الفصل الثامن

تقسيم المسألة و تجميع الحلول Divide and Conquer

8-1 مفهوم تقنية تقسيم المسألة و تجميع الحلول

بفرض لدينا مسألة ما P حجم المعطيات فيها n ، تقترح إستراتيجية تقسيم المسألة و تجميع الحلول بتقسيم المسألة P إلى k $(1 < k \le n)$ من المسائل الجزئية و التي تكون من نوع المسألة الأصلية نفسه $P_1, P_2, ..., P_K$. ثم يتم حل المسائل الجزئية الأصغر – أي حجم المعطيات فيها صغير بشكل كاف – و إذا كانت بعض المسائل الجزئية لا تزال كبيرة – أي حجم المعطيات فيها لا يزال كبيراً – فيتم إعادة تطبيق الإستراتيجية بأسلوب عودي لتقسيم تلك المسائل إلى مسائل جزئية أصغر وحلها بعد ذلك يتم تجميع الحلول لتكوّن حل المسألة الأصلية .

يعبر عن آلية عمل إستراتيجية التقسيم و التجميع على مسألة ما p بالدالة DandC :

```
type DandC(P)
   {
 if  (Small (P) == true)  return S(P);
 else {
   Divide p into smaller sub-problems
        P1 , P2 , ....., Pk , k ≥ 1;
   apply DandC to each of these sub-problems;
return Combine (DandC(P1), DandC(P2),….,DandC(Pk));
                }
        }
```

حيث Small(P) تابع بولياني يختبر فيما إذا كان للمسألة حجم معطيات صغير بشكل كاف . إذا كانت قيمة التابع Small(P) صحيحة ، عندئذ ينتج الحل بدون تقسيم المسألة و ذلك بتنفيذ الدالة S(P) . و غير ذلك تقسم المسألة P إلى مسائل جزئية أصغر $P_1, P_2, ..., P_k , k \ge 1;$. هذه المسائل تحل بتطبيق عودي للدالة DandC() .

الدالة Combine() تكوّن حل المسألة P باستخدام حلول المسائل الجزئية .

تحليل الدالة DandC( )

لتكن n حجم معطيات المسألة P يساوي n و ليكن $n_1, n_2, \ldots\ldots, n_k$ حجم المعطيات المسائل الجزئية ; $k \geq 1$, $P_1, P_2, \ldots\ldots, P_k$ على التوالي ، يعبر عن زمن الحساب للدالة DandC بالعلاقة العودية التالية:

$$T(n) = \begin{cases} g(n) & n \ small \\ T(n_1) + T(n_2) + \ldots + T(n_k) + f(n) & otherwise \end{cases}$$

حيث الدالة T(n) هي زمن الحساب لـــ DandC ، و الدالة *g(n)* هي زمن الحساب اللازم لإيجاد حل المسألة P ذات حجم المعطيات الصغير. الدالة *f(n)* هي زمن تقسيم المسألة P و تجميع حلول المسائل الجزئية .

8-2 البحث الثنائي Binary search

لتكن a_i قائمة من العناصر $1 \leq i \leq n$ من نوع Type مخزنة بترتيب تصاعدي و ليكن x عنصراً ما من نوع Type أيضاً . المسألة هي تحديد فيما إذا كان العنصر x موجوداً في القائمة . إذا كان العنصر x في القائمة ، عندئذ يجب تحديد قيمة الدليل j الذي يحقق $a_j = x$. أما إذا كان العنصر x غير موجود في القائمة ، عندئذ تحدد القيمة صفر للدليل j . يرمز لمسألة البحث هذه بـــ $P = (n, a_i, \ldots\ldots, a_l, x)$ حيث n عدد العناصر في القائمة $a_i, \ldots\ldots, a_l$ و x عنصر البحث .

تصاغ خوارزمية حل المسألة باستخدام إستراتيجية التقسيم و التجميع بالشكل:

– إذا أخذت الدالة Small(P) القيمة true – أي إذا كان n=1 – في هذه الحالة الدالة S(P) تأخذ القيمة i إذا كان $x = a_i$ و غير ذلك تأخذ القيمة صفر . عندئذ يكون $g(n) = \Theta(1)$.

– إذا كانت أخذت الدالة Small(P) القيمة false – أي إذا كان n >1 – في هذه الحالة يتم تقسيم المسألة إلى مسألتين جزئيتين بالشكل : نختار $q \in [i, l]$ و نقارن x مع a_q .

1. إذا كان $x = a_q$: في هذه الحالة تكون قد حلت المسألة .
2. إذا كان $x < a_q$: في هذه الحالة يتم البحث عن x في القائمة الجزئية $a_i, a_{i+1}, ..., a_{q-1}$. و بذلك تختصر المسألة P إلى المسألة : $P1 = (q - i, a_i, ..., a_{q-1}, x)$.
3. إذا كان $x > a_q$: في هذه الحالة يتم البحث عن x في القائمة الجزئية $a_{q+1}, ..., a_l$. و بذلك تختصر المسألة P إلى المسألة : $p2 = (l - q, a_{q+1}, ..., a_l, x)$.

ملاحظات:

– في المثال السابق تم تقسيم المسألة إلى مسألة جزيئه واحدة بزمن تنفيذ $\Theta(1)$

– إذا اختير الدليل q بالشكل $q = \left\lfloor \frac{n+1}{2} \right\rfloor$ – أي يقع العنصر a_q في وسط القائمة

– عندئذ تنتج خوارزمية بحث تدعى البحث الثنائي .

النسخة العودية للدالة الإجرائية التي تنفذ خوارزمية البحث الثنائي :

```
int BinSrch (Type a[], int i, int l, Type x)
/* given an array a[i: l] of elements in non- decreasing order,
1<= i <=l , determine whether x is present, and if so, return j
such that x ==a[j]; else return 0.*/
{
```

```
    if (i == l) { /* if small(P)
      if (x == a[i]) return i;
       else return 0;
  }
   else { /* reduce P into a smaller sub-problems.
       int mid = (i+l)/2
       if (x == a[mid]) return mid;
 else if (x<a[mid]) return BinSrch(a, i, mid-1, x);
  else  return BinSrch(a, mid+1, l, x);
     }
 }
```

أما النسخة التكرارية للدالة الإجرائية التي تنفذ خوارزمية البحث الثنائي فهي:

```
int BinSearch(Type a[], int n, Type x)
   /*Given an array a[1 : n] of elements in non-decreasing order,
   /*  n>=0, determine whether x is present, and if so,
   /* return j such that x == a[j]; else return 0;
      {
       int low=1, high = n;
       while (low <= high){
            int mid = (low + high)/2
            if (x < a[mid]) high=mid-1;
            else if(x > a[mid]) low = mid+1;
            else return (mid);
                 }
              return (0);
               }
```

مثال: لتكن لدينا القائمة[14 : 1]a و المكونة من العناصر التالية :

-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151

لنطبق الدالة BinSearch في البحث (بحث ناجح) عن العنصرين x: 151, 9 و البحث عن العنصر (بحث فاشل) x = -14 في القائمة المعطاة .

