مبدأ فرق تسد (Divide and Conquer)

إن كثيرا من الخوارزميات تستخدم مبدأ "فرق تسد" أثناء عملها. تتلخص فكرة هذا المبدأ في تقسيم المشكلة المطروحة إلى عدة مشاكل أصغر في الحجم (ونعني بالكبر والصغر هنا هو عدد العناصر التي يتم معالجتها) ومماثلة للمشكلة الكبيرة من حيث المبدأ (عادة ما تكون المشكلات الصغيرة مستقلة عن بعضها)، وبعد حل كل مشكلة صغيرة على حدة تأتي مرحلة تجميع الأجزاء المحلولة بحيث تكون معا حل المشكلة. وهكذا فإن المراحل الأساسية التي تمر بها كل خوارزمية

- مرحلة التفريق أو التقسيم (The Divide Step) وفي هذا المرحلة يتم تحديد حجم المشاكل الصغيرة والتي إليها تنقسم المشكلة الكبيرة.
- مرحلة السيادة (The Conquer Step) وهنا يتم حل المشاكل الصغيرة، كل على حدة.
- مرحلة التجميع (The Combine Step) وهنا يتم تجميع حلول المشاكل الصغيرة والتي تكون معا حل المشكلة الأصلية .

وهكذا تستمر عملية التقسيم حتى تصل المشكلة إلى حجم يمكن معه حلها مباشرة دون اللجوء إلى التقسيم.

MaxMin Problem

لنبين فكرة عمل مبدأ (Divide and Conquer) تخيل المسالة التالية: أوجد أكبر عنصر و أصغر عنصر (maximum and minimum) داخل مصفوفة تحتوي على n من العناصر.

إن أول حل يفكر به المرء (Straightforward Approach) هو التالي:

$min \leftarrow A[1], max \leftarrow A[1]$

$\boldsymbol{for}\ i \leftarrow 2\ \boldsymbol{to}\ n\ \boldsymbol{do}$

$\quad \boldsymbol{if}\ A[i] < \min\ \boldsymbol{then}\ min \leftarrow A[i]$

$\quad \boldsymbol{if}\ A[i] > max\ \boldsymbol{then}\ max \leftarrow A[i]$

$\boldsymbol{return}\ (min, max)$

فإذا أخذنا عدد المقارنات كمقياس لأداء الخوارزمية فإن الخوارزمية المذكورة أعلاه تقوم بحوالي2n-2 من المقارنات. يمكن بقليل من التفكير أن نقلل عدد المقارنات إلى النصف ولكن هذا التحسين سيكون في أفضل الحالات فقط :

$min \leftarrow A[1], max \leftarrow A[1]$

$\boldsymbol{for}\ i \leftarrow 2\ \boldsymbol{to}\ n\ \boldsymbol{do}$

$\quad \boldsymbol{if}\ A[i] < \min\ \boldsymbol{then}\ min \leftarrow A[i]\ \boldsymbol{else}$

$\quad\quad \boldsymbol{if}\ A[i] > max\ \boldsymbol{then}\ max \leftarrow A[i]$

$\boldsymbol{return}\ (min, max)$

ليس من الصعب أن نرى أن عدد المقارنات هو (n-1)، ذلك أن تنفيذ الأولى يستبعد الثانية مباشرة فيما عدا ذلك، فإن الخوارزمية ستنفذ (2n-2) من الخطوات وسنرى كيف سيؤثر استخدام مبدأ (Divide and Conquer) جذريا في حل المسألة ولكن للسهولة سنفترض أن العدد n هو احد قوى العدد 2 أي $n=2^k$ كما يلي:

