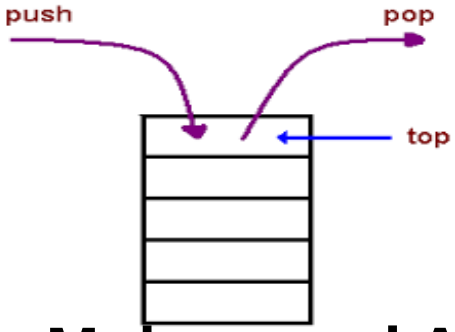
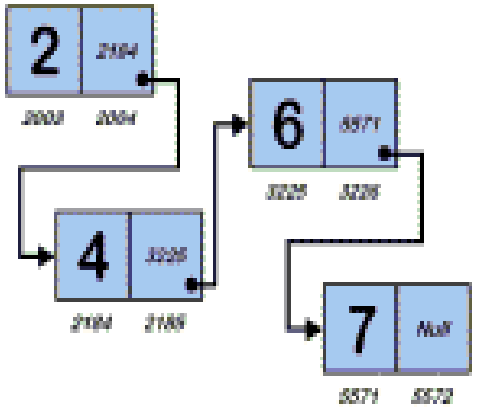
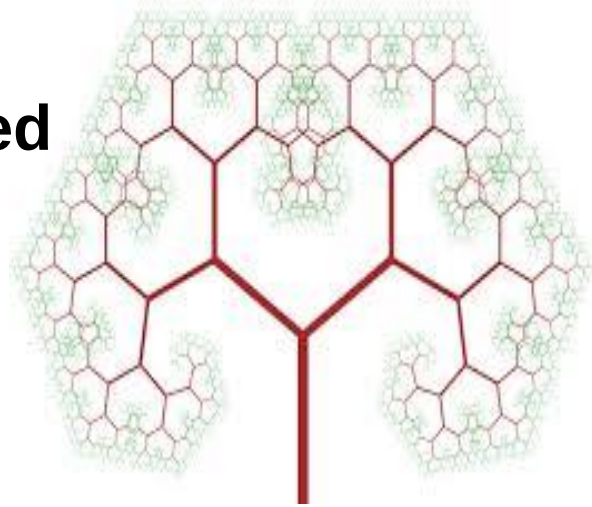
جامعة حماة
الكلية التطبيقية
قسم تقنيات الحاسوب
الفصل الثاني



- المقرر: الخوارزميات وبنى المعطيات
- المحاضرة :الرابعة

Dr. Mohammed AL-Mohammed





Chapter 4

خوارزميات الفرز والترتيب

Sorting Algorithms

- 2- جزء

الفرز بالحشر (الإدراج) *Insertion sort*

مبدأ عمل الخوارزمية:

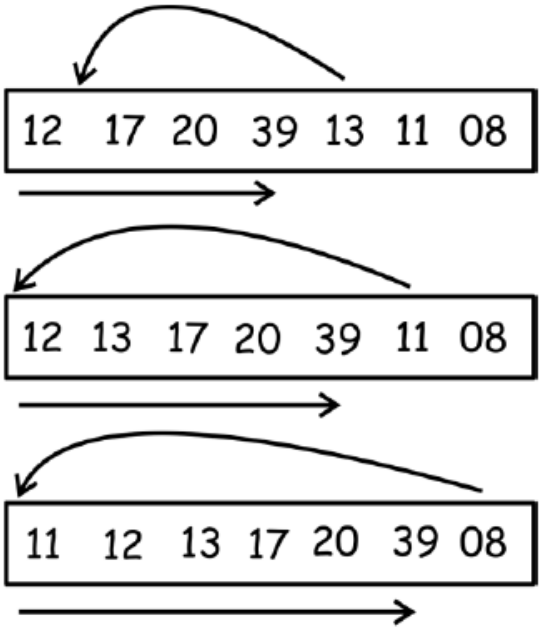
تعتمد هذه الخوارزمية على تحريك العنصر من مكانه ليتم وضعه (اقحامه – إدراجه- حشره) في المكان المناسب ضمن مجموعة العناصر المرتبة حيث أن هذه الخوارزمية تقوم بمقارنة العنصر المراد ترتيبه مع جميع العناصر التي قبله فإذا كان هذا العدد أصغر من الأعداد التي قبله نقوم بعدة إزاحات حتى يصبح في موضع تكون فيه كل الأرقام التي قبله أصغر منه وكل الأعداد التي بعده أكبر منه (على مبدأ ترتيب أوراق الشدة).

تبدأ الخوارزمية مع العنصر الثاني في المصفوفة لتضعه في مكانه , ثم تأخذ العنصر الثالث لتنقله لمكانه المناسب وهكذا حتى آخر عنصر في الخوارزمية.

حيث يقارن بين أول عنصرين ثم يقارن بين ثاني عنصرين و يعود ليتأكد من ترتيب العنصرين الاوليين و هكذا ...