الجدول (8-1) يوضح تتبعاً لخطوات الدالة BinSearch في البحث عن العناصر: 151, -14, 9

الجدول(8 –1) : يوضح تتبعاً لخوارزمية البحث الثنائي على ثلاثة عناصر

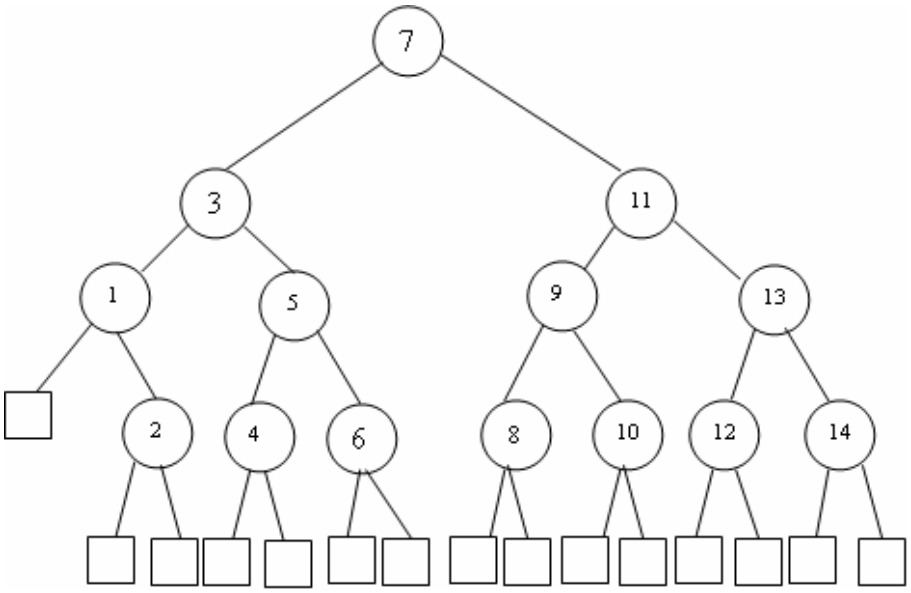
	low	high	mid		low	high	mid		low	high	mid
	1	14	7		1	14	7		1	14	7
	8	14	11		1	6	3		1	6	3
X=151	12	14	13	X=-14	1	2	1	X=9	4	6	5
	14	14	14		2	2	2				found
			found		2	1	not found				

8–2–1 تحليل خوارزمية البحث الثنائي Binary Search Analysis

أولا : زمن التنفيذ في أسوأ الأحوال

نظرية : بفرض أن n حجم المسألة ، يوجد من أجل كل n عدد صحيح موجب k يحقق $n \in [2^k, 2^{k+1}[$ ، الدالة BinSearch تحتاج إلى (k+1) مقارنة للبحث الناجح و تحتاج إلى k أو k+1 مقارنة من أجل البحث الفاشل – أي زمن تنفيذ البحث الناجح $O(\log_2 n)$ و زمن تنفيذ البحث الفاشل $\Theta(\log_2 n)$.

البرهان: لتكن شجرة القرار الثنائية (binary decision tree) الموضحة في الشكل(8–1) التي تصف عمل الدالة ()BinSearch على n عنصراً .



الشكل (8-1): شجرة قرار ثنائية للبحث الثنائي ، n = 14

نلاحظ من الشجرة أن كل بحث ناجح ينتهي في عقدة دائرية ، بينما ينتهي كل بحث فاشل في عقدة مربعة . و كون $2^k \le n < 2^{k+1}$ حيث .... k = 0, 1, 2 ، فإن العقد الدائرية تقع في المستويات ....., 0, 1, 2, k في حين تقع العقد المربعة في المستويات k و k+1 (حيث يقع جذر الشجرة في المستوى 0) . إن عدد المقارنات الضرورية للوصول إلى عقدة داخلية تقع في المستوى i يساوي i+1 مقارنة. و كذلك عدد المقارنات للوصول إلى عقدة مربعة (عقدة خارجية) تقع في المستوى i يساوي i مقارنة . و بذلك يتم المطلوب .

ثانياً : زمن التنفيذ الوسطي

نلاحظ من شجرة القرار الثنائية أن بعد (distance) عقدة عن عقدة الجذر يساوي إلى مستوى هذه العقدة ؛ أي أن distance (i) = level (i) .

- طول المسار الداخلي Internal path length (I) : هو مجموع أبعاد كل العقد الداخلية عن عقدة الجذر .

- طول المسار الخارجي External path length (E) : هو مجموع أبعاد كل العقد الخارجية عن عقدة الجذر .

يرتبط طول المسار الداخلي مع طول المسار الخارجي في شجرة مكونة من n عقدة داخلية بالصيغة : E =I+ 2n . لتكن $A_s(n)$ العدد الوسطي للمقارنات الضرورية للبحث الناجح ، و $A_u(n)$ العدد الوسطي للمقارنات الضرورية للبحث الفاشل .

و كما أشرنا سابقا أن عدد المقارنات الضرورية للوصول إلى عنصر ممثل بعقدة داخلية هي أكبر بواحد من مستوى هذه العقدة فإن :

$$A_s(n) = \frac{\sum_{i=1}^{n} [1 + level\ (i)]}{n} = \frac{n + \sum_{i=1}^{n} dis\ \tan\ ce\,(i)}{n} = \frac{I + n}{n} = 1 + \frac{I}{n}$$

كما أن المقارنات الضرورية للوصول إلى عنصر ممثل بعقدة خارجية يساوي إلى مستوى هذه العقدة . و كون كل شجرة ثنائية مكونة من n عقدة داخلية تتضمن n+1 عقدة خارجية فإن :

$$A_u(n) = \frac{\sum_{i\ is\ external\ \ node} level\ (i)}{n+1} = \frac{\sum_{i\ is\ external\ \ node} dis\ \tan\ ce\,(i)}{n+1} = \frac{E}{n+1}$$

من العلاقات السابقة نجد أن $A_s(n)$ يرتبط مع $A_u(n)$ من خلال الصيغة :

$$A_s(n) = (1 + \frac{1}{n}) A_u(n) - 1$$

تكون قيمة $A_s(n)$ (و بالتالي قيمة $A_u(n)$) أصغر ما يمكن (minimum value) إذا كان طول المسار الداخلي (الخارجي) في شجرة القرار الثنائية أصغر ما يمكن . تتحقق القيمة الأصغر لطول المسار في الشجرة الثنائية إذا كانت كل العقد الخارجية فيها تقع في مستويات متجاورة (adjacent) .

من النظرية السابقة نلاحظ أن قيمة E (و كذلك قيمة I) متناسبة طرداً مع $nlog_2n$ و بالتالي قيم $A_s(n)$ و $A_u(n)$ متناسبة مع log_2n . و بالتالي إن عدد المقارنات الوسطية هو

نفسه عدد المقارنات في أسوأ الأحوال . و بالتالي زمن تنفيذ خوارزمية البحث الثنائي (في حالتي البحث الناجح و البحث الفاشل) في أسوأ – أفضل – متوسط الأحوال و هو موضح في الجدول التالي :

Run Time	Successful Searches	Unsuccessful Search
Best	$\Theta(1)$	$\Theta(\log_2 n)$
Average	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$
Worst	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$

ملاحظة:

يوجد أشكال مختلفة و مهمة من خوارزمية البحث الثنائي التي تتضمن مقارنة واحدة في التكرار الواحد بحلقة while مثل :

```
BinSearch1 (Type a[], int n, Type x))
   {
  int low = 1, high = n+1 ; /* high is one more than possible */
  while (low <high-1) {
        int mid = (low + high)/2;
        if (x < a[mid]) high = mid ; /* only one comparison in the loop /*
        else low = mid ;  /* x >= a[mid]
             }
  if (x == a[low]) return low ; /* x is present .
  else return (0);  /* x is not present .
}
```

في حالة البحث الناجح : الدالة BinSearch1 تحتاج إلى $(\log_2 n)/2$ من المقارنات للوصول إلى عنصر ، و بالتالي زمن تنفيذ الدالة BinSearch1 موضح بالجدول التالي:

Run Time	Successful Searches	Unsuccessful Search
Best	$\Theta(1)$	$\Theta(\log_2 n)$
Average	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$
Worst	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$

8-3 إيجاد Maximum and Minimum

بفرض لدينا قائمة مكونة من n عنصراً من نوع type ، لنوجد العنصر الأكبر و العنصر الأصغر في القائمة.

