصر الوتدية متروكة بمفردها ومظللة.



وهكذا تكون عناصر المنظومة النهائية المرتبة هي:



3-8- أ) عندما تكون جميع عناصر المنظومة A لها القيمة نفسها نلاحظ أن شرط المقارنة فى السطر 4 فى الإجراء PARTITION، أى الشرط (if (A[j] ≤ x متحقق دائماً، ولذلك فإن i تتم زيادتها (incremented) فى كل تكرير (iteration). وحيث أنه فى البداية i = p - 1 وأن قيمة i + 1 هى التى تعاد (returned) فإن القيمة المعادة هى r - 1.

ب) كى نجعل الإجراء PARTITION يعيد القيمة (p + r)/2 عندما تكون جميع عناصر المنظومة متساوية القيمة، نقوم ببساطة بتعديل الخوارزمية لتتحقق من هذه الحالة صراحة (to check for this case explicitly).

3-9) زمن تشغيل الإجراء PARTITION هو $\Theta(n)$ لأن كل تكرير (each iteration) فى عروة for يشتمل على عدد ثابت من العمليات (a constant number of operations)، والعدد الإجمالى للتكريرات هو $\Theta(n)$.

3-10) كى نجعل الخوارزمية QUICKSORT تقوم بترتيب العناصر ترتيباً غير تزايدى، نستبدل بمؤثر المقارنة ≤ فى السطر 4 فى الإجراء PARTITION المؤثر ≥.

3-11) عندما تكون جميع عناصر المنظومة A متساوية القيمة، فإننا نرى - من السؤال 3-8- أ) - أن القيمة المعادة من كل استدعاء للإجراء PARTITION (A, p, r) هى r - 1، مما يؤدى إلى

التجزئة فى أسوأ حالة (worst–case partitioning). وبالتالى فمن السهل أن نرى أن زمن التشغيل الكلى هو $\Theta(n^2)$.

3-12) عندما تكون عناصر المنظومة A متمايزة ومرتبة ترتيباً تنازلياً، فكما رأينا فى السؤال السابق يكون لدينا تجزئة فى أسوأ حالة (worst–case partitioning). وهكذا فإن زمن التشغيل $\Theta(n^2)$ مرة أخرى.

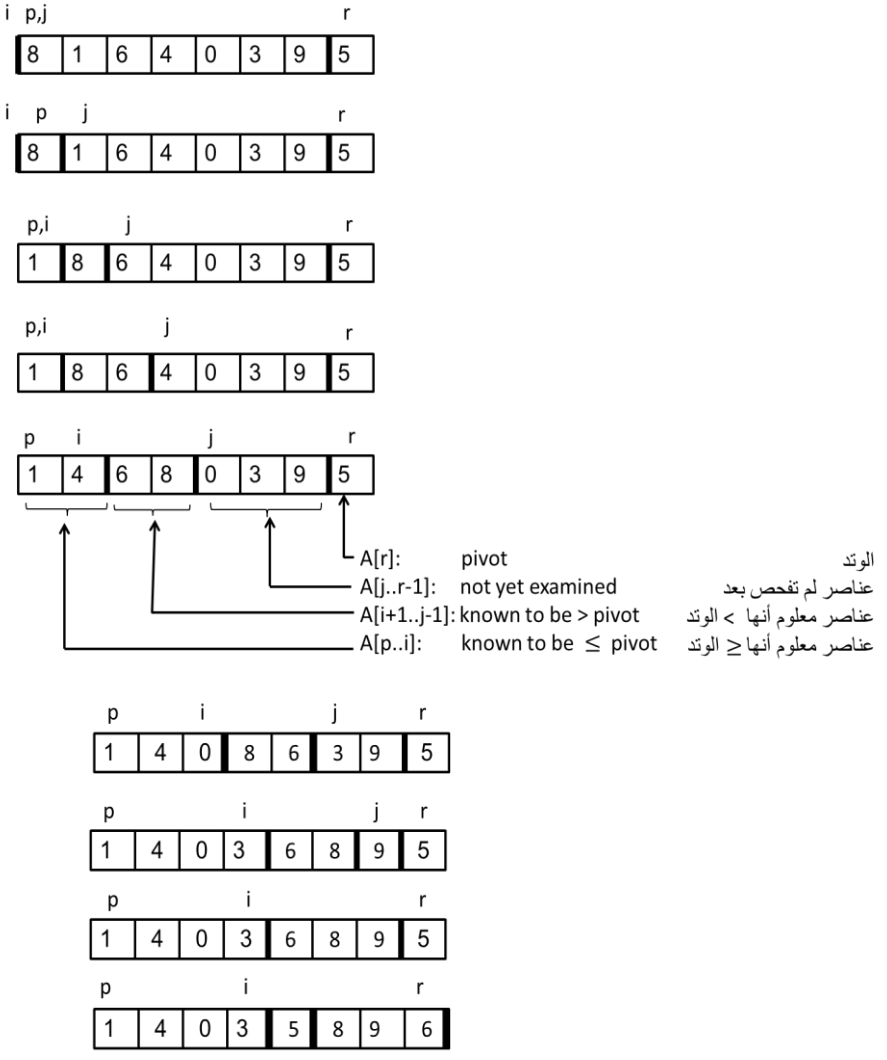
نلاحظ أن الإجراء PARTITION يجرى "تجزئة فى أسوأ حالة" (worst–case partitioning) عندما تكون العناصر مرتبة تنازليا. وهو يقلل (reduces) سعة المنظومة الجزئية (size of the subarray) قيد الاعتبار بواحد فقط فى كل خطوة، مما يؤدى إلى زمن تشغيل $\Theta(n^2)$.

وبصورة خاصة فإن الإجراء PARTITION إذا أعطى منظومة جزئية A[p .. r] (subarray) من عناصر متمايزة مرتبة تنازليا فإنه ينتج قسماً/جزءًا فارغا فى A[p .. q-1] (an empty partition in)، ويضع العنصر الوتدى {الموجود أصلا فى A[r]} فى A[p]، وينتج قسماً/جزءًا A[p+1..r] (a partition) يقل عن A[p .. r] بعنصر واحد فقط. والعلاقة الارتدادية (recurrence) لخوارزمية Quicksort تصبح:

$$T(n) = T(n-1) + \Theta(n)$$

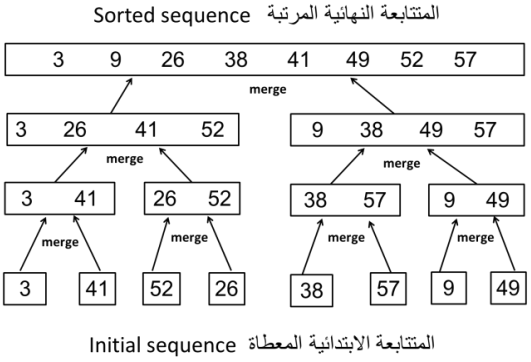
وهذه حلها هو $T(n) = \Theta(n^2)$

(i -13-3



ملاحظة: المؤشر j يختفى لأنه لم تعد هناك حاجة إليه بمجرد الخروج من عروة for.

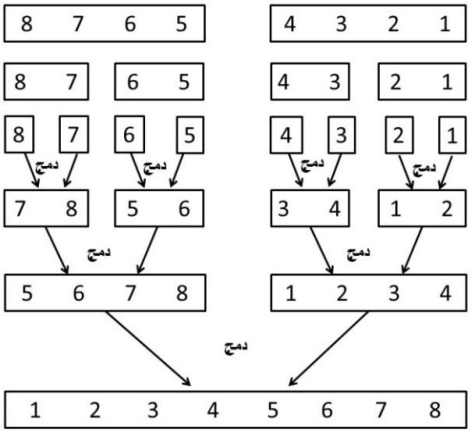
(14-3



15-3- أ) تحدث أسوأ حالة عندما تكون المفاتيح مرتبة تنازلياً، حيث تنفذ عروة for لكل قيمة من قيم numPairs تنازليا من $n-1$ إلى 1. وعدد مقارنات المفاتيح هو $n(n-1)/2$.

ب) إذا كانت المفاتيح مرتبة فعلا، فإن عروة for تنفذ مرة واحدة تجرى خلالها $n-1$ مقارنة، ولا تقوم بإجراء أي تبديل (no interchanges).