1. نقوم بتقسيم المصفوفة المعطاة والتي حجمها n إلى مصفوفتين حجم كل واحدة منهما هو n/2 وهذا ما يسمى بخطوة التقسيم (The Divide Step)
2. نقوم بإيجاد العنصر الأصغر لكل مصفوفة (min_1,min_2) وكذلك العنصر الأكبر (max_1,max_2)
3. مرحلة التجميع (The Combine Step) وفيها نقارن min_1,min_2) ونختار أصغرهما:

```
Algorithm DivideAndConquerMinMax
Input An array A[1..n] of n integers, n is a power of 2.
Output (min, max): the minimum and maximum integers in A.
    1. minmax (1, n)
Procedure minmax(low; high)
    1. if high – low = 1 then
    2.     if A[low] < A[high] then return(A[low], A[high])
    3.     else return (A[high], A[low])
    4.     end if
    5. else
    6.     mid ← ⌊(low + high)/ 2⌋
    7.     (min1, max1) ← minmax (low, mid)
    8.     (min2, max2) ← minmax (mid + 1, high)
    9.     min ← minimum{min1, min2}
    10.    max ← maximum{max1, max2}
    11.    return(x, y)
    12. end if
```

واضح أن هناك عمليتي مقارنة في الخطوتين (1) و (2) وكذلك في الخطوتين (9) و (10)

إن الخطو ة (6) تمثل عملية التقسيم بينما الخطوتان (7) و (8) يمثلان مرحلة حل المشكلات الجزئية (The Conquer Step) و تشكل الخطوتان (9) و (10) مرحلة التجميع (The Combine Step) سنقوم بحساب تعقيد الخوارزمية وسنضع في حسباننا أن الخوارزمية تحتوي على استدعاء ذاتي (recursive) أو تعاود:

$$C(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2C(n/2) + 2 & \text{if } n > 2. \end{cases}$$

باستخدام المبرهنة الأساسية نلاحظ أن $f(n) = 2$ وكذلك لاحظ أن $n^{log_2 2} = n$ اي أن $a = 2; b = 2;$ سنستخدم التعويض لنرى المراحل التي تمر بها الخوارزمية حتى تصل إلى قيمة المتغير n إلى 2 $C(n) \epsilon\ O(n)$

كما يلي :

$$\begin{aligned} C(n) &= 2C(n/2) + 2 \\ &= 2(2C(n/4) + 2) + 2 \\ &= 4C(n/4) + 4 + 2 \\ &= 4(2C(n/8) + 2) + 4 + 2 \\ &= 8C(n/8) + 8 + 4 + 2 \\ &\vdots \end{aligned}$$

وإذا انتبهنا إلى أن $n = 2^k \Rightarrow k = \log n$ فلابد أننا سنلاحظ أن العبارة س ستؤول إلى الصورة

$$\begin{aligned} &= 2^{k-1}C(n/2^{k-1}) + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2 \\ &= 2^{k-1}C(2) + \sum_{j=1}^{k-1} 2^j \end{aligned}$$

فإذا أخذنا بعين الاعتبار الحقائق الرياضية التالية :

$$\sum_{i=0}^{n} 2^i = 2^n - 1 \Rightarrow \sum_{i=1}^{n} 2^i = 2^n - 2 \qquad (1)$$

$$n = a^{log_a(n)} \qquad (2)$$

وفي حالتنا هذه فإن القيمة a من العلاقة (2) تساوي العدد 2 فإن العلاقة تؤول إلى الشكل التالي:

$$C(n) = 2^{k-1}C(2) + \sum_{j=1}^{k-1} 2^j = \left(\frac{2^k}{2}\right)C(2) + 2^k - 2 = \frac{n}{2}(1) + n - 2 = \frac{3n}{2} - 2$$

والنتيجة الأخيرة هي أكثر فعالية من $2n - 2$ وهي تستحق أن تحمل لقب مبرهنة

Binary Search

إن الهدف من هذه الخوارزمية هو تحديد مكان عنصر ما x داخل مصفوفة مرتبة بشكل تصاعدي (غير تنازلي non decreasing order) تحتوي على n من العناصر. طبيعي أن أول ما يخطر على بالنا هو الخوارزمية التالية

```
Procedure Search (A,x)

Input : An array A[1..n] of elements sorted in none decreasing order and an element x.

Output: i if A[i]=x , 1 ≤ I ≤ n and 0 otherwise.

for i=1 to n do

        if(A[i]=x)then return i

return 0
```

أن هذه الخوارزمية تقوم ب مقارنة واحدة في أحسن الحالات (The Best Case) بين تقوم ب n مقارنة في أسوأ حالة (worst case) .