الخوارزمية التقليدية لإيجاد العنصر الأكبر و العنصر الأصغر تعطى بالدالة التالية :

```
void straight-max-min(type a[], int n, type &max, type &min)
    {
   max = min = a[1];
   for (int i =2 ; i <=n; i++){
      if (a[i] > max) max = a[i];
      if (a[i] <min) min = a[i];
      }
  }
```

8-3-1 تحليل الدالة straight-max-min()

الدالة straight-max-min() تتطلب 2(n-1) مقارنة بين العناصر في أفضل الأحوال – أسوأ الأحوال – و الحالة الوسطية . يمكننا بسهولة إجراء تحسين على زمن حساب الدالة بتحقيق أن المقارنة a[i] < min تكون ضرورية فقط عندما تكون نتيجة المقارنة a[i] > max خاطئة . و بناء على ذلك نستبدل محتوى الحلقة for بـــــ :

```
if (a[i] > max) max = a[i];
else if (a[i] <min) min = a[i];
```

نلاحظ أن ***زمن التنفيذ في أفضل الأحوال*** يحدث عندما تكون العناصر مرتبة تصاعدياً ، في هذه الحالة يكون عدد المقارنات بين العناصر يساوي n-1 . ***زمن التنفيذ في أسوأ الأحـــوال*** يحدث عندما تكون تلك العناصر مرتبة تنازلياً ، في هذه الحالة يكون عدد المقارنات بـــين العناصر يساوي 2(n-1) . أما ***زمن التنفيذ الوسطي*** فيحدث عندما يكون نصف العناصـــر أكبر من max ، في هذه الحالة يكون عدد المقارنات بين العناصر:

$$\frac{n-1}{2}+\frac{2(n-1)}{2}=\frac{3(n-1)}{2} \quad .$$

لنستخدم تقنية التقسيم و التجميع في إيجاد العنصر الأكبر و العنصر الأصغر في القائمة a[i], ….,a[j] و المكونة من n عنصراً. يعبر عن هذه المسألة بالصيغة : P = (n, a[i],…,a[j]) .تكون الدالة Small(P) = true عندما n <= 2 ، في هذه الحالة يكون max = min = a[i] إذا كان n = 1 . أما إذا كان n = 2 فالمسألة تحل بإجراء مقارنة واحدة .

إذا كانت القائمة تتضمن أكثر من عنصرين ، فعندئذ المسألة P تقسم إلى مسألتين جزئيتين : $P_1=(\left\lfloor\frac{n}{2}\right\rfloor, a[1],.....,a[\left\lfloor\frac{n}{2}\right\rfloor])$ و $P_2=(n-\left\lfloor\frac{n}{2}\right\rfloor, a[\left\lfloor\frac{n}{2}\right\rfloor+1],.....,a[n])$. بعد ذلك يتم حل المسألتين P_1 و P_2 بإجراء تطبيق عودي لإستراتيجية التقسيم و التجميع لنحصل

على $\max(P_1), \min(P_1)$ و $\max(P_2), \min(P_2)$ و من ثم يتم تجميع الحلول لتكوين حل المسألة P بالشكل :

$$\min(P) = \text{MIN}\{\min(P_1), \min(P_2)\} , \ \max(P) = \text{MAX}\{\max(P_1), \max(P_2)\}$$

الدالة الإجرائية max-min هي المسؤولة عن تنفيذ تطبيق تقنية التقسيم و التجميع في إيجاد العنصر الأكبر و العنصر الأصغر بالشكل التالي :

```
void max-min(int i, int j, Type& max, Type& min)

/* a[1:n] is a global array. Parameters i and j are integers,
1<=i<=j<=n . The effect is to set max and min to the largest
 and smallest values in a[1:n], respectively .*/
{
```

```
if (i==j)max-min=a[i]; // small (P)
else if (i==j-1){// another of small (P)
        if (a[i]<a[j]){ max=a[j]; min=a[i];}
      else { max=a[i]; min=a[j];}
          }
   else {// If P is not small
        // divide P into sub problems .
       // Find where to split  the set
          int mid=(i+j)/2;
          Type max1; Type min1;
           // Solve the sub problems
        MaxMin(i, mid, max, min);
        MaxMin(mid+1, j, max1, min1);
         // Combine the solutions.
       if (max<max1) max=max1;
       if (min >min1) min=min1;
         }
     }
```

8-3-2 تحليل الدالة max-min()

ليكن $T(n)$ عدد المقارنات الضرورية في الدالة max-min() و يعطى بالصيغة التالية :

$$T(n)=\begin{cases} T(n/2)+T(n/2)+2 & n>2 \\ 1 & n=2 \\ 0 & n=1 \end{cases}$$

كما نفترض أن $n=2^k$ حيث k عدد موجب عندئذ :

$$
\begin{aligned}
T(n) &= 2T(n/2)+2 \\
&= 2(2T(n/4)+2)+2 \\
&= 4T(n/4)+4+2 \\
&\ldots\ldots\ldots\ldots\ldots \\
&= 2^{k-1}T(2)+\sum_{i=1}^{k-1}2^i \\
&= 2^{k-1}+2^k-2 = 3n/2-2
\end{aligned}
$$

ملاحظات

– نلاحظ أن المقدار $3n/2-2$ يمثل زمن تنفيذ الخوارزمية في الحالة الأفضل – الحالة الأسوأ – و الحالة المتوسطة و ذلك عندما تكون $n=2^k$.

– كما نلاحظ أيضاً أن هذه الخوارزمية قد خفضت عدد المقارنات مقارنة مع عدد المقارنات في الدالة straight-max-min بمقدار 25% .

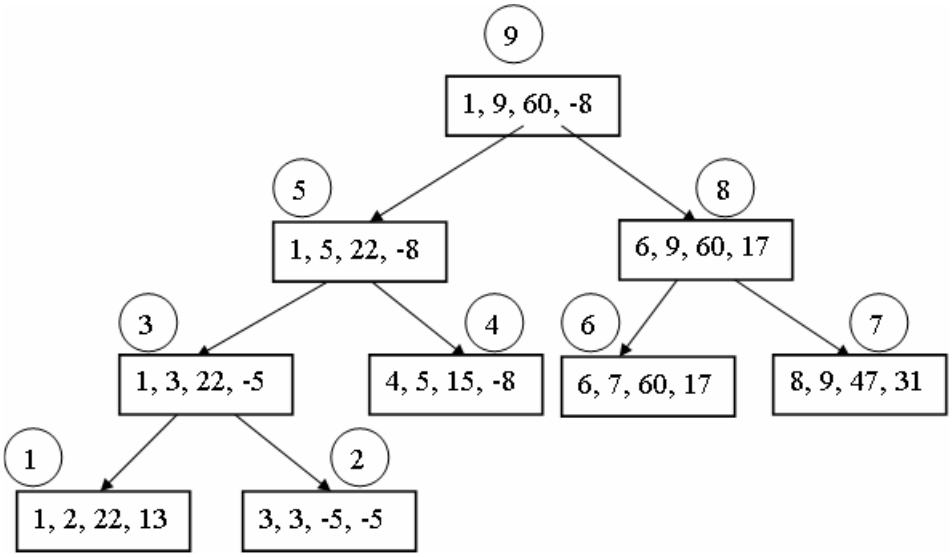
– لا توجد أي خوارزمية أخرى تقوم بإيجاد العنصر الأكبر و العنصر الأصغر في قائمة عناصر ، يكون فيها عدد المقارنات أقل من $3n/2-2$ و بالتالي تكون الخوارزمية max-min أمثلية (optimal) .

مثال :

لتكن القائمة التالية :

a:	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	22	13	-5	-8	15	60	17	31	47

يتم حفظ مسار الاستدعاءات العودية للدالة max-min ببناء شجرة عن طريق إضافة عقدة في كل استدعاء جديد للدالة . الشجرة الموضحة في الشكل (8–2) تمثل تتبعاً للدالة max-min() حيث كل عقدة تتضمن أربعة حقول هي i, j, max, min .