16-3- أ) خوارزمية الترتيب بالدمج:



ب) خوارزمية الترتيب الفقاعى:

8	7	6	5	4	3	2	1
7	6	5	4	3	2	1	2
6	5	4	3	2	1	3	3
5	4	3	2	1	4	4	4
4	3	2	1	5	5	5	5
3	2	1	6	6	6	6	6
2	1	7	7	7	7	7	7
1	8	8	8	8	8	8	8
المنظومة المعطاة	المرحلة الأولى	المرحلة الثانية	المرحلة الثالثة	المرحلة الرابعة	المرحلة الخامسة	المرحلة السادسة	المنظومة المرتبة

17-3- أ) نظراً لأن الكومة هى شجرة ثنائية كاملة تقريباً (an almost-complete binary tree)، [كاملة عند جميع المستويات، ربما عدا (except possibly) أدنى مستوى]، فإن:

أكبر عدد من العناصر فى كومة ارتفاعها h هو: $2^{h+1}-1$ [ويتحقق عندما تكون الكومة شجرة ثنائية كاملة/تامة (a complete binary tree)].

وأقل عدد من العناصر فى كومة ارتفاعها h هو: $2^h = 2^h-1+1$ [ويتحقق عندما يحتوى أدنى مستوى (lowest level) فى الكومة على عنصر واحد فقط، وتكون باقى المستويات كاملة (complete)].

ب) من الجزء أ) نرى أن

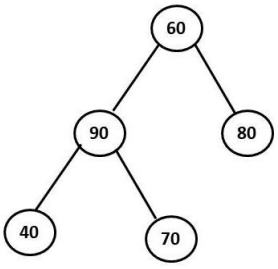
$$\text{min. \# of elements} \le n \le \text{max. \# of elements}$$

$$\Rightarrow 2^h \le n \le 2^{h+1}-1 < 2^{h+1}$$

$$\Rightarrow h \le \lg n < h+1$$

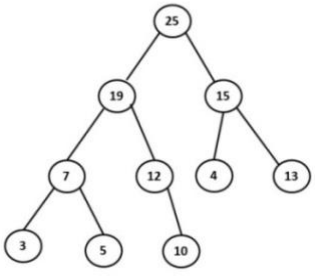
$$\Rightarrow h = \lfloor \lg n \rfloor \qquad \text{[نظراً لأن h عدد صحيح (integer)]}$$

3-18- أ) (i) بنية كومة المنظومة A:



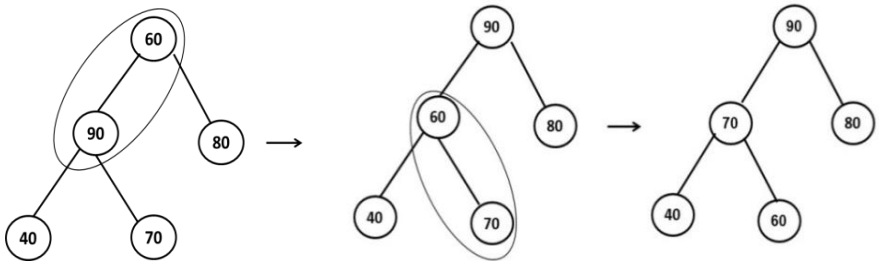
لا، هذه ليست كومة نظراً لأن العنصر/الرأس 60 (vertex) أصغر من كل من ابنيه (two children).

(ii) بنية كومة المنظومة B:

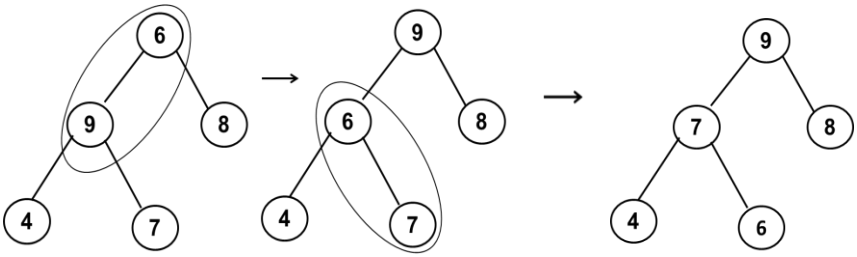


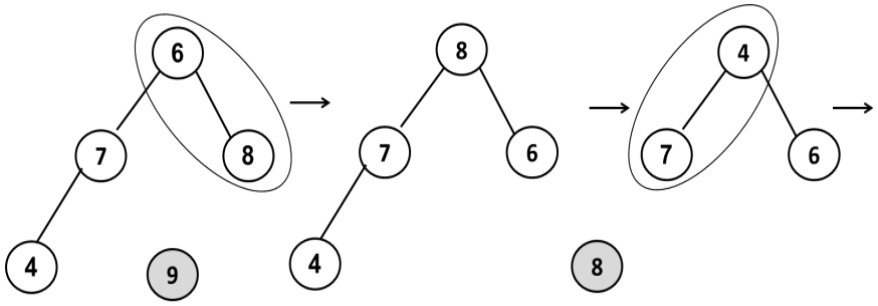
نعم، هذه كومة.

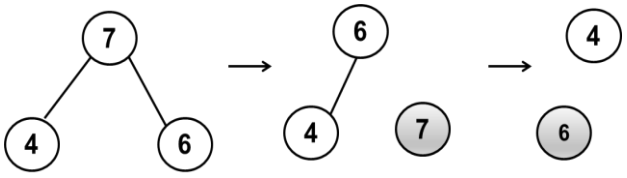
ب) نحول بنية كومة القائمة A إلى كومة كما يلى:



(19-3

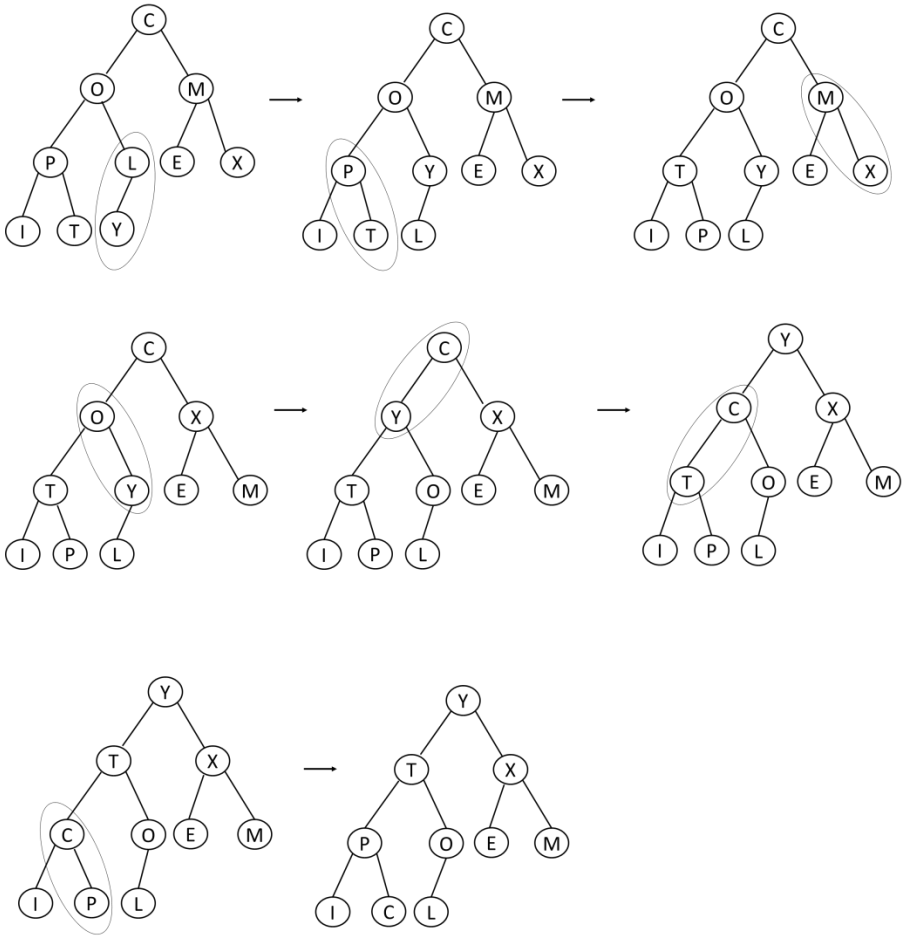






4	6	7	8	9

3-20- أ)



المنظومة (التى تحتوى الكومة) بعد مرحلة بناء الكومة (مبتدئين عند المؤشر1) تصبح:

Y	T	X	P	O	E	M	I	C	L

ب) نحتاج إلى 14 مقارنة بين المفاتيح.

3-21- أ) i) فى أسوأ حالة:

الترتيب بالدمج.

الترتيب بالكومة.

الترتيب بالإدخال، والترتيب السريع.

ii) في المتوسط: الترتيب السريع، والترتيب بالدمج، والترتيب بالكومة

الترتيب بالإدخال.