سنستخدم مبدأ "فرق تسد" لحل نفس المعضلة وسندرس تعقيد الخوارزمية لنرى الفرق بين الطريقتين:

Algorithm BINARYSEARCHREC

Input: An array $A[1..n]$ of n elements sorted in non decreasing order and an element x.

Output: j if $x = A[j]$, $1 \le j \le n$, and 0 otherwise.

1. *binarysearch* (1, n)

Procedure *binarysearch* (*low, high*)
1. if $low > high$ then return 0
2. else
3. $mid \leftarrow \lfloor (low + high)/2 \rfloor$
4. if $x = A[mid]$ then return mid
5. else if $x < A[mid]$ then return *binarysearch* (*low, mid* -1)
6. else return *binarysearch* (mid + 1, *high*)
7. end if

سنوضح عدة أمور قبل الشروع في التحليل:

- سنتخذ عدد المقارنات كمقياس لتعقيد كمقياس للخوارزمية
- سنعتبر العبارة if التي تحوي بداخلها مقارنتين أو 3 مقارنة كمقارنة واحدة.

لنعتبر $C(n)$ هو عدد المقارنات المطلوب نحن أمام أكثر من خيار حسب قيمة n فإذا كان لدينا

- n=0 فالمصفوفة خالية ولا تحتوي على أي عنصر
- n=1 فالمصفوفة تحتوي على عنصر وحيد ونقوم بمقارنة واحدة
- إما إذا كانت قيمة n>1 فنحن أمام حالتين أيضا
  - إما أن A[mid]=x وبالتالي نكون قد وجدنا العنصر المطلوب (لاحظ في الخطوة 3 من الخوارزمية السابقة أن حسب mid تمثل مرحلة التقسيم).
  - وإما A[mid]≠ x فيكون عدد المقارنات المطلوب هو 1+ عدد المقارنات الناتج من فحص **أحد** شقي المصفوفة قد يكون هذا الشق الأيمن إذا كان x >A[mid] أو الأيسر في الحالة الأخرى.

يمكن تمثيل ما سبق رياضيا بالشكل الأتي

$$C(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 + C(\lfloor n/2 \rfloor) & \text{if } n \geq 2. \end{cases}$$

بإستخدام المبرهنة الأساسية (Master theorem) **الحالة الثانية** نستطيع أن نقول أن تعقيد الخوارزمية هو $O(\log n)$.

انظر إن f(n) = 1 دالة ثابتة ولاحظ أن

$$a = 1; b = 2 \Rightarrow n^{log_2 1} = n^0 = 1$$

نستطيع أن نقول أن التعقيد المطلوب هو $O(n^0 logn) = O(logn)$

كما أن استخدام التعويض سهل لحل هذه المعادلة التعاودية :

$$C(n) = C\left(\frac{n}{2}\right) + 1 \; ; C(1) = 1$$

$$C(n) = C\left(\frac{n}{2}\right) + 1 = C\left(\frac{n}{4}\right) + 1 + 1 = C\left(\frac{n}{8}\right) + 1 + 1 + 1 = C\left(\frac{n}{2^k}\right) + k$$

$$put\ n = 2^k \Rightarrow \log n = k \qquad (3)$$

سنستفيد من (3) لحل المسالة كما يلي

$$C(n) = C\left(\frac{n}{2^k}\right) + k = C(1) + k = 1 + \log n$$

وهكذا فإن $C(n) = O(\log n)$ وهذا هو ارتفاع شجرة الاستدعاء الذاتي.