الشكل (8– 2) : شجرة تمثل الاستدعاءات العودية للدالة max-min()

8–4 الفرز بالدمج Merge Sort

ليكن لدينا قائمة من العناصر a[1 : n] من نوع Type و لنرتب عناصر القائمة ترتيباً تصاعدياً باستخدام تقنية التقسيم و التجميع . **أولاً:** نقوم بتقسيم القائمة المعطاة إلى قائمتين جزئيتين : $a[1 : \left\lfloor \frac{n}{2} \right\rfloor]$ و $a[\left\lfloor \frac{n}{2} \right\rfloor + 1 : n]$.

ثانيا : نرتب كل قائمة جزئية باستدعاء عودي لدالة الفرز بالدمج .

ثالثا: ندمج القائمتين الجزئيتين المترتبتين لتشكيل قائمة واحدة مرتبة مكونة من n عنصراً .

الدالة الإجرائية التي تنفذ خوارزمية الفرز بالدمج هي :

```
void merge-sort (int low, int high )
  /* a[low : high] is a global array to be sorted . Small(P) is true if there
is only one element to sort . In this case the list is already sorted . /*
  {
    if (low < high){
/* If there are more than one element . Divide P into sub-problems . Find
where to split the set .
    int = (low+high)/2;
    /* Solve the sub-problems
    merge-sort (low, mid);
    merge-sort(mid+1,high);
    /* Combine the solutions.
     Merge (low, mid, high);
      }
    }
```

دالة الدمج merge تعطى بالشكل :

```
    void merge (int low, int mid, int high )
/* a[low, high] is a global array containing two sorted subsets in
a[low, mid] and in a[mid+1 : high] . The goal is to merge these two sets
into a single set residing in a[low : high] . b[] is an auxiliary global array
*/


    {
    int h = low, i = low, j = mid+1 , k;
    while ((h <= mid) && (j <= high ))  {
          if (a[h] <= a[j]) { b[i] = a[h];  h++; }
          else {b[i] = a[j] ; j++ ;} i++;
            }
    if (h > mid) for(k = j ; k <= high ; k++) {
                        b[i] = a[k] ; i++;
                         }
    else for(k = h ; k <= mid ; k++) {
            b[i] = a[k] ; i++;
             }
    for ( k= low; k <= high ; k++)   A[k] = b[k];
       }
```

8-4-1 تحليل خوارزمية الفرز بالدمج

ليكن n عدد عناصر القائمة المراد ترتيبها . إذا كان زمن تنفيذ دالة الدمج merge متناسباً طرداً مع n ، عندئذ زمن حساب دالة الفرز بالدمج يوصف بالعلاقة العودية التالية :

$$T(n) = \begin{cases} a & n = 1,\ a\ \ is\ \ cons\ \ \tan\ t \\ 2T(\frac{n}{2}) + cn & n > 1,\ c\ \ is\ \ cons\ \ \tan\ t \end{cases}$$

عندما تكون n من الشكل 2^k ، يمكننا حل العلاقة العودية باستخدام الطريقة التكرارية :

$$\begin{aligned} T(n) &= 2(2T(\frac{n}{4}) + c\frac{n}{2}) + cn \\ &= 4T(\frac{n}{4}) + 2cn \\ &= 4(2T(\frac{n}{8}) + c\frac{n}{4}) + 2cn \\ &= 8T(\frac{n}{8}) + 3cn \\ &\ \ \cdot \\ &\ \ \cdot \\ &= 2^k T(1) + kcn \\ &= an + cn\log_2 n \end{aligned}$$

و كما نعلم أنه من أجل أي عدد صحيح موجب n يوجد عدد صحيح غير سالب وحيد k بحيث $2^{k-1} < n \le 2^k$ ، و بذلك يكون $T(n) \le T(2^k)$. و بناء على ذلك يكون $T(n) = O(n\log_2 n)$

مثال:

لتكن لدينا متجهة مكونة من عشرة عناصر :

a[1: 10] = (310, 285, 179, 652, 351, 423, 861, 254, 450, 520)

الدالة merge-sort() تقسم المتجهة []a إلى المتجهتين الجزئيتين :

a[1: 5] , a[6: 10] . كما أن العناصر في المتجهة [1:5]a تقسم أيضاً إلى متجهتين جزئيتين [5 : 4]a, [3 : 1]a العناصر في المتجهة الجزئية [3 : 1]a تقسم إلى متجهتين جزئيتين [3 : 3]a , [2 : 1]a و كذلك يتم تقسيم المتجهة [2 : 1]a إلى [2 , 2]a , [1 : 1]a و الآن نبدأ بعملية الدمج بالشكل :

(310 | 285 | 179 | 652, 351 | 423, 861, 254, 450, 520)

حيث المستقيمات العمودية تشير إلى الحدود بين المتجهات الجزئية . يتم دمج العناصر a[1] و a[2] لنحصل على:

(285, 310 | 179 | 652, 351 | 423, 861, 254, 450, 520)

ثم ندمج a[3] مع a[1 , 2] نحصل على :

(179 , 285, 310 | 652, 351 | 423, 861, 254, 450, 520)

بدمج a[4] مع a[5] نحصل على :

(179 , 285, 310 | 351, 652 | 423, 861, 254, 450, 520)

و من ثم ندمج a[1 : 3] مع a[4 : 5] نجد

(179 , 285, 310, 351, 652 | 423, 861, 254, 450, 520)

بنفس الطريقة يتم ترتيب المتجهة a[6 : 10] لنحصل على :

(179 , 285, 310, 351, 652 | 254, 423, 450, 520, 861)

يوجد متجهتان جزئيتان مرتبتان و بدمجهما ينتج المتجهة المرتبة التالية :

(179 , 254, 285, 310, 351, 423, 450, 520 , 652, 861)

8-5 الفرز السريع Quick Sort

في خوارزمية الفرز بالدمج ، تم تقسيم المتجهة a[1 : n] في المنتصف إلى متجهتين جزئيتين و من ثم تمّ ترتيب كل متجهة على حدة و بعد ذلك تمت عملية الدمج . في خوارزمية الفرز السريع يتم إعادة ترتيب العناصر في المتجهة a[1 : n] بحيث:

a[i] <= a[j] من أجل كل i بين 1 و m و من أجل كل j بين m+1 و n ، حيث m تحقق $1 \le m \le n$ و بعد ذلك ترتب العناصر في كل من المتجهتين a[1 : m] و a[m+1, n] بشكل مستقل .

إعادة ترتيب العناصر ينجز بأخذ عنصر من المتجهة a[] ، مثلاً t == a[s] ، و من ثم نعيد ترتيب العناصر الأخرى بحيث تكون كل العناصر التي تظهر قبل t في المتجهة a[1 : n] أصغر (أو تساوي) من كل العناصر التي تظهر بعد t . ندعو الصيغة التي يتم فيها إعادة الترتيب بهذا الشكل *تجزئه Partitioning* .

تعطى الدالة partition التي تنفذ عملية إعادة ترتيب العناصر في المتجهة a[] بالشكل التالي:

```
int partition (Type a[], int m, int p)
   /* within a[m], a[m+1], …, a[p-1] the elements are rearranged in
      such a manner that if initially t == a[m], then after completion
      a[q] == t for some q between m and p-1 , a[k] <= t  for
       m <= k <q , and a[k] >= t for q <k < p  .  q is returned . */
{
    type v = a[m] ; int i = m, j = p;
    do{
          do i++;
          while (a[i]<v);
          do j--;
          while (a[j] > v);
          if (i < j ) interchange (a, i, j);
   }  while (i < j);
  a[m] = a[j]; a[j] =v; return (j);
}

void interchange (Type a[], int i, int j)
  /*Exchange a[i] with a[j] */
   {
    type p = a[i];
```

```
    a[i] = a[j] ; a[j] = p;
    }
```

ملاحظة :

الدالة partition تنجز عملية التجزئة (التقسيم) للعناصر a[m : p-1] . حيث يفترض أن a[p] >= a[m] و يسمى العنصر a[m] عنصر تقسيم أو عنصر تجزئة . إذا كان m == 1 و p-1 = n عندئذ a[n+1] يجب أن يعرف و يكون أكبر (أو يساوي) من كل العناصر في المتجهة a[1 : n] .