ب) ترتيب الخوارزميات من حيث الحيز المستخدم من الأقل إلى الأعلى:

الترتيب بالإدخال، والترتيب بالكومة (فى موضعه)

الترتيب السريع (حيز إضافى متناسب مع log n)

الترتيب بالدمج (حيز إضافى متناسب مع n)

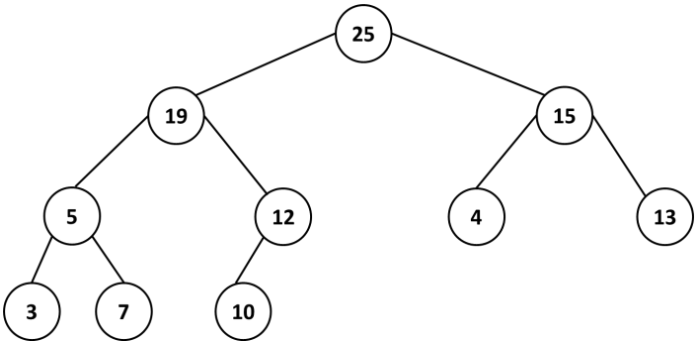
3-22) نفرض أن هذا الإدعاء خاطئ، أى أنه توجد شجرة فرعية جذرها ليس أكبر عنصر فى هذه الشجرة الفرعية. معنى هذا أن أكبر عنصر موجود فى موضع ما آخر فى الشجرة الفرعية، وربما فى أكثر من موضع. نفرض أن m هو المؤشر الذى يظهر عنده أكبر عنصر (أصغر مؤشر إذا ظهر أكبر عنصر أكثر من مرة). نظراً لأن أكبر عنصر ليس عند جذر الشجرة الفرعية، فإن العقدة m (node) لها والد (parent). وحيث أن مؤشر والد أى عقدة يكون أقل من مؤشر العقدة، و m كانت أصغر مؤشر لأكبر قيمة، فإن A[PARENT(m)] < A[m]. ولكن بخاصية الكومة

التكبيرية (max-heap property) يجب أن يتحقق لدينا $A[PARENT(m)] \geq A[m]$، وهذا يعنى أن فرضنا خاطئ، والادعاء المعطى صحيح.

23-3) أصغر عنصر فى أى كومة تكبيرية يقع دائما عند ورقة فى الشجرة بفرض أن جميع العناصر متمايزة.

24-3) لا. المتتابعة المعطاة ليست كومة تكبيرية.

25-3) نقوم أولاً بوضع العناصر فى شجرة ثنائية:



ومنها يتضح أن المنظومة المعطاة ليست كومة، حيث أن العنصر 5 أصغر من ابنه 7.

26-3- أ) 9

ب) نظرا لأن المفاتيح مرتبة تنازلياً، فإن الإجراء fixHeap لا يقوم إلا بإجراء مقارنة واحدة فقط أو مقارنتين بين المفاتيح (key comparisons) فى كل مرة يتم استدعاؤه، ولا يتم تحريك أى مفاتيح (no keys move). وإذا كانت n زوجية فإن

آخر عقدة داخلية (last internal node) يكون لها ابن واحد فقط (only one child)، وبالتالى فإن fixHeap سيجرى مقارنة واحدة فقط عند هذه العقدة. وبالنسبة لكل عقدة داخلية أخرى يتم إجراء مقارنتين. وأما إذا كانت n فردية فإن كلاً من العقد الداخلية [وعددها (n-1)/2] لها ابنان (2 children)، وبالتالى فإن fixHeap يقوم بإجراء مقارنتين لكل منها. وبناءً عليه فإن العدد الإجمالى للمقارنات فى أى من الحالتين هو n-1.

جـ) أحسن حالة (best case).

3-27- أ) نضيف العبارة

interchange E[1] and E[2]

ب) يتم الاستغناء عن استدعاء إضافى للإجراء fixHeap، ولكن عدد المقارنات لا ينقص.

❖❖❖

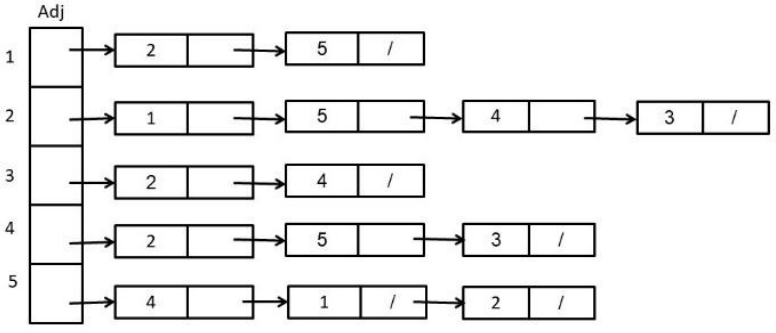
أجوبة تمرينات الفصل الرابع

(i -1-4

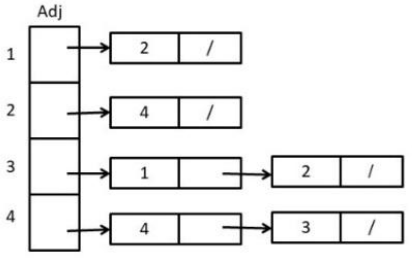
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

G1 G2

(ii

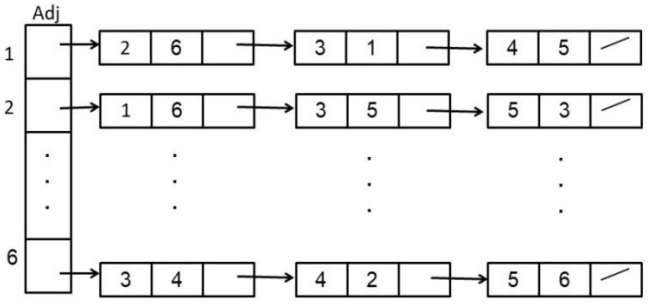


G1



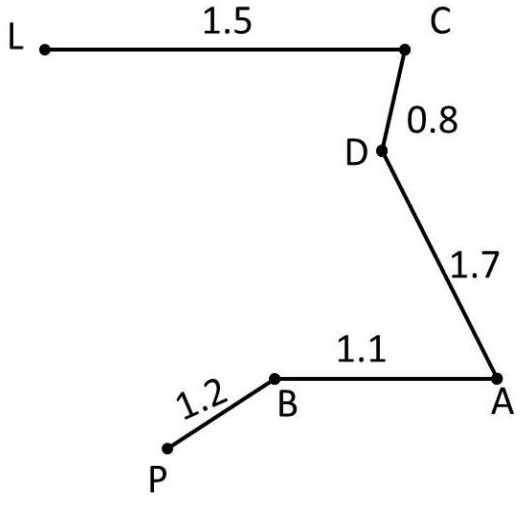
G2

2-4)



3-4-أ)

الحرف	الوزن	
DC	0.8	✓
BA	1.1	✓
PB	1.2	✓
LC	1.5	✓
LD	1.6	x
DA	1.7	✓
CA	1.8	x
PL	1.9	x
BD	2.2	x
LB	2.5	x
PD	2.9	x



ب) Total weight = 1.5+0.8+1.7+1.1+1.2 = 6.3

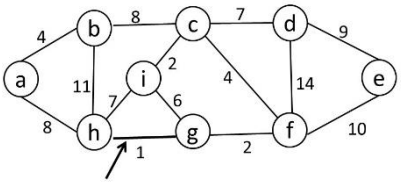
جـ) **نعم هذه الشجرة وحيدة، فخوارزمية "كروسكال" تعطى الشجرة المولِّدة الصغرى الوحيدة المبينة فى أ)، ولا يوجد حرفان لهما الوزن نفسه** (same weight).