Merge Sort

تخيل أننا نحتاج إلى ترتيب مصفوفة تحتوي n من العناصر. طبيعي إن أول خوارزمية تخطر على بالنا هي الخوارزمية التالية التي تحمل اسم خوارزمية الفرز الاختياري (Selection Sort):

```
For i ← 1 to n - 1 do
 min ←i;
 for j ← i + 1 to n do
          if a[min] > a[j] then
             min ← j
  t ←a[i]
 a[i] ← a[min]
 a[min] ← t
```

لحساب تعقيد هذه الخوارزمية نراعي كون الحلقة الداخلية تعمل مقارنة واحدة

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

$$= n(n-1)/2$$

إذن تعقيد هذا الخوارزمية هو $O(n^2)$

إن استخدام مبدأ "فرق تسد" يقلل كثيرا من اللازم لفرز مصفوفة كما سنرى عند تنفيذه في الفرز في بالدمج (merge Sort) كما سنوضح الان:

Algorithm MERGESORT

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in nondecreasing order.

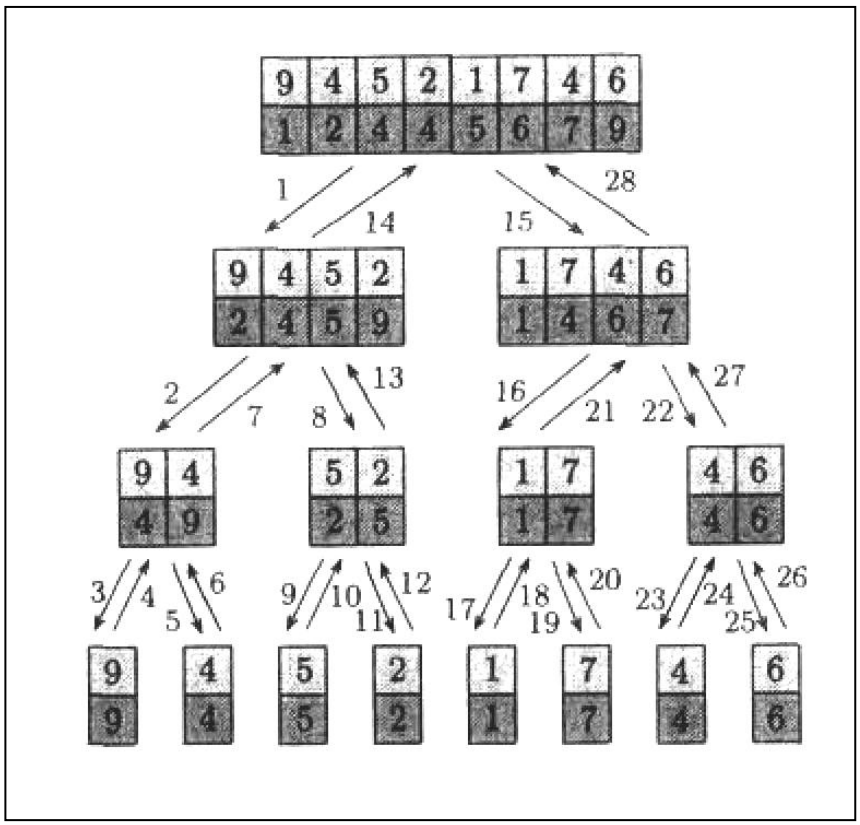
1. *mergesort*$(A, 1, n)$

Procedure *mergesort*$(A, low, high)$

1. **if** $low < high$ **then**
2. $mid \leftarrow \lfloor (low + high)/2 \rfloor$
3. *mergesort*(A, low, mid)
4. *mergesort*$(A, mid + 1, high)$
5. MERGE$(A, low, mid, high)$
6. **end if**

هناك عدة نقاط نستطيع أن نستشفها من الخوارزمية المذكورة أعلاه :

1. الخطوة رقم 2 تمثل مرحل التقسيم إلى مشاكل اصغر وحجم كل مشكلة
2. الخطوتان 3 و 4 تمثلان مرحل حل المشاكل الأصغر
3. الخطوة 5 تمثل مرحلة تجميع الحلول الجزئية حيث يتم فيها دمج مصفوفتين مرتبتين، طول كل واحد منهما هو $n/2$ داخل مصفوفة طولها. n نحتاج إلى من الخطوات لإنجاز مهمة الدمج كما سنرى لاحقا.