يعبر عن خوارزمية الفرز السريع بالدالة الإجرائية التالية :

```
void quick-sort (int p, int q)
   /* Sorts the elements a[p], .., a[q] which reside in the global array
      a[1 : n] into ascending order; a[n+1] is considered to be defined
      and must be >= all the elements in a[1 : n] . */
    {
       if (p < q) {
/* if there are more than one element divide P into two sub-problems */
          int j = partition (a, p, q+1);
             /* j is the position of the partitioning element.*/
          /* solve the sub-problems .
               quick-sort(p, j-1);
               quick-sort(j+1, q);
  /* there is no need for combining solutions.*/
       }
    }
```

مثال:

لتكن المتجهة a[1: 9] = (65, 70, 75, 80, 85, 60, 55, 50, 45) . لنطبق الدالة partition (a, 1, 10) .

العناصر القابلة للمبادلة في المتجهة هي تلك الواقعة على طرفي الخط الأفقي و ذلك تمهيداً لإنتاج خط جديد . كما أن عنصر التجزئة هو a[1] = 65 .

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	i	p
65	70	75	80	85	60	55	50	45	∞	2	9
65	45	75	80	85	60	55	50	70	∞	3	8
65	45	50	80	85	60	55	75	70	∞	4	7
65	45	50	55	85	60	80	75	70	∞	5	6
65	45	50	55	60	85	80	75	70	∞	6	5
60	45	50	55	65	85	80	75	70	∞		

نلاحظ من السطر الأخير أن كل العناصر التي تظهر قبل عنصر التجزئة 65 أصغر (أو تساوي) من كل العناصر التي تلي العنصر 65 .

8-5-1 تحليل خوارزمية الفرز السريع

ليكن n عدد عناصر المتجهة المراد ترتيبها ، و لتكن T(n) عدد عمليات المقارنة في الخوارزمية . لنفترض أن : 1- عناصر المتجهة المراد ترتيبها مختلفة ، 2- عنصر التقسيم v=a[m] يملك احتمالات متساوية في كل المواقع i و $1 \le i \le p - m$ من المتجهة a[m : p-1] .

زمن التنفيذ في أسوأ الأحوال $T_w(n)$ عدد المقارنات بين العناصر في كل استدعاء للدالة partition يكون على الأكثر p-m+1 . ليكن r العدد الكلي للعناصر في كل الاستدعاءات للدالة partition على أي مستوٍ من العودية . في المستوي الأول يوجد استدعاء واحد فقط partition (a, 1, n+1) ، و يكون r = n ؛ في المستوي الثاني يوجد على الأكثر استدعائين للدالة و يكون r = n-1 ؛ و هكذا . و بناءاً على ذلك يوجد $\Theta(r)$ مقارنة بين العناصر في كل مستوى من العودية . في كل مستوٍ ، تكون قيمة r أقل بـــ 1 من المستوي الذي يسبقه. و بذلك يكون :

$$T_w(n) = \sum_{r=2}^{n} r = \Theta(n^2)$$

زمن التنفيذ في الوسطي$T_A(n)$: نفرض أن عنصر التقسيم v يملك احتمالات متساوية في كل المواقع i و $1 \le i \le p - m$ من المتجهة a[m : p-1] .

$$T_A(n) = n + 1 + \frac{1}{n}\sum_{k=1}^{n}[T_A(k-1) + T_A(n-k)] \qquad (1)$$

حيث عدد المقارنات المطلوبة بين العناصر في أول استدعاء لدالة partition يساوي n+1 .
و كذلك $T_A(0) = T_A(1) = 0$. بضرب طرفي المساواة (1) بـــ n نحصل على :

$$nT_A(n) = n(n+1) + 2[T_A(0) + T_A(1) + ... + T_A(n-1)] \qquad (2)$$

نستبدل كل n بـــــ n-1 :

$$(n-1)T_A(n-1) = n(n-1) + 2[T_A(0) + T_A(1) + ... + T_A(n-2)]$$

بطرح المساواة السابقة من المساواة (2) نحصل على :

$$nT_A(n) - (n-1)T_A(n-1) = 2n + 2T_A(n-1)$$

أو

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

بتكرار المعادلة السابقة نحصل على :

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

$$= \frac{T_A(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

.

.

$$= \frac{T_A(1)}{2} + \sum_{k=3}^{k=n+1}\frac{1}{k}$$

و بما أن : $\sum_{k=3}^{n+1}\frac{1}{k} \le \int_{2}^{n+1}\frac{1}{x}dx = \ln(n+1)-\ln(2)$ لذلك نجد أن :

$$T_{A(n)} \le 2(n+1)[\ln(n+2)-\ln(2)] = O(n\log n)$$

8-5-2 قياس أداء خوارزميتي الفرز بالدمج و الفرز السريع Performance Measurement

تم تقييم خوارزميتي الفرز بالدمج و الفرز السريع على SUN workstation 10/30 . حيث تم توليد عناصر المتجهة عشوائياً بأعداد صحيحة قيمها تتراوح بين 0 و 1000 ، وكذلك عنصر التقسيم في الدالة partition أخذ عنصر التقسيم ثلاثة مواضع في المتجهة : a[m], a[(m+p-1)/2], a[p-1] . الجدولان (8-2) و(8-3) يستعرضان أزمنة الحساب بـــ milliseconds الوسطية و في أسوأ الأحوال للخوارزميتين .

الجدول(8-2) : أزمنة التنفيذ الوسطية لخوارزميتي الفرز بالدمج و الفرز السريع

n	1000	2000	3000	4000	5000
Merge Sort	72.8	167.1	275.1	378.5	500.6
Quick Sort	36.6	85.1	138.9	205.7	269.0
n	6000	7000	8000	9000	10000
Merge Sort	607.6	723.4	811.5	949.2	1073.6
Quick Sort	339.4	411.0	487.7	556.3	645.2

الجدول(8-3) : أزمنة التنفيذ في أسوأ الحالات لخوارزميتي الفرز بالدمج و الفرز السريع

n	1000	2000	3000	4000	5000
Merge Sort	105.7	206.4	335.2	422.1	589.9
Quick Sort	41.6	97.1	158.6	244.9	397.8
n	6000	7000	8000	9000	10000
Merge Sort	691.3	794.8	889.5	1067.2	1167.6
Quick Sort	383.8	497.3	569.9	616.2	738.1

نلاحظ من الجدولين 8-1 و 8-2 أن خوارزمية الفرز السريع أسرع من خوارزمية الفرز بالدمج و ذلك من أجل كل القيم ، على الرغم من أن كلتيهما تأخذ زمن حساب وسطي $O(n\log_2 n)$.

8-6 خوارزمية إيجاد العنصر الأصغر من المرتبة k The kth-Smallest Element

تعطى متجهة العناصر a[1 : n] من النوع Type ، و لنوجد العنصر الأصغر من المرتبة k . إذا كان موضع عنصر التقسيم v للمتجهة المعطاة في a[j] عندئذ يوجد j-1 عنصر أصغر أو يساوي a[j] و n-j عنصر أكبر أو يساوي a[j] . الآن إذا كان k < j ، عندئذ العنصر الأصغر ذو المرتبة k يقع في المتجهة a[1 : j-1] ؛ أما إذا كان k = j عندئذ يكون a[j] العنصر الأصغر من المرتبة k ؛ و إذا كان k > j ، عندئذ العنصر الأصغر ذو المرتبة k يقع في المتجهة a[j+1, n] .