(4-4

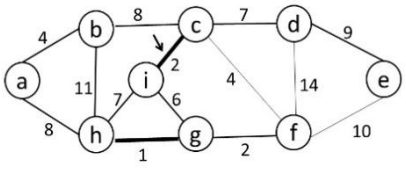
Iteration	Y	w	D[w]	D[v2]	D[v3]	D[v4]	D[v5]
Initial	{1}	-	-	10	∞	30	100
1	{1,2}	2	10		60	30	100
2	{1,2,4}	4	30		50		90
3	{1,2,4,3}	3	50				70
4	{1,2,4,3,5}=V	5	70				

4-5- أ) نقوم أولا بترتيب أحرف المخطط البياني تصاعديا بناء على أوزانها:

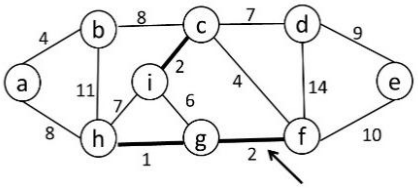
edge	gh	ci	gf	ab	cf	gi	cd	hi	ah	bc	de	ef	bh	Df
weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14



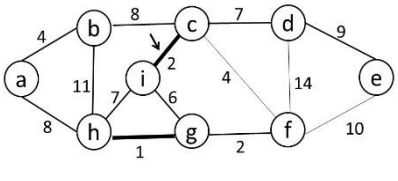
(a)



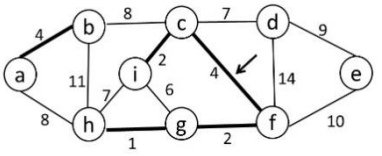
(b)



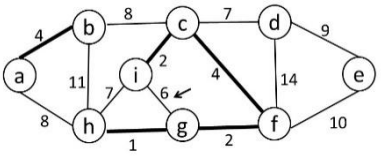
(c)



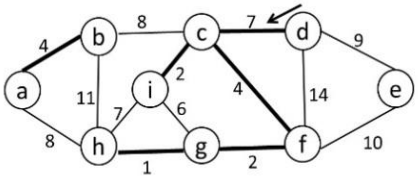
(d)



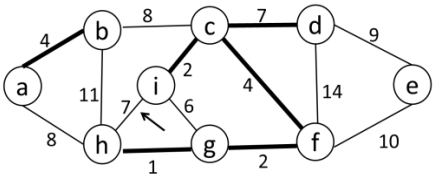
(e)



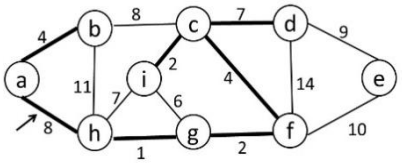
(f)



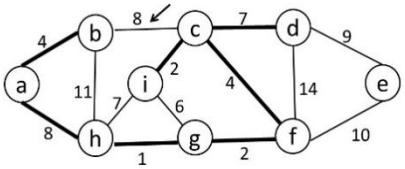
(g)



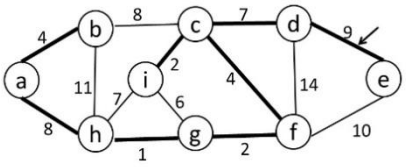
(h)



(i)



(j)



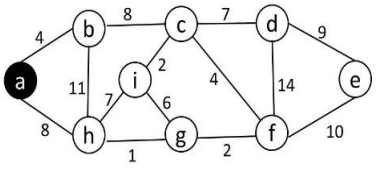
(k)

الأحرف المُظَلَلة هى الأحرف التى تُكَوِّن تباعاً الشجرة المولِّدة الصغرى (MST). والسهم فى أي خطوة يشير إلى الحرف الذى ندرس إضافته إلى الشجرة MST فى تلك الخطوة بناءً على دوره فى الترتيب التصاعدى للأوزان. فإن أدى الحرف إلى تكوين دورة (cycle) فإنه لا يضاف ونتجاهله.

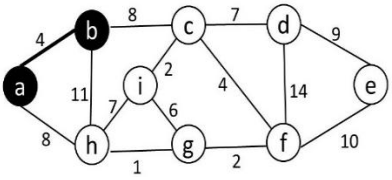
ب) Total weight =1+2+2+4+4+7+8+9 = 37

جـ) **هذه الشجرة MST ليست وحيدة، حيث نلاحظ أنه يمكن استبدال الحرف (b,c) بالحرف (a,h) - وكلاهما وزنه 8- ويظل الوزن الكلى للشجرة 37.**

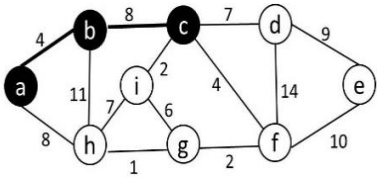
(6-4



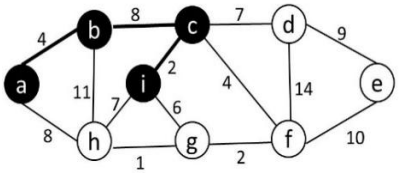
(a)



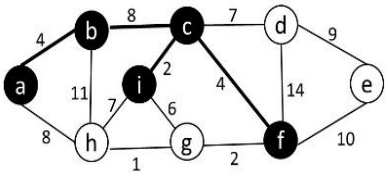
(b)



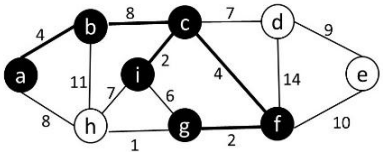
(c)



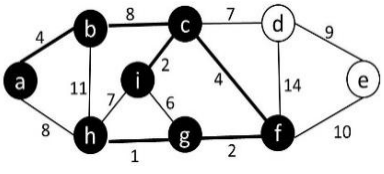
(d)



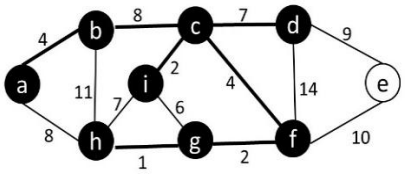
(e)



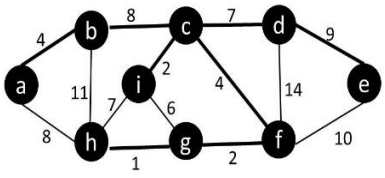
(f)



(g)



(h)



(i)

الأحرف المُظَلَّلة هى الأحرف التى تُكَوِّن تباعاً الشجرة المولِّدة الصغرى (MST). ويلاحظ أنه فى الخطوة الثانية (c) مثلا يمكن للخوارزمية أن تختار (choose) بين إضافة الحرف (b,c) أو الحرف (a,h) إلى الشجرة (MST).

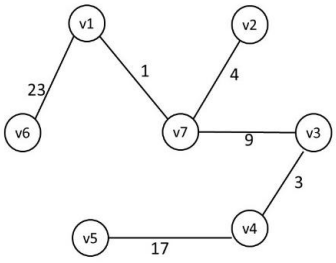
4-7) فيما يلى قائمة بأحرف المخطط البيانى G مرتبا ترتيبا تصاعديا بناءً على أوزانها.

Edge	Cost	Edge	Cost
(v1,v7)	1	(v4,v5)	17
(v3,v4)	3	(v1,v2)	20
(v2,v7)	4	(v1,v6)	23
(v3,v7)	9	(v5,v7)	25
(v2,v3)	15	(v5,v6)	28
(v4,v7)	16	(v6,v7)	36

الجدول التالى يوضِّح خطوات تطبيق خوارزمية "كروسكال" عن طريق تتبع وفحص الأحرف المرتَّبة حرفا حرفا لإضافته (Add) إلى الشجرة المولِّدة الصغرى التى ننشئها، أو لعدم إضافته (Reject).

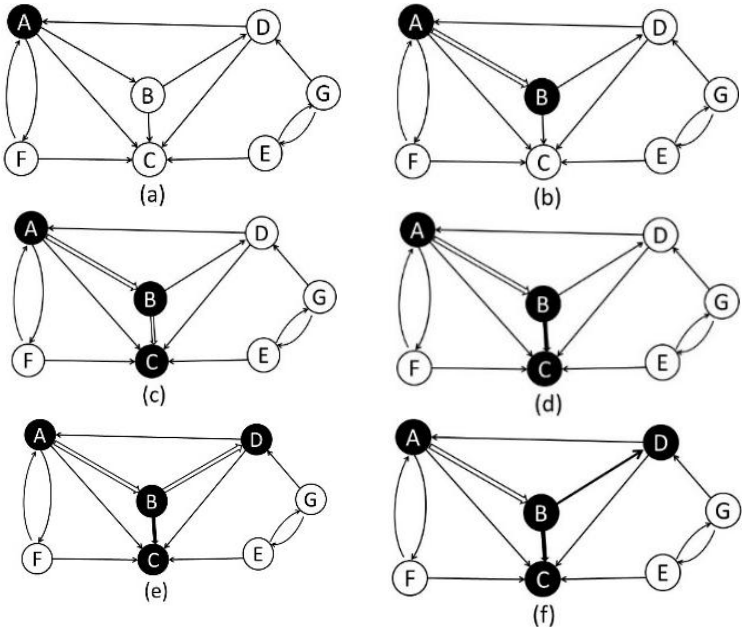
Edge	Action	Sets in connected components
(v1,v7)	Add	{v1,v7},{v2},{v3},{v4},{v5},{v6}
(v3,v4)	Add	{v1,v7},{v2},{v3,v4},{v5},{v6}
(v2,v7)	Add	{v1,v2,v7},{v3,v4},{v5},{v6}
(v3,v7)	Add	{v1,v2,v3,v4,v7},{v5},{v6}
(v2,v3)	Reject	
(v4,v7)	Reject	
(v4,v5)	Add	{v1,v2,v3,v4,v5,v7},{v6}
(v1,v2)	Reject	{v1,v2,v3,v4,v5,v6,v7}