في الرسم المقابل لاحظ أن الأسهم النازلة هي أسهم تدل على عملية تقسيم، بينما تدل الأعمدة الصاعدة على عملية دمج. الصفوف ذات اللون الفاتح هي صفوف داخلة إلى عملية التقسيم وبالتالي هي صفوف غير مرتبة أما الصفوف ذات اللون الداكن هي صفوف ناتجة من عملية الدمج وهي صفوف مرتبة.

```
procedure merge
Input A is array[1..n] of integer , L lower bound to start merging,
H higher bound to end the merging,
mid middle element of array
  i ←L,    j ← mid + 1;   k ← 1;
  while (i <= mid) and (j <=R) do
            if  (A [i]> A[j]) then
                    tmp[k] ←A[j];  j←j+1
                else
                    tmp[k] ←A[i];  i←i+1
            k←k+1
  if i <= mid then
   for c ←i to mid do
    tmp[k] ←A[c]; k←k+1;
  else
   for c ← j to R do
    tmp[k] := A[c];  k←k+1
end
```

هنا سنبين كيف تعقيد الإجرائية Merge المسؤولة عن عملية الدمج بين المصفوفتين الصغيرتين المرتبتين لتكوين مصفوفة مرتبة، حجمها يساوي مجموع حجمي المصفوفتين الصغيرتين وسنأخذ كمثال تقسيم المصفوفة إلى جزئين اي عندما يكون طول كل مصفوفة صغيرة هو $n/2$.

لنفترض عند خروجنا من الحلقة while أن $i=x_1$ وقيمة **$j=x_2=n/2+f$**). إذن عدد عمليات المنح ل tmp[k] هو

$$k = x_1 + f$$

فإذا كان الشرط $i \le mid$ متحققا فهذا يعني أن سبب الخروج من الحلقة while هو وصول j إلى القيمة R أي إلى القيمة n و هذا يعني أن $f=n/2$ ويكون عدد عمليات المنح التي أجريت هو

$$k = x_1 + n/2$$

لاحظ حينئذ أن الحلقة for تبدأ التنفيذ من القيمة $c=i=x_1$ وبما أن $mid=n/2$ فإن

عدد الخطوات التي سيتم إجراؤها $n/2-x_1$ وسيصل عدد عمليات المنح التي أجريت إلى $k = x_1 + \frac{n}{2} + \frac{n}{2} - x_1 = n$.

لنفترض أن ذلك الشرط لم يتحقق معنى هذا أن سبب الخروج من الحلقة while هو وصول قيمة x_1 إلى القيمة $n/2$

ويكون عدد عمليات المنح التي أجريت هو $k = n/2 + f$. لاحظ حينئذ أن الحلقة for تبدأ من القيمة $c=j=n/2+f$. أن عدد الخطوات التي ستجري في هذه الحلقة هو $R - j = n - \frac{n}{2} - f = \frac{n}{2} - f$ وهذا يعني عدد عمليات المنح هو $k = \frac{n}{2} + f + \frac{n}{2} - f = n$.

إذا تعقيد خوارزمية هو n .

سنعود الآن إلى حساب تعقيد الخوارزمية الرئيسية وهي الMeregeSort

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

إن المبرهنة الأساسية (Master theorem) تعطينا جوابا فوريا هو $O(n \log n)$ ذلك أن هذه العلاقة تحقق الحالة الثانية (Case 2) من تلك المبرهنة .