تنفذ خوارزمية إيجاد العنصر الأصغر من المرتبة k باستخدام الدالة الإجرائية select :

```
void select (Type a[], int n, int k)
 /* selects the kth- smallest element  in a[1:n] and places it in the kth position of a[]. The remaining elements are rearranged such that a[m] <= a[k] for 1 <= m <= k , and a[m]  >= a[k]  for k <= m <= n .*/
    {
        int low = 1, up = n+1;
        a[up] = IFTY;
       do {
            int j = partition(a, low, up);
         if (k == j) return ;
         else if (k < j ) up = j;
         else low = j+1 ;
          } while (TRUE);
   }
```

مثال:

لتكن المتجهة a[1: 9] = (65, 70, 75, 80, 85, 60, 55, 50, 45) . المطلوب هو إيجاد العنصر الأصغر من المرتبة السابعة .

بتطبيق الدالة partition (a, 1, 10) (الاستدعاء الأول للدالة partition) نجد أن العنصر 65 يوضع في الموقع a[5] . و كون 5 < 7 نطبق الدالة partition (a, 6, 10) (الاستدعاء الثاني للدالة partition) نحصل على :

a : [6]	[7]	[8]	[9]	[10]
85	80	75	70	∞
70	80	75	85	∞

نتيجة لتطبيق partition (a, 6, 10) وجدنا أن 85 (عنصر التجزئة) وضع في الموقع a[9] ، و كون 9 > 7 نطبق الدالة partition (a, 6, 9) (الاستدعاء الثاني للدالة partition) نحصل على :

a : [6]	[7]	[8]	[9]
70	80	75	85
70	75	80	85

الآن 80 هو عنصر التجزئة و موجود في الموقع a[8] ، و كون 8> 7 لذلك نستدعي الدالة partition(a, 6, 8) مرة أخرى و تعود الدالة بالقيمة 7 .

8-7 ضرب الأعداد الصحيحة Integer Multiplication

بفرض أن لدينا عددين صحيحين موجبين x, y و يتضمن كل منهما n رقم (digit) . تنجز عملية جمع العددين x, y بزمن تنفيذ قدره $\Theta(n)$. كما أن الخوارزمية التقليدية التي تنجز عملية ضرب العددين x, y تحتاج إلى زمن تنفيذ $\Theta(n^2)$.

تنجز إستراتيجية التقسيم و التجميع جداء العددين x, y اعتماداً على الصيغة الجبرية :

$$(10^{m} a + b)(10^{m} c + d) = 10^{2m} ac + 10^{m}(bc + ad) + bd$$

$$where \quad m = \left\lceil \frac{n}{2} \right\rceil$$

إن كلاً من الجداءات الجزئية الأربعة : ac, bc, ad, bd تحسب عودياً . كما تعتبر هذه المسألة صغيرة عندما يكون n = 1 أي يكون x رقماً و كذلك y .

الدالة الإجرائية التالية تنجز عملية الضرب :

```
int int-mult (int x, int y, int n)
   {
  if (n ==1) return x*y;
    else {
          m=⌈n/2⌉;
          a=⌊x/10^m⌋;   b = x % 10^m ;
          c =⌊y/10^m⌋;   d = y % 10^m ;
          int e = int-mult (a, c, m) ;
          int f = int-mult (b, d, m);
          int g = int-mult (b, c, m);
          int h = int-mult (a, d, m);
          return (10^2me+10^m(g+h)+f ;

            }
          }
```

التعقيد الزمني للدالة int-mult يعطى بالعلاقة العودية الآتية :

$$T(n) = \begin{cases} 1 & if \quad n = 1 \\ 4T(\left\lceil \frac{n}{2} \right\rceil) + \Theta(n) & if \quad n > 1 \end{cases}$$

يعطى حل هذه العلاقة باستخدام الطريقة الرئيسية $T(n) = \Theta(n^2)$. و بالتالي لا يوجد أي تخفيض لزمن التنفيذ مقارنة مع زمن تنفيذ الخوارزمية التقليدية لضرب الأعداد الصحيحة.

نستطيع تخفيض عدد الاستدعاءات العودية للدالة int-mult() من أربعة إلى ثلاثة ، و ذلك بالاستفادة من المطابقة الجبرية :

$$(a + b)(c + d) - ac - bd = ad + bc$$

و بناء على ذلك يمكننا كتابة الدالة int-mult1() بالشكل التالي :

```
int int-mult 1 (int x, int y, int n)
  {
  if (n ==1) return x*y;
else {
  m=⌈n/2⌉;
  a=⌊x/10^m⌋;  b = x % 10^m ;
  c =⌊y/10^m⌋;  d = y % 10^m ;
  int e = int-mult 1 (a, c, m) ;
  int f = int-mult 1 (b, d, m);
  int g = int-mult 1 (a+b, c+d, m);
  return (10^2m e+10^m (g-f-e)+f ;
   }
     }
```

التعقيد الزمني للدالة int-mult1 يعطى بالعلاقة العودية الآتية :

$$T(n) = \begin{cases} 1 & if\ n=1 \\ 3T(\lceil \frac{n}{2} \rceil) + \Theta(n) & if\ n>1 \end{cases}$$

يعطى حل هذه العلاقة باستخدام الطريقة الرئيسية $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585})$

8-8 جداء كثيري حدود The Polynomial Multiplication

بفرض أن لدينا كثيري الحدود A(x), B(x) من الدرجة n :

$$A(x) = a_0 + a_1 x + ... + a_n x^n \quad , \; a_n \neq 0$$
$$B(x) = b_0 + b_1 x + ... + b_n x^n \quad , \; b_n \neq 0$$

إن جداء كثيري الحدود A(x) . B(x) هو كثير حدود P(x) من الدرجة 2n ، و يعطى من خلال الدالة الإجرائية:

```
void poly-mult(Type a[],Type b[],Type p[],int n)
        {
         int i, j;
         for( i=0; i<=2*n; i++)
              p[i] = 0;
        for(i=0 ; i <= n ; i++)
             for(j=0 ; j <= n ; j++)
           p[i+j] = p[i+j]+a[i]*b[j];
           }
```

نلاحظ أن عدد الجداءات المستخدمة في الدالة هو $\Theta(n^2)$ ، و عدد المجاميع الضرورية في الدالة يكون أيضاً $\Theta(n^2)$. و بالتالي التعقيد الزمني للدالة يكون $\Theta(n^2)$.

لنطبق إستراتيجية تقسيم المسألة و تجميع الحلول لتطوير الخوارزمية بالشكل :

تعتبر المسألة صغيرة (أي Small(P) = TRUE) إذا كان n = 1 في هذه الحالة ننجز عملية الجداء باستخدام عملية جداء كثيري الحدود المباشرة ، و التي تنجز بزمن حساب $\Theta(1)$. أما إذا كان n > 2 (أي Small(P) = FALSE) ، فعندئذ نعرف كثيري الحدود $A_0(x), A_1(x)$:

$$A_0(x) = a_0 + a_1 x + \ldots\ldots + a_{\left\lfloor \frac{n}{2} \right\rfloor - 1} x^{\left\lfloor \frac{n}{2} \right\rfloor - 1}$$

$$A_1(x) = a_{\left\lfloor \frac{n}{2} \right\rfloor} + a_{\left\lfloor \frac{n}{2} \right\rfloor + 1} x + \ldots\ldots + a_n x^{n - \left\lfloor \frac{n}{2} \right\rfloor}$$

عندئذ يكون : $A(x) = A_0(x) + A_1(x) x^{\left\lfloor \frac{n}{2} \right\rfloor}$. و بنفس الطريقة نعرف $B_0(x), B_1(x)$ بحيث يكون $B(x) = B_0(x) + B_1(x) x^{\left\lfloor \frac{n}{2} \right\rfloor}$ عندئذ يكون :

$$A(x)\,B(x) = A_0(x)B_0(x) + [A_0(x)B_1(x) + A_1(x)B_0(x)]\, x^{\left\lfloor \frac{n}{2} \right\rfloor} + A_1(x)B_1(x)\, x^{2\left\lfloor \frac{n}{2} \right\rfloor}$$

نلاحظ أنه تم تقسيم المسألة ذات حجم المعطيات n إلى أربع مسائل جزئية حجم المعطيات في كل منها $\frac{n}{2}$.