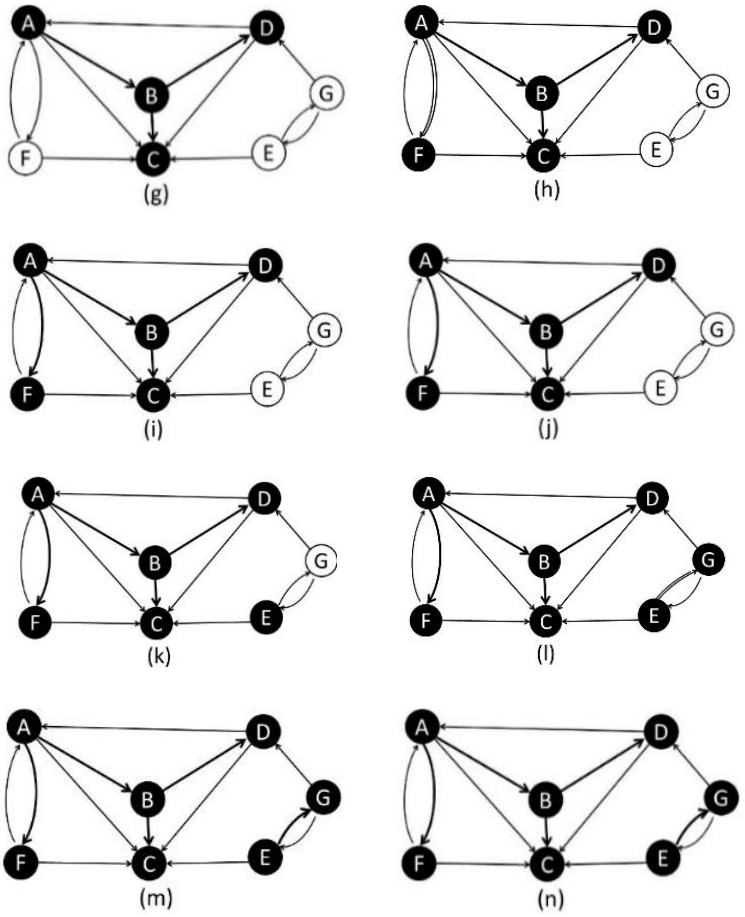
والشكل التالى يعطى الشجرة المولِّدة الصغرى (MST) التى أنشأناها من الحروف المضافة.



8-4) إذا كان v,w أي رأسين مختلفين (two distinct vertices) فى المخطط G، وكان هناك مسار (a path) من v إلى w، فيجب أن يكون هناك حرف (v,w) فى G.

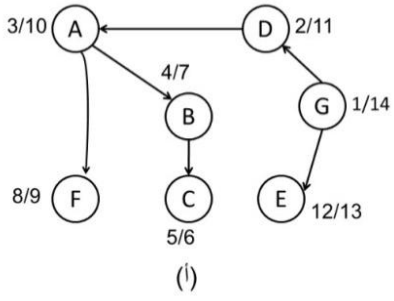
9-4)



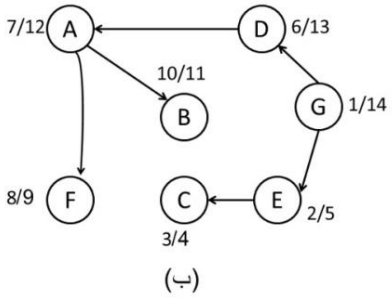


الخطوط المزدوجة (double lines) هى أحرف شجرة (tree edges) لم نرجع عبرها بعد (have not yet been backtracked over)، بينما الخطوط السميكة (heavy lines) هى أحرف شجرة تم الرجوع عبرها. ونلاحظ أن الغابة المولدة حصلنا عليها بإنشاء شجرتين بالبحث بالعمق أولا (two depth-first search trees).

4-10- i) الشكل التالى يعطى الشجرة DFS المطلوبة فى كل من أ) و ب)، وفى الشكل تظهر أرقام ترتيب اكتشاف أي رأس من رؤوس المخطط (زيارتها أول مرة) وزيارتها عند الانتهاء من إكمال البحث عبر جميع المسارات التى تبدأ عند هذه الرأس [discovery and finishing (d/f) times]، وهذه الأرقام غير مطلوبة فى السؤال، ولكنها تساعد فى توضيح خطوات إنشاء الشجرة DFS.

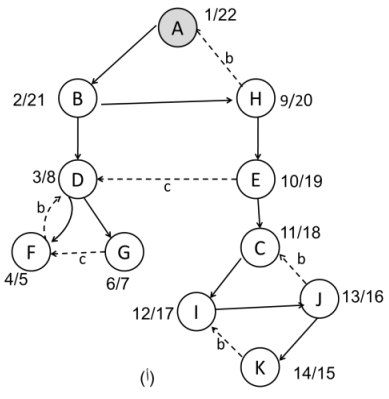


(أ)

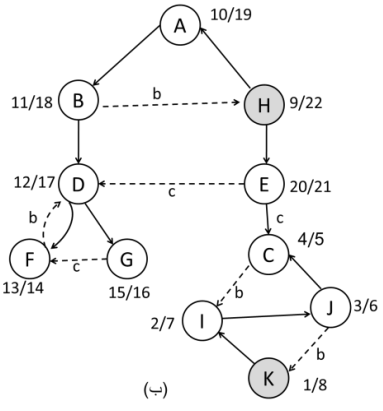


(ب)

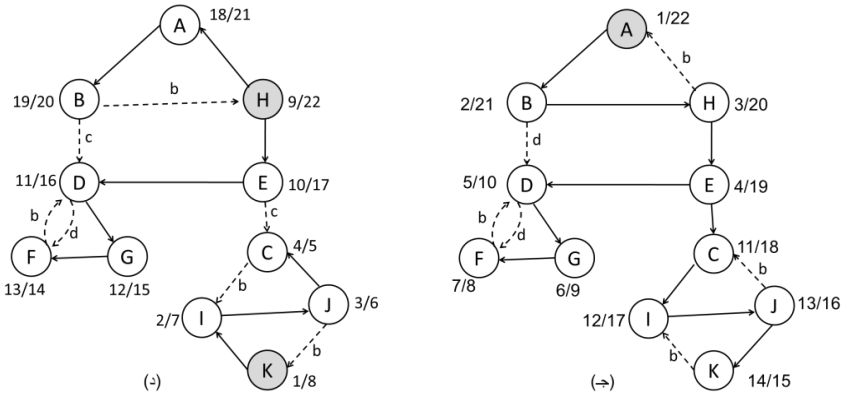
4-11- i)



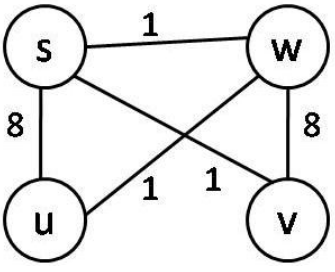
(أ)



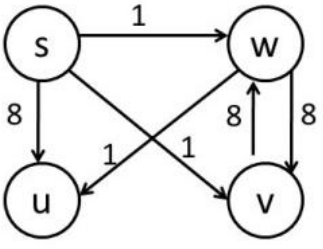
(ب)



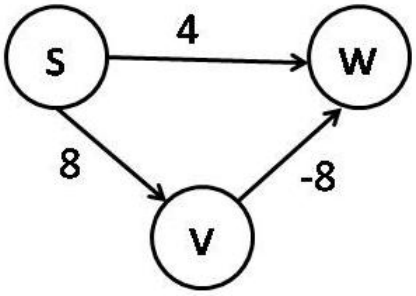
4-12) ابدأ عند الرأس S.



4-13) الرأس المصدر هى S.