Quick Sort

خوارزمية الفرز السريع

تعتمد هذه الخوارزمية أيضا على مبدأ فرق تسد حيث تقوم هذه الخوارزمية بتقسيم المصفوفة المعطاة إلى مصفوفتين صغيرتين ثم ترتب كل واحدة منهما على حدة وبعد ذلك تقوم بدمج النتائج دون اللجوء إلى ذاكرة خارجية بعكس الفرز عن طريق الدمج (Merge Sort)

Algorithm QUICKSORT
Input: An array $A[1..n]$ of n elements.
Output: The elements in A sorted in nondecreasing order.

1. *quicksort*$(A, 1, n)$

Procedure *quicksort*$(A, low, high)$
1. **if** $low < high$ **then**
2. SPLIT$(A[low..high], w)$ {w is the new position of $A[low]$}
3. *quicksort*$(A, low, w-1)$
4. *quicksort*$(A, w+1, high)$
5. **end if**

واضح من هذه الشفرة أن العمل الأساسي للخوارزمية يقع على عاتق الإجراء SPLIT حيث يقوم هذا الإجراء بتحديد عنصر إسنادي Pivot Element ويقوم بإزاحة كل العناصر التي تكبره إلى يمينه وإزاحة كل العناصر التي تصغره إلى يساره (بفرض أننا نريد فرزا تصاعديا) والشفرة التالية توضح عمل هذا الإجراء.

Algorithm SPLIT
Input: An array of elements $A[low..high]$.
Output: (1) A with its elements rearranged, if necessary, as described above
(2) w, the new position of the splitting element $A[low]$.

1. $i \leftarrow low$
2. $x \leftarrow A[low]$
3. **for** $j \leftarrow low + 1$ **to** $high$
4. **if** $A[j] \leq x$ **then**
5. $i \leftarrow i + 1$
6. **if** $i \neq j$ **then** interchange $A[i]$ and $A[j]$
7. **end if**
8. **end for**
9. interchange $A[low]$ and $A[i]$
10. $w \leftarrow i$
11. **return** A and w

واضح أن هذه الخوارزمية SPLIT تقوم ب n-1 من المقارنات من اجل n من العناصر، فإذا نظرنا إلى كون QuickSort تحتوي على حالتي استدعاء ذاتي فإن تعقيدها سيعطى بالشكل التالي :

$T(n)=2T(n/2)+n$

وحسب المبرهنة الأساسية فإن تعقيد هذه الخوارزمية هو $O(n \log n)$ طبعا هذا في أحسن الحالات؛ أما أسوأ حالة فتحدث عندما يكون العنصر الإسنادي هو اصغر عنصر في المصفوفة ($w = low$) و نتيجة لهذا فإن أحد حالتي الاستدعاء الذاتي لن تعمل :