يتم حل المسائل الجزئية التالية باستدعاء عودي للدالة poly-mult :

$$U(x) = A_0(x)B_0(x)$$
$$V(x) = A_0(x)B_1(x)$$
$$W(x) = A_1(x)B_0(x)$$
$$Z(x) = A_1(x)B_1(x)$$

عندئذ يكون الحل :

$$U(x) + [V(x) + W(x)] x^{\left\lfloor \frac{n}{2} \right\rfloor} + Z(x) x^{2\left\lfloor \frac{n}{2} \right\rfloor}$$

التعقيد الزمني للخوارزمية يعطى بالعلاقة العودية الآتية :

$$T(n)=\begin{cases}\theta(1) & if\ \ n=1\\ 4T(\left\lceil\frac{n}{2}\right\rceil)+cn & if\ n>1\end{cases}$$

حيث cn عدد المجاميع المستخدمة في الخوارزمية .

يعطى حل هذه العلاقة باستخدام الطريقة الرئيسية $T(n)=\Theta(n^2)$. و بالتالي لا يوجد أي تخفيض لزمن التنفيذ مقارنة مع زمن تنفيذ الخوارزمية التقليدية لجداء كثيري حدود .

يمكننا تخفيض عدد جداءات كثيرات الحدود من أربعة إلى ثلاثة بتعريف كثيرات الحدود التالية :

$$U(x)=A_0(x)B_0(x)$$
$$Y(x)=[A_0(x)+A_1(x)][B_0(x)+B_1(x)]$$
$$Z(x)=A_1(x)B_1(x)$$

عندئذ يكون :

$$Y(x)-U(x)-Z(x)=A_0(x)B_1(x)+A_1(x)B_0(x)$$

وبالتالي يكون $A(x)B(x)$ مساوياً :

$$U(x)+[Y(x)-U(x)-Z(x)]x^{\left\lfloor\frac{n}{2}\right\rfloor}+Z(x)x^{2\left\lfloor\frac{n}{2}\right\rfloor}$$

و يكون زمن التنفيذ الخوارزمية في هذه الحالة :

$$T(n)=\begin{cases}\Theta(1) & if\ \ n=1\\ 3T(\left\lceil\frac{n}{2}\right\rceil)+cn & if\ n>1\end{cases}$$

يعطى حل هذه العلاقة باستخدام الطريقة الرئيسية $T(n)=\Theta(n^{\log_2 3})=\Theta(n^{1.585})$

8-9 ضرب المصفوفات Matrix Multiplication

لتكن المصفوفتان المربعتان : $A = [a_{ij}]_{(n,n)}$ و $B = [b_{ij}]_{(n,n)}$ عندئذ جداء المصفوفتين A, B هو مصفوفة مربعة $C = [c_{ij}]_{(n,n)}$ حيث $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ من أجل كل $1 \le i, j \le n$.

الطريقة التقليدية لحساب هذا الجداء توضح من خلال الدالة الإجرائية التالية :

```
void mat-multi (Type a[][], Type b[][], Type c[][], int n)
    {
    for (int i=1; i<=n, i++)
       for (int j=1; j<=n, i++) {
          c[i][j] = 0;
          for (int k=1; k<=n, i++)
              c[h][j] +=a[i][k]*b[k][j];
            }
     }
```

إن زمن حساب الدالة mat-multi : $T(n) = \Theta(n^3)$

لنطبق إستراتيجية التقسيم و التجميع في ضرب المصفوفتين A, B .

تعتبر المسألة صغيرة (أي Small(P) = TRUE) إذا كان n <= 2 في هذه الحالة ننجز عملية الجداء باستخدام عملية جداء المصفوفات التقليدية ، و التي تنجز بزمن حساب $\Theta(1)$.

أما إذا كان n > 2 (أي Small(P) = FALSE) ، فنفترض أن $n = 2^k$ حيث k عدد صحيح غير سالب ، إذا لم تكن n من الشكل 2^k عندئذ نضيف أسطراً من الأصفار و أعمدة من أصفار حتى تصبح n من الشكل 2^k . نجزئ كلاً من المصفوفتين A, B إلى أربع مصفوفات جزئية مربعة بحجم $\frac{n}{2} \times \frac{n}{2}$ لكل مصفوفة جزئية . عندئذ يحسب الجداء AB باستخدام تعريف جداء المصفوفات بالشكل :

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \qquad (1)$$

حيث :

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{21} + A_{22}B_{22} \end{aligned} \qquad (2)$$

نلاحظ من (2) أن الجداء AB يحتاج إلى ثمانية جداءات لمصفوفات مربعة كل منها بحجم $\frac{n}{2}\times\frac{n}{2}$ و أربع عمليات جمع المصفوفات كل منها بحجم $\frac{n}{2}\times\frac{n}{2}$, و بالتالي زمن حساب جداء المصفوفتين A, B يعطى بالشكل :

$$T(n) = \begin{cases} \Theta(1) & if \quad n \le 2 \\ 8T(\frac{n}{2}) + cn^2 & if \quad n > 2 \end{cases}$$

حيث cn^2 زمن حساب جمع مصفوفتين كل منها بحجم $\frac{n}{2}\times\frac{n}{2}$

باستخدام الطريقة الرئيسة لحل العلاقة العودية السابقة نجد أن $T(n) = \Theta(n^3)$ ، و بالتالي لا يوجد أي تحسين على خوارزمية جداء المصفوفات المصممة بتقنية التقسيم و التجميع .

لتخفيض زمن التنفيذ ، اقترح Volker Strassen طريقة لحساب C_{ij} تستخدم 7 جداءات مصفوفات و 18 عملية جمع أو طرح فقط . تبدأ هذه الطريقة بحساب سبع مصفوفات كل منها بحجم $\frac{n}{2}\times\frac{n}{2}$: P, Q, R, S, T,U, V بالشكل :

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{12} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{11}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

و بعد ذلك يتم حساب C_{ij} بالشكل :

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

زمن تنفيذ طريقة Volker Strassen يعطى بالعلاقة العودية :

$$T(n) = \begin{cases} \Theta(1) & if \quad n = 2 \\ 7T(\frac{n}{2}) + cn^2 & if \quad n > 2 \end{cases}$$

باستخدام الطريقة الرئيسية في حل العلاقات العودية نجد :

$$T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81})$$

10-8 حساب القوى Exponentiation

يعطى عدد a (صحيحاً أو حقيقياً) و عدد صحيح موجب n . تحتاج الطريقة التقليدية في حساب القوة a^n إلى (n-1) عملية ضرب و الموضحة من خلال الإجرائية التالية :

```
Type straight-power ( Type a, int n)
     {
 Type x = a;
 for ( int i = 2 ; i <= n ; i++)
    x = x*a;
 return x;
    }
```

لنطبق إستراتيجية التقسيم و التجميع في حساب القوة a^n

تعتبر المسألة صغيرة (أي Small(P) = TRUE) إذا كان n = 1 و يكون الحل هو a

أما إذا كان n > 1 (أي أن Small(P) = FALSE) ، عندئذ باستخدام الصيغة $a^n = a^{\lfloor \frac{n}{2} \rfloor} a^{\lceil \frac{n}{2} \rceil}$ يتم حساب المقدار $a^{\lfloor \frac{n}{2} \rfloor}$ أولاً و من ثم نحسب $a^{\lceil \frac{n}{2} \rceil}$ باستخدام عملية ضرب إضافية على الأكثر لـــــ a إلى المقدار الأول . الدالة التالية توضح عملية حساب القوة بهذه الطريقة :

```
Type power ( Type a, int n)
      {
       if (n == 1) return a;
       else {
       Type x = power(a, n/2);
       if (n %2 == 0)
            return x*x;
       else return x*x*a
          }
```

8-10-1 تحليل الدالة power()

ليكن n حجم المعطيات في المسألة ، يعطى عدد الجداءات في الدالة power() بالعلاقة العودية :

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

إن حل العلاقة العودية باستخدام الطريقة الرئيسة : $T(n) = O(\log_2 n)$.