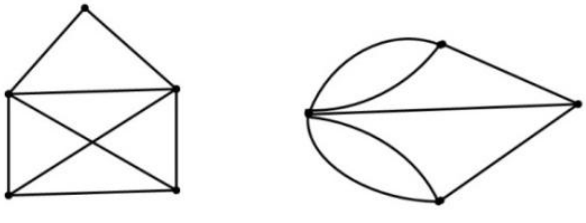


14-4) خوارزمية ديجسترا لإيجاد أقصر مسار لا تعمل بصورة صحيحة فى حالة الأوزان السالبة. انظر مثلا إلى المخطط البيانى فى الشكل التالى:



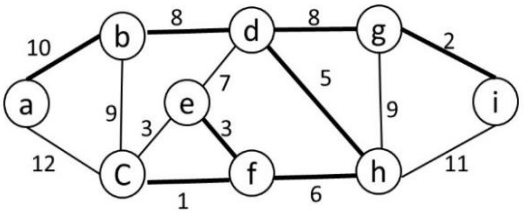
باعتبار S هى رأس البداية ستقوم الخوارزمية بجذب الرأس W إلى الشجرة بالمسار الذى طوله 4، ثم تجذب الرأس V إلى الشجرة، وبالتالى فإن الحرف ذا الوزن السالب - والذى يعطى مسارا أقصر إلى w (a shorter path to) - لن يصبح إطلاقا حرفا مُرشَّحاً (never becomes a candidate).

17-4- ب) بالنسبة للمخطط البياني G1 فى الشكل التالي ترتيب زيارة الرؤوس فى خوارزمية ديجسترا لإيجاد أقصر مسار هو نفسه ترتيب زيارتها فى خوارزمية البحث بالعمق أولا، وكذلك فى خوارزمية البحث بالعرض أولا. بينما بالنسبة للمخطط G2 الترتيب مختلف عن الترتيب فى أي منهما.



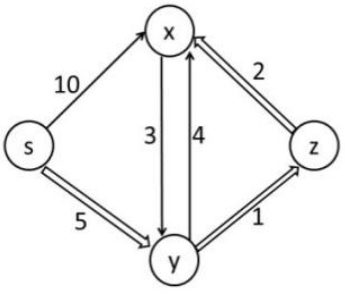
G1 G2

(18-4



الأحرف المظللة هي أحرف الشجرة المولدة الصغرى MST. إذا استبدلنا بالحرف (e,f) الحرف (c,e) والذى له الوزن نفسه (3)، فإننا نحصل علي شجرة مولدة صغرى مختلفة، بمعنى أن المخطط البيانى المعطى له أكثر من شجرة مولدة صغرى (MST) واحدة.

(19-4



ترتيب إضافة الرؤوس: s, y, z, x.

(20-4 **إذا أُعطينا تمثيل مخطط بقائمة تجاور فيمكننا حساب كل من درجة الخروج ودرجة الدخول لكل رأس بسهولة بإذن الله فى وقت يساوى** O(m+n).

(21-4 $G^T \equiv$ **مدوَّر أي مخطط بياني موجه** G **يمكن حسابه كما يلي: إذا كان المخطط مُمثَّلاً بمصفوفة تجاور** A**، فإننا ببساطة نحسب مدوَّر**

المصفوفة A^T (transpose) فى زمن قدرة $O(n^2)$. وإذا كان المخطط ممثَّلاً بقائمة تجاور فإن مسحاً واحدا عبر هذه القائمة (a single scan through this list) يكفي لإنشاء المدور (to construct the transpose). والوقت المُستغرَق هو $O(m+n)$.

22-4) مربع أي مخطط بياني يمكن حسابه كما يلي: إذا كان المخطط ممثَّلاً بمصفوفة تجاور A، فنحسب ببساطة حاصل الضرب (the product) A^2، حيث الضرب والجمع نستبدل بهما عمليتى AND و OR، أي عمليتي "و" و"أو" [multiplication and addition are exchanged by and'ing and or'ing]. واستخدام الخوارزمية البسيطة (simple algorithm) يؤدي إلى وقت تشغيل قدره $O(n^3)$.

ملاحظة: هناك خوارزمية تُدعي"خوارزمية ستراسن" (Stassen's algorithm) تُحَسِّن الحد إلى $O(n^{log7})=O(n^{2.81})$. وإذا كنا نستخدم التمثيل بقائمة تجاور، فيمكننا ببساطة إلحاق قوائم مع إلغاء أي مكررات (append lists eliminating duplicates). وبفرض أن القوائم مرتبة (lists are sorted) يمكننا اجراء الحسابات علي النحو التالي:

لكل رأسٍ v فى قائمةٍ ما (for each node v in some list) استبدل Adj[v] بالرأس v وأدمج هذه فى القائمة (merge this into the list). وحيث أن أي قائمة يمكن أن يكون طولها علي الأكثر n،

فأي عملية دمج (merge operation) تستغرق على الأكثر زمنا قدره $O(n)$. وبالتالي فإن زمن التشغيل الكلى يصبح $O((m+n)n) = O(mn+n^2)$.

23-4) إذا قمنا بتشغيل خوارزمية البحث بالعرض أولا علي مخطط بياني مُمثَّل بمصفوفة تجاور فإن الزمن المُستغرَق لمسح المجاورات (the time used scanning for neighbor) يمكن أن يزداد إلى $O(n^2)$ مما يؤدي إلى زمن تشغيل كلى قدره $O(n^2+n) = O(n^2)$.

24-4) إذا أُعطينا شجرة مولدة صغرى T، فإننا نود أن نرتب الأحرف فى خوارزمية كروسكال بحيث أن الخوارزمية تعيد T. لأي حرف e فى الشجرة T نتأكد ببساطة من أن هذا الحرف يسبق (precedes) أي حرف آخر ليس فى T وزنه $w(e)$.

❖❖❖

دليل المطلحات العلمية

فيما يلى قائمة بالمصطلحات العلمية التي وردت في الكتاب باللغتين العربية والإنجليزية، والقائمة مرتبة أبجدياً حسب حروف اللغة الإنجليزية.

المصطلح الإنجليزى	المصطلح العربى
Abstract	مُجَرّد
Abstraction	تجريد
Accelerated form	صيغة مُعَجّلَة
Activation	نشاط
Acyclic	لادَوْرَوِىّ
Addition	جَمْع / إضافة
Adjacency	تَجَاوُر
Adjacency list	قائمة التجاور
Adjacency matrix	مصفوفة التجاور
Adjacent	مجاور
Algorithm	خوارزمية
Allocate	يُعيّن موضع
Analysis	تحليل
Analytical	تحليلي
Ancestor	سَلَف / جَدّ / من آباء وأجداد
Antimonotonic function	دالة غير رتيبة
Antisymmetric	قُطْرى التناظر

المصطلح الإنجليزى	المصطلح العربى
Application	تطبيق
Approximation	تقريب
Argument	حُجّة، وسيط
Arithmetic average	متوسط حسابى
Arrange	يُرَتِّب
Array	منظومة
Arrow	سهم
Asymptotic	مُقارِب
Automatical	تلقائى
Average behavior	السلوك المتوسط
Average case	الحالة المتوسطة
Average complexity	درجة التعقيد المتوسطة
Average value	القيمة المتوسطة
Back edge	حرف خَلفى / عَكْسِي
Backwards	الاتجاه العكسى من الخلف إلى الأمام
Barrier	حاجز
BFS (breadth first search)	البحث بالعرض أولا
Binary	ثنائى
Binary relation	علاقة ثنائية
Binary search algorithm	خوارزمية البحث الثنائي
Binary tree	شجرة ثنائية

المصطلح الإنجليزى	المصطلح العربى
Bookkeeping	تَتَبُّع
Boolean	بُولي
Bounded above	محدود من أعلى
Branch	فرع
Breadth first search (BFS)	البحث بالعرض أولا
Breadth first traversal	الاجتياز بالعرض أولا
Bubble sort algorithm	خوارزمية الترتيب الفقاعى
Call	استدعاء
Candidate edge	حرف مُرَشّح
Capacity	سِعَة
Cardinality (of a set)	عدد عناصر (مجموعة)
Cartesian product	حاصل ضرب كارتيزى / تبادلى
Category	فئة
Ceiling function	دالة السقف
Cell	خلية
Characteristics	خواص / خصائص
Child	ابن
Chop off	يقطع
Clarity	الوضوح
Class	طبقة / طائفة
Classify	يُصَنِّف

المصطلح الإنجليزى	المصطلح العربى
Combine	يجمع
Comparison	مقارنة
Complete binary tree	شجرة ثنائية كاملة
Complete graph	مخطط بيانى كامل
Complexity	درجة تعقيد
Conclusion	استنتاج
Conditional Probability	احتمال شَرْطي / مشروط
Connected graph	مخطط بيانى متصل
Connectivity	اتصالية
Consecutive	متعاقب
Construct	يَبْنى
Construction	إنشاء / بناء
Continuous	متصل
Contradiction	تناقض
Contrapositive	المكافئ العَكْسى
Convex function	دالة مُحَدّبة
Corollary	نتيجة
Correctness	الصحة
Cost	تكاليف
Criteria	معايير
Criterion	معيار