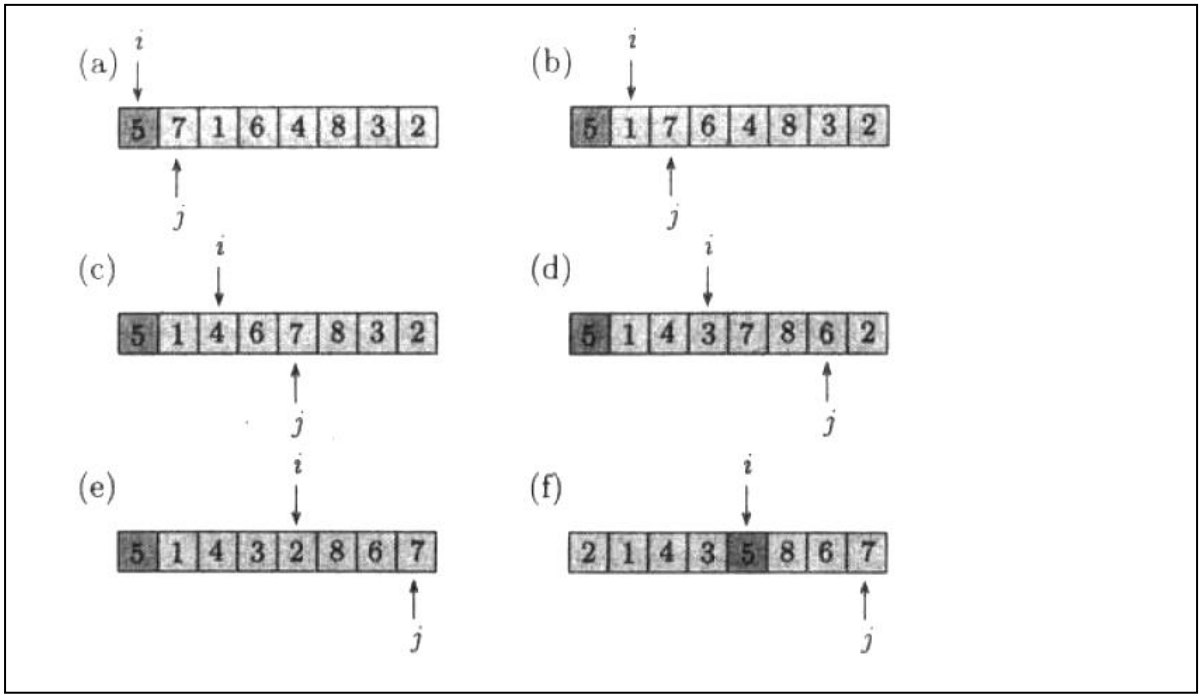
QuickSort(A,1,n), QuickSort(A,2,n), QuickSort(A,3,n),…., QuickSort(A,n,n)

وهكذا فإن

SPLIT(A[1..n],w), SPLIT(A[2..n],w), SPLIT(A[3..n],w),…., SPLIT(A[n..n],w)

$(n-1)+(n-2)+(n-3)+\ldots+1+0=\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$

ويبين الشكل التالي عمل الأجراء SPLIT



Integer Exponentiation algorithm

خوارزمية الرفع إلى أس طبيعي

واضح أن أول شيء يفكر فيه المرء عندما يكتب خوارزمية لرفع عدد حقيق إلى أس طبيعي هو التالي

```
Algorithm EXPNONRECURSIVE
Input: A real number x and a nonnegative integer n.
Output: x^n.
    1. y ← 1
    2. for i ← 1 to n
    3.     y ← y * x
    4. end for
    5. return y
```

في الحقيقة إن هذه الخوارزمية فعالة فهي تقوم ب n من عمليات الضرب وهو زمن خطي ولكن يمكن جعل هذه الخوارزمية أسرع بإستخدام تقنية Divide and Conquer كما سنرى :

لنفرض أن m=[n/2] فإذا كان ولنفرض أننا نعلم كيف نحسب x^m

1. سنعتبر أن $y = x^m \cdot x^m$
2. نحن الآن أمام حالتين
   a. إذا كان n عددا زوجيا فإن y هو العدد المطلوب (كنتيجة لرفع x إلى الأس n)
   b. إذا كان n عددا فرديا فإن y=y . x .

يمكن تمثيل هذه الفكرة في الشفرة التالية :

```
Algorithm EXPREC
Input: A real number x and a nonnegative integer n.
Output: x^n.

    1. power(x, n)

Procedure power(x, m)      {Compute x^m}
    1. if m = 0 then y ← 1
    2. else
    3.     y ← power(x, ⌊m/2⌋)
    4.     y ← y^2
    5.     if m is odd then y ← xy
    6. end if
    7. return y
```

لاحظ أن الشفرة الأخيرة تقوم بعمليتي مقارنة علاوة على عملية استدعاء ذاتي واحدة وأن بارامت ر هذا الاستدعاء الذاتي يساوي نصف البارامتر المدخل. بمراعاة كل الأمور السابقة يمكن أن زمن هذه الخوارزمية يعطى بالعلاقة :

$$T(n) = T(n/2) + 2$$

وباستخدام المبرهنة الأساسية (Mater theorem) الحالة الثانية نجد أن زمن الخوارزمية هو $O(\log n)$