ملاحظات

- تكون الخوارزمية power أمثلية عندما يكون n من الشكل 2^k حيث k عدد صحيح موجب .

- توجد طرق أخرى لحساب القوة a^n أسرع بقليل من الخوارزمية power . على سبيل المثال: الخوارزمية power تحسب a^{15} بستة جداءات :

$$(a^{15} = a^7.a^7.a;\ a^7 = a^3.a^3.a;\ a^3 = a.a.a)$$

بينما توجد خمسة جداءات ضرورية لحساب a^{15} :

$$(a \to a^2 \to a^3 \to a^5 \to a^{10} \to a^{15})$$

8-11 تمارين الفصل الثامن

تمرين 1 : أثبت باستخدام الاستقراء الرياضي مايلي :

1. تعطى العلاقة بين طول المسار الداخلي I و طول المسار الخارجي E في شجرة ثنائية مكونة من n عقدة داخلية بالشكل : $E = I+2n$
2. تعطى العلاقة بين عدد العقد الخارجية $X(n)$ و عدد العقد الداخلية n في شجرة ثنائية بالشكل : $X(n)=1+n$

تمرين 2 : اقترح خوارزمية بحث " ثنائية" تقوم بتقسيم مجموعة ما حجمها n إلى مجموعتين غير متساويتين ، حجم المجموعة الجزئية الأولى يساوي $n/3$ و حجم المجموعة الثانية يساوي $2n/3$ ، ثم قارن الخوارزمية المقترحة مع خوارزمية البحث الثنائية الواردة في الكتاب .

تمرين 3 : اعتماداً على تقنية البحث الثنائي ، اقترح خوارزمية تقوم بإيجاد الجذور التربيعية (square-roots) للأعداد الطبيعية .

تمرين 4 :عدل في خوارزمية البحث الثنائية للحصول على خوارزمية بحث ثلاثية (ternary) تقوم بتقسيم مجموعة ما حجمها n إلى ثلاث مجموعات متساوية بالحجم ، حجم كل منها يساوي $n/3$.

تمرين 5 :صمم خوارزمية بتقنية التقسيم و التجميع لحل مسألة أبراج هانوي .

تمرين 6 : وجدت أن زمن تنفيذ دالة merge-sort() في أسوأ الأحوال هو $O(n\log_2 n)$. أوجد زمن تنفيذها في أفضل الأحوال .

تمرين 7 : بفرض أن لديك متجهتين من العناصر a[1: n] و b[1 : m] مرتبتين ترتيباً تصاعدياً . قدم خوارزمية بزمن تنفيذ أقل من زمن تنفيذ خوارزمية ()merge تقوم بدمج المتجهتين السابقتين في متجهة مرتبة c[1: n+m] .

تمرين 8 : لتكن $X_1, X_2, \ldots, X_l$ متتاليات مرتبة بحيث يكون $\sum_{i=1}^{l} |X_i| = n$ بيّن أنه نحتاج إلى زمن قدره $O(n \log_2 l)$ لدمج المتتاليات المعطاة .

تمرين 9 : أوجد زمن تنفيذ خوارزمية إيجاد العنصر الأصغر من المرتبة k

تمرين 10 : بين كيف تعمل خوارزمية الفرز السريع Quick Sort لترتيب متتالية العناصر :

1, 1, 1, 1, 1, 1, 1 و المتتالية 5, 5, 8, 3, 4, 3, 2

تمرين 11 : اكتب دوال إجرائية تكرارية تنفذ الخوارزميتين Quick Sort , Merge Sort

تمرين 12 : بفرض أن لدينا عددين ثنائين x, y و يتضمن كل منهما n رقماً (digit) . تنجز عملية جمع العددين x, y بزمن تنفيذ قدره $\Theta(n)$. كما أن الخوارزمية التقليدية التي تنجز عملية ضرب العددين x, y تحتاج إلى زمن تنفيذ $\Theta(n^2)$ و المطلوب :

1. باستخدام إستراتيجية التقسيم و التجميع ، قدم خوارزمية لحساب جداء العددين x, y معتمداً على الصيغة الجبرية :

$$x.y = (2^m a + b).(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

حيث $m = \left\lceil \frac{n}{2} \right\rceil$ ثم احسب التعقيد الزمني للخوارزمية .

2. اقترح طريقة لتخفيض زمن تنفيذ الخوارزمية المقدمة.

تمرين 13 : بفرض أن لدينا عددين صحيحين : الأول x يتضمن n رقماً و الثاني y يتضمن m رقماً ، و $n \geq m$. قدم خوارزمية تنجز عملية الجداء xy بزمن تنفيذ قدره $\Theta(nm^{\log_2 3-1})$

تمرين 14 : لتكن المتجهتان $V = (v_1, v_2, \ldots, v_n)$ و $W = (w_1, w_2, \ldots, w_n)$ حيث $n = 2p$. يتم حساب الجداء V*W بالصيغة التالية :

$$VW = \sum_{i=1}^{p} (v_{2i-1} + w_{2i})(v_{2i} + w_{2i-1}) - \sum_{i=1}^{p} v_{2i-1}v_{2i} - \sum w_{2i-1}w_{2i}$$

و التي تتطلب 3n/2 جداء . **المطلوب** بين كيف يتم استخدام هذه الصيغة لحساب جداء مصفوفتين تتضمن كلّ منهما n سطراً و n عموداً و التي تستلزم فقط $n^3/2+n^2$ جداء بدلا من n^3 جداء .

تمرين 15 : لتكن a[1 : n] متجهة مرتبة من الأعداد الصحيحة . قدم خوارزمية بزمن تنفيذ $O(\log_2 n)$ تقوم بالبحث عن أول دليل k يحقق a[k] = k .

تمرين 16: لتكن لدينا k متجهة مرتبة من العناصر ، كل متجهة تتضمن n عنصراً . المطلوب : قدم خوارزمية تقوم بدمج المتجهات المعطاة للحصول على متجهة مرتبة و مكونة من kn عنصراً بالطريقتين التاليتين :

- استخدام دالة الدمج merge لدمج المتجهتين الأولى و الثانية ، ومن ثم أدمج المتجهة الناتجة مع المتجهة الثالثة ، و بعد ذلك أدمج المتجهة الناتجة الجديدة مع المتجهة الرابعة ... و هكذا . احسب التعقيد الزمني للخوارزمية المقدمة التابع لـــــ k, n .
- استخدام تقنية تقسيم المسألة و تجميع الحلول للحصول على خوارزمية فعالة .

تمرين 17 : لتكن X, Y متجهتين مرتبتين من العناصر ، تتضمن الأولى m عنصراً و الثانية n عنصراً . قدم خوارزمية بزمن تنفيذ $O(\log_2 m+\log_2 n)$ لإيجاد العنصر الأصغر في المتجهة $X \bigcup Y$.

تمرين 18 : بفرض لديك مصفوفة مربعة A مكونة من n سطراً و n عموداً و المطلوب :

1. من أجل n = 2 بيّن أنه باستخدام خمسة جداءات لعناصر المصفوفة A تكون كافية لحساب الجداء AA .
2. حدّد الخطأ في الخوارزمية التالية التي تحسب الجداء AA . " باستخدام تقنية تقسيم المسألة و تجميع الحلول و استخدام تقنية Volker Strassen في حساب جداء مصفوفتين ، نقسم المسألة إلى خمس مسائل جزئية حجم المعطيات في كل منها n/2 ، كما أن زمن تنفيذ هذه الخوارزمية يعطى بالعلاقة العودية:

$$T(n) = \begin{cases} \Theta(1) & if \quad n = 2 \\ 5T(\frac{n}{2}) + cn^2 & if \quad n > 2 \end{cases}$$

باستخدام الطريقة الرئيسية في حل العلاقات العودية نجد :

$$T(n) = \Theta(n^{\log_2 5})$$