8,4,3,2,9,1	يقارن بين اول عنصرين
4,**8,3**,2,9,1	يقارن بين ثاني عنصرين
4,3,8,2,9,1	يعود ليتأكد من ترتيب العنصرين الاوليين
3,4,**8,2**,9,1	ثم يكمل و يقارن بين العناصر ...

الفرز بالحشر (الإدراج) *Insertion sort*



يعتمد على مقارنة العنصر مع جميع العناصر التي قبله إذا كان أصغر من أحدها يغير مكانه حتى تكون كل الأعداد قبله أصغر منه.

بمعنى آخر:

تهدف هذه الخوارزميّة إلى القيام بترتيب مصفوفة عبر اعتبار أن المصفوفة مكوّنة من قسمين: الأول يحوي عناصر مرتّبة (تكون في البدء مكوّنة من عنصر وحيد)، والثاني عبارة عن مجموعة من العناصر، يتم سحب العناصر منها واحداً تلو الآخر، والبدء بمقارنة العنصر الجديد مع أكبر عنصر من القسم المرتّب، وعندما يكون العنصر الجديد أكبر من قرينه، نكون قد اوجدنا المكان المناسب له، وإلا، نقوم بالاستمرار بالمقارنة مع العنصر الذي يليه، حتى نهاية المصفوفة.

الكود البرمجي:

```
1. void insertNextItem(int a[],int i)
2. {
4. int newItem(a[i]),insertPos(i);
4. for(; insertPos && newItem<a[insertPos-1];insertPos--)
5. a[insertPos]=a[insertPos-1];
6. a[insertPos]=newItem;
7. }
8. void insertionSort(int a[],int n)
9. {
10. int i;
11. for(i=1;i<n;i++)
12. insertNextItem(a,i);
14. }
```

التابع الأساسي هو $insertionSort$ له بارامتراته: مصفوفة a عدد عناصرها n يحتوي على حلقة for نستدعي بداخلها التابع $insertNextItem$

في التابع $insertNextItem$ عرفنا متغيرين $newItem$ وأسندنا له قيمة $a[i]$ أي قيمة آخر عنصر في المصفوفة التي عدد عناصرها هو i وعرفنا المتغير $insertPos = i$.

تعني حلقة for أننا نقوم بمقارنة العنصر الحالي مع العنصر الذي قبله أي $a[insertPos]$ مع $insertPos - 1$ فإذا كان العنصر الحالي أصغر من العنصر الذي قبله يتم التبديل وتستمر المقارنة مع العناصر إلى أن تصل إلى العنصر الأول ونلاحظ أنه لا يوجد قيم ابتدائية إذاً الحلقة تعمل ما دام الشرط محقق وتعمل الحلقة طالما أن $index$ العنصر وقيمته أصغر من قيمة العنصر الذي يسبقه.

في حلقة for هذه يتم ترتيب حد واحد فقط من عناصر المصفوفة لذلك قمنا باستدعائه ضمن حلقة for حتى يترتب جميع عناصر المصفوفة.

التعقيد Big-O لهذه الخوارزمية

يلعب توزع القيم في هذه الخوارزمية دوراً كبيراً فلدينا حالات للتعقيد:

أسوأ حالة: عندما تكون المصفوفة مرتبة تنازلياً ونريد أن نرتبها تصاعدياً.

ثاني حد يقوم بمقارنة واحدة

وثالث حد يقوم بمقارنتين

⋮

الحد الأخير يقوم بـ $n-1$ مقارنة

وبالتالي فالتعقيد لها هو $O(n^2)$.

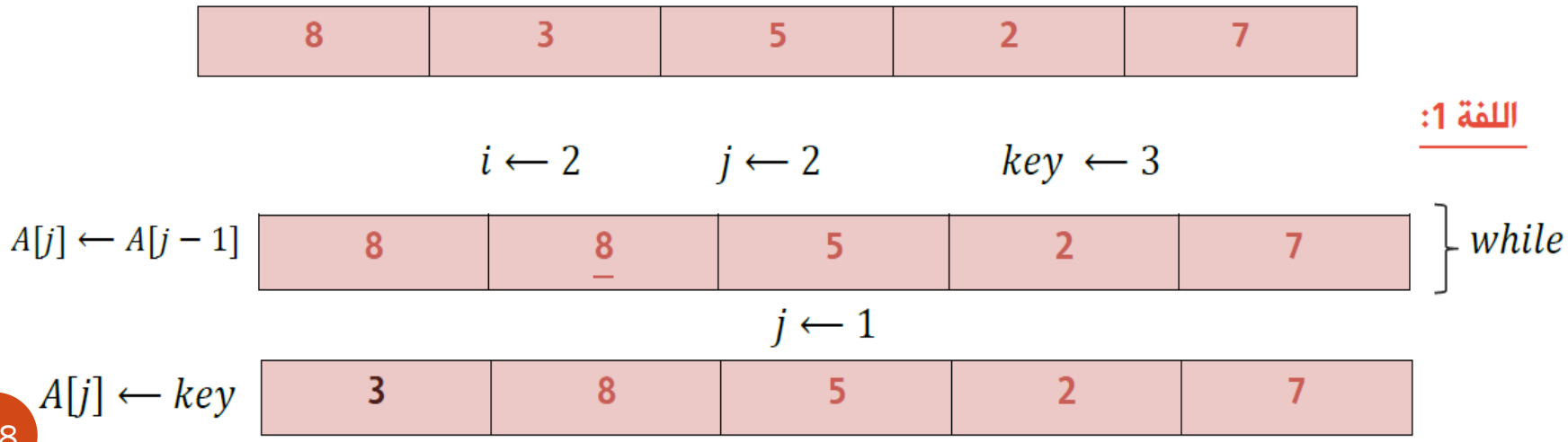
أفضل حالة: إذا كانت مرتبة بالاتجاه المطلوب عندها فإن حلقة for ستدور مرة وحيدة وبالتالي فالتعقيد لها هو $O(n)$.