المصطلح الإنجليزى	المصطلح العربى
Cross edge	حرف مستعرض / متعارض / متقاطع
Current	حالى
Cycle	دَوْرَة
DAG (directed acyclic graph)	مخطط بيانى مُوَجَّه لادَوْرَوِىّ
Dashed line	خط متقطع
Data structures	بنى المعطيات / هياكل البيانات
De Morgan’s laws	قوانين "دى مورجان"
Debug	يصحح الأخطاء
Decision	قرار
Decision tree	شجرة قرار/ شجرة حَسم
Definiteness	تحديد
Delete	يحذف
Depth first search (DFS)	البحث بالعُمق أولا
Depth first traversal	الاجتياز بالعُمق أولا
Derivation	اشتقاق / استنتاج
Derivative	مُشْتَقَّة
Descendant	سليل/ حفيد/ من ذرية/ من أبناء وأحفاد
Descendant edge	حرف نازل / هابط/ منحدر من سلف
Description	وصف
Design	تصميم
Detachment rule	قاعدة الفَصْل

المصطلح الإنجليزى	المصطلح العربى
DFS (depth first search)	البحث بالعُمق أولا
Diagram	رسم بيانى
Dijkstra's algorithm	خوارزمية ديجسترا
Directed edge	حرف مُوَجَّه
Directed graph	مخطط بيانى مُوَجَّه
Discard	يهمل
Distinct	متباين
Divide-and-conquer	قَسِّمْ وتَغَلَّبْ / فَرِّق تَسُدْ
Duplicates	قيم مكررة
Edge	حرف
Efficiency	كفاءة
Efficient	كُفء / ذو كفاءة عالية
Elementary	بسيط / ابتدائى
Eliminate	يلغى
Encounter	يواجه / يقابل
Entry	عنصر
Equally likely	متساوية الاحتمالات
Equivalence	تكافؤ
Estimate	تقدير
Event	حَدَث
Exact	دقيق / مضبوط

المصطلح الإنجليزى	المصطلح العربى
Examine	يفحص
Execution time	وقت التنفيذ
Exhaust	يَسْتَنفذ
Expanded outline	مخطط مُوَسّع
Expectation	تَوَقُّع
Explore	يستكشف
Exponent	أُسّ
Exponential function	دالة أُسِّيّة
Extension	امتداد
External sort	ترتيب خارجى
Factor	عامل
Fail	يفشل
Fibonacci numbers	أعداد "فيبوناتشى"
Field	مجال
File	ملف
Finite	محدود
Flexibility	مرونة
Floor function	دالة الأرضية
Forest	غابة
Form	يكوّن
Formal	رَسْمي / شَكْلي

المصطلح الإنجليزى	المصطلح العربى
Forward edge	حرف أمامي / متجه إلى الأمام
Fraction	كَسْر
Frame	إطار
Free tree	شجرة حُرّة
Fringe	هُدّاب / حاشية / شراشيب / حافة
Function	دالة
Gap	فجوة
Generality	العمومية
Generalized	مُعَمّم
Geometric	هندسى
Global	شامل
Graph	مخطط بياني / رسم بياني
Greedy algorithm	خوارزمية جَشِعَة / طَمّاعة / شَرِهة
Growth	نُمُوّ / زيادة
Handle	يتعامل مع
Harmonic	توافقى
Head	رأس
Heap	كَوْمة
Heap property	خاصية الكَوْمة
Heap structure	بِنْية / هَيْكَل كَوْمة
Heapsort algorithm	خوارزمية الترتيب بالكَوْمة

المصطلح الإنجليزى	المصطلح العربى
Height	ارتفاع
High-level	عالى المستوى
Identifier	اسم تعريفى
Identity	متطابقة
Implementation	تنفيذ
Improve	يُحسِّن
In place	فى مكانها/ موضعها (الصحيح/ الملائم)
Incident upon	يقع على / يحدث عند
Increment	زيادة
Independence	استقلال
Index	مؤشِّر، فهرس
Indexed	مُؤَشّر
Inequality	متباينة
Inference	استدلال
Informal	غير رَسْمي / غير شَكْلي
Inherent complexity	درجة تعقيد ذاتية
Initialization	إبداء / إعطاء قيم ابتدائية / بُداءة
Input sequence	متتابعة الإدخال
Inputs	مُدْخلات
Insertion sort algorithm	خوارزمية الترتيب بالإدخال
Instructions	تعليمات

المصطلح الإنجليزى	المصطلح العربى
Integers	الأعداد الصحيحة
Interchange	تبديل
Intermediate	وَسَطى / مَرْحَلِى
Internal sort	ترتيب داخلى
Interpolation	استكمال
Inversion	انعكاس
Invocation	استدعاء
Iterative	تكرارى
Key	مفتاح
Key comparisons	مقارنات بين المفاتيح
Kruskal’s algorithm	خوارزمية "كروسكال"
L’Hopital’s rule	قاعدة "هوبيتال"
Label	اسم / عنوان
Labeled	مُعَنْوَن / مُرَقِّم
Leaf	ورقة (من شجرة)
Left Child	الابن الأيسر
Lemma	فَرَضيّة
Level	مستوى
Linear	خَطِّى
Linked list	قائمة مترابطة
Local variable	متغير محلى

المصطلح الإنجليزى	المصطلح العربى
Location	موقع
Logarithm	اللوغاريتم
Logic	منطق
Loop	عُرْوَة
Loser	خاسر
Lower bound	حَدْ سُفْلى
Lower limit	نهاية سُفْلى
Manipulate	يعالج
Matrix	مصفوفة
Max-heap	كومة تكبيرية
Max-heap property	خاصية الكومة التكبيرية
Maximal	أَعْظَمِي
Maximizing heap	كومة تكبيرية
Maximizing partial order tree	شجرة ترتيب جزئى تكبيرية
Maximum	القيمة العُظمى
Max-sort	الترتيب بالأكبر
Measure	مقياس
Median	أوسط
Memory	ذاكرة
Merge	يدمج
Merge algorithm	خوارزمية الدمج

المصطلح الإنجليزى	المصطلح العربى
Merge sort algorithm	خوارزمية الترتيب بالدمج
Min-heap	كومة تصغيرية
Min-heap property	خاصية الكومة التصغيرية
Minimum	قيمة صُغرى
Minimum spanning tree (MST)	شجرة مُولَدة صغرى
Model	نموذج
Modify	يُعَدِّل
Modus ponens rule	قاعدة "مودَس بونِنْز"
Monotonic function	دالة رتيبة
MST (minimum spanning tree)	شجرة مُولَدة صغرى
Multiple	مضاعف
Natural numbers	الأعداد الطبيعية
Node	عُقْدَة
Non iterative	غير تكرارى
Nondecreasing function	دالة غير متناقصة
Nonincreasing function	دالة غير متزايدة
Notation	اصطلاح
Numeric	عددى
One-to-one function	دالة واحد لواحد
Operation	عملية
Optimal	أمْثَل / مُثْلى

المصطلح الإنجليزى	المصطلح العربى
Optimality	الأمثلية
Optimization problems	مسائل الحلول المُثْلى
Order	ترتيب / رُتبة / مرتبة
Orientation	توجيه
Outline	مخطط عام
Outputs	مُخْرجات
Over approximation	تقريب عُلْوى
Overhead operations	عمليات إضافية
Paradigm	نموذج
Parameter	وسيط
Parent	والد
Partial order tree property	خاصية شجرة الترتيب الجزئى
Partition algorithm	خوارزمية التجزئة
Passes	تمريرات/ مسارات/ اجتيازات/ دورات
Path	مسار
Perform	يُنْجِز
Permutation	تبديل
Phase	مرحلة
Phrase	عبارة
Pivot	وَتَد
Pivot element	عنصر وتدى

المصطلح الإنجليزى	المصطلح العربى
Pointer	مؤشِّر
Polynomial	حدودية
Pop	يَرْفَع
Positive scaling	قياس موجب
Post order	لاحق الترتيب
Postconditions	شروط لاحقة
Power	قوة / أس
Powerful	عالى القدرة
Precision	دِقَّة
Preconditions	شروط سابقة
Preorder	سابق الترتيب
Prim’s algorithm	خوارزمية "بريم"
Priority queue	طابور أولوية
Probability theory	نظرية الاحتمالات
Procedure	إجراء
Processing	معالجة / تشغيل
Proper	مضبوط / حقيقى / فعلى / صحيح
Proportional	متناسب
Pseudocode	شبه كود / كود زائف
Push	يَدْفَع
Quadratic	تربيعى

المصطلح الإنجليزى	المصطلح العربى
Qualifier	قَيْد
Quicksort algorithm	خوارزمية الترتيب السريع
Random	عشوائى
Random access	وصول عشوائى
Range	مدى
Rate	مُعَدّل
Reachable	يمكن الوصول إليه
Real numbers	الأعداد الحقيقية
Recurrence	تكرار
Recursive	ارتدادى
Reference	مرجع / إسناد
Refine	يُحَسِّن
Reflexive	انعكاسي
Relation	علاقة
Relational operator	مؤثِّر علاقى
Restriction	قَيْد
Right Child	الابن الأيمن
Routine	برنامج روتينى
Rule of cases	قاعدة الحالات
Run	تنفيذ / تشغيل
Running time	وقت التنفيذ / التشغيل

المصطلح الإنجليزى	المصطلح العربى
Scan	يمسح
Search	البحث
Section	جُزء / قطاع
Segment	قطعة / شريحة
Separate	منفصل
Sequence	متتابعة
Sequential	تتابعى
Sequential search algorithm	خوارزمية البحث التتابعى
Series	متسلسلة
Set	مجموعة
Share	يشارك
Shift	إزاحة، يتزحزح
Shrink	ينكمش
Simple cycle	دورة بسيطة
Simple path	مسار بسيط
Simplicity	البساطة
Single-source	مصدر أحادى
Size	حجم / سعة
Skeleton procedure	إجراء هيكلى
Small sort algorithm	خوارزمية الترتيب الصغير
Solid line	خط متصل / مصمت

المصطلح العربى	المصطلح الإنجليزى
مُعَقَّد	Sophisticated
ترتيب / فَرْز	Sorting
الحَيِّز	Space
الحَيِّز المستخدم	Space usage
شجرة مُوَلِّدة	Spanning tree
هش / متناثر	Sparse
مُعَيّن	Specific
تسريع	Speedup
يقسم	Split
رَصّة	Stack
عبارة	Statement
صيغة "سترلنج"	Sterling's formula
تصادفى	Stochastic
الاستراتيجية	Strategy
متّصل بقوة / قَوِىُّ الاتصال	Strongly connected
برمجة مَبْنيّة / برمجة مُهَيْكَلَة	Structured programming
مخطط بيانى جزئى	Subgraph
مسألة جزئية	Subproblem
مدى جزئى	Subrange
برنامج فرعى	Subroutine
مؤشِّر سُفلى	Subscript

المصطلح الإنجليزى	المصطلح العربى
Subset	مجموعة جزئية
Subtree	شجرة جزئية
Success	نجاح
Summation	تجميع
Swap	يبدِّل
Syllogism rule	قاعدة القياس المنطقى
Symmetric	متماثل / متناظر
Tail	ذيل
Tail recursion	الارتداد الذّيْلي
Technique	طريقة
Term	حَدّ
Terminate	تتوقف
Three-way branch	ثلاثية التفرع
Tight	مُحْكَم
Time analysis	تحليل الزمن
Time limit	حَد زمني
Trade-off	مقايضة / مفاضلة
Transitive	متعدى
Transpose	المدوّر
Traversal	اجتياز
Tree edge	حرف شجرة

المصطلح الإنجليزى	المصطلح العربى
Under approximation	تقريب سُفْلى
Undirected	غير مُوَجَّه
Union	اتحاد
Unordered	غير مُرَتَّب
Updating	تحديث
Upper bound	حَد أعلى
Upper limit	نهاية عُليا
Vacancy	فراغ
Valid	صالح
Variations	تغيرات / تعديلات
Version	صيغة
Vertex	رأس
Vertices	رؤوس
Weight	وَزْن
Weighted average	متوسط مُوَزَّن
Weighted graph	مخطط بيانى مُوَزَّن
Width	عَرْض
Work	عمل / جُهد / شُغل
Workspace	حيّز للشُّغل
Worst case	أسوأ حالة

❖❖❖

المراجـــــع

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms (1974), Addison-wesley.
2. A. V. Aho, J. E. Hopcroft, J. D. Ullman, Data Structures and Algorithms (1983), Addison-wesley.
3. Amrinder Arora, Analysis and Design of Algorithms (2017), Cognella Academic Publishing.
4. Sara Baase, Allen Van Gelder, Computer Algorithms: Introduction to Design and Analysis 3rd Edition (1999), Addison-Wesley.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 3rd Edition (2009), MIT Press.
6. Himanshu B. Dave, Design and Analysis of Algorithms (2013), Pearson.
7. Anany Levitin, Introduction to the Design and Analysis of Algorithms 3rd Edition (2011), Pearson.
8. Jeffrey J. McConnell, Analysis of Algorithms: An Active Approach (2001), Jones and Bartlett.
9. V.V. Muniswamy, Design and Analysis of Algorithms (2009), I. K. International Publishing House.

10. Robert Sedgewick, Kevin Wayne, Algorithms 4th Edition (2011), Addison-wesley.
11. Steven S. Skiena, The Algorithm Design Manual 2nd Edition (2011), Spriger.
12. Rod Stephens, Essential Algorithms: A Practical Approach to Computer Algorithms 1st Edition (2013), John Wiley & sons.
13. Dana Vrajitoru, William Knight, Practical Analysis of Algorithms (2014), Springer.

❖❖❖

كتب للمؤلِّفَيْن في علوم الحاسوب والرياضيات

1 - برمجة الحاسب بلغة الفورتران، ط 4، أبو بكر السيد، دار القلم .. الكويت 1992.

2 - مقدمة في نظرية المعلومات، ط 2، أبو بكر السيد، دار القلم .. الكويت 1993.

3 - الشبكات الرقمية، أبو بكر السيد، دار القلم .. الكويت 1986.

4 - التحليل العددي، أبو بكر السيد، دار القلم .. الكويت 1988.

5 - الجبر الخطي، ط 2، أبو بكر السيد، دار القلم .. الكويت 1995.

6 - برمجة الحاسب بلغة الباسكال، ط2، أبو بكر السيد، دار القلم .. الكويت 1999.

7 - البرمجة المتقدمة بلغة الباسكال، حمزة رشوان و أبو بكر السيد، دار القلم .. الكويت 1994.

8 - الدوائر المتكاملة الرقمية (ترجمة)، أبو بكر السيد، دار القلم .. الكويت 1993.

9 - الخوارزميات والبرمجة بلغة الباسكال، أبو بكر السيد، دار القلم .. الكويت 1997.

10 - بنى المعطيات، حمزة رشوان و أبو بكر السيد، دار القلم ..الكويت 1998.

11 - الحلول العددية للمعادلات التفاضلية العادية، أبو بكر السيد، دار القلم .. الكويت 2000.

12 - الحلول العددية للمعادلات التفاضلية الجزئية، أبو بكر السيد، دار القلم .. الكويت 2001.

13 - برمجة الحاسب بلغة C++ ، أبو بكر السيد، دار القلم .. الكويت 2002.

14 - البر مجة المتقدمة بلغة C++ ، أبو بكر السيد، دار القلم .. الكويت 2003.

15 - الرياضيات المتقطعة في علم الحاسوب ، أبو بكر السيد، مكتبة الفلاح .. الكويت 2005.

16 - هياكل البيانات بلغة C++، حمزة رشوان و أبو بكر السيد، مكتبة الفلاح.. الكويت 2006.

17 - البر مجة المتقدمة بلغة C ، أبو بكر السيد، دار اقرأ .. الكويت 2006.

18 - برمجة الحاسوب بلغة C ، أبو بكر السيد، دار اقرأ .. الكويت 2007.

19 - نظم الحاسوب ، أبو بكر السيد، مكتبة الفلاح .. الكويت 2011.

20 - نظرية الحوسبة والأوتوماتية واللغات الشكلية، أبو بكر السيد، مكتبة الفلاح .. الكويت 2011.

21 - الطرق العددية والتحليل العددي، أبو بكر السيد، مكتبة الفلاح .. الكويت 2011.

22 - النظرية المتقدمة في الحوسبة والأوتوماتية واللغات الشكلية، أبو بكر السيد، ذات السلاسل .. الكويت 2016.

23 - تصميم وتحليل خوارزميات الحاسوب، أبو بكر السيد وحمزة رشوان، دار اقرأ .. الكويت 2018.

❖❖❖

تصميم وتحليل خوارزميات الحاسوب

د. أبو بكر أحمد السيد
قسم علوم الحاسوب
جامعة الكويت

د. حمزة سيد رشوان
قسم الرياضيات
جامعة الأزهر