حالة متوسطة: تعقيدها $O(n^2)$.

مقارنة بين الفرز الانتقائي والفرز بالحشر:

Insertion	Selection
في الحالة المتوسطة يكون $O(n^2)$	الـ $big - oh$ دائماً $O(n^2)$
في أفضل حالة يكون $O(n)$	لا يوجد أفضل حالة
في أسوأ حالة يكون $O(n^2)$	لا يوجد أسوأ حالة
عمليات التبديل $(swap)$ في أسوأ حالة والحالة المتوسطة هي $O(n^2)$	عمليات التبديل $(swap) \leftarrow O(n)$

مثال: لتكن لدينا المصفوفة



اللفة 2:

$i \leftarrow 3$ $j \leftarrow 3$ $key \leftarrow 5$

$A[j] \leftarrow A[j-1]$

3	8	8	2	7

while

$j \leftarrow 2$

$A[j] \leftarrow key$

3	5	8	2	7

اللفة 3:

$i \leftarrow 4$ $j \leftarrow 4$ $key \leftarrow 2$

$A[j] \leftarrow A[j-1]$

3	5	8	8	7

$j \leftarrow 3$

$A[j] \leftarrow A[j-1]$

3	5	5	8	7

$j \leftarrow 2$

$A[j] \leftarrow A[j-1]$

3	3	5	8	7

$j \rightarrow 1$

while

Shell Sort

تعتمد على الـ *Dynamic Programming* أو تقسيم العمل أو المشكلة إلى مشكلات صغيرة وهي تعديل لخوارزمية *insertion* وتعتمد على تقسيم المصفوفة إلى مصفوفات جزئية وفق قوانين معينة ومن ثم ترتيب الجزئيات وفق خوارزمية *insertion*.

ترتيب شيل *Shell*: إن ترتيب الإدراج *Insertion sort* يكون أسرع في الحالات التالية:

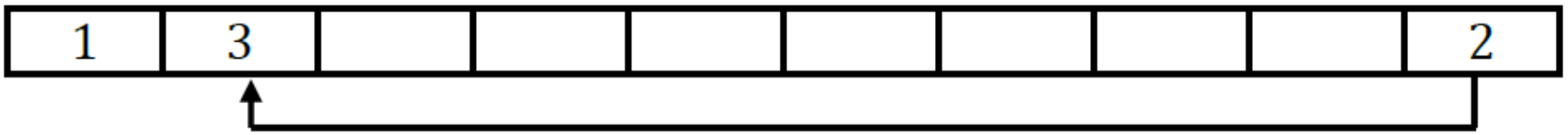
1- عندما تكون القائمة مرتبة تقريباً.

2- عندما يكون حجم القائمة صغيراً (n صغيرة).

إن ترتيب *Shell* يجب أن يستفيد من نقاط قوة ترتيب الإدراج:

1- سيتعامل مع عدد قليل من العناصر في كل مرة.

2- يتم نقل العناصر مسافة طويلة في القائمة مع كل عملية مبادلة مما يجعل القائمة مرتبة **تقريباً** (بخلاف باقي خوارزميات الترتيب).



⇐ أصبح العنصر في مكانه تقريباً بعد عملية $shift$ نختصر الكثير من التعقيد الزمني وسواء تم وضع العنصر المنقول في مكانه الأصلي أم لا، فإننا قربناه إلى أقرب موضع من مكانه الأصلي فلا يتبقى إلا عملية ترتيب صغيرة.

الكود البرمجي:

```
1. void InsertionSort(int a[],int n,int k)
2. {
4. int j;
4. for (int I=k;I<n;I++)
5. {
6. j=I;
7. while((j>=k)&&(a[j]<a[j-k]))
8. {
9. swap(a[j],a[j-k]);
10. j=j-k;
11. }
12. }}
14. void shellsort(int a[],int n)
14. {
15. int k=n/2;
16. while(k)
17. {
18. InsertionSort (a,n,k);
19. k=k/2;
20. }
21. }
```

إيجاد الـ Big-O للخوارزمية:

نلاحظ أن الخوارزمية السابقة تحتوي على ثلاث حلقات $while$ للتابع $shellsort$ والتي نستدعي ضمنها التابع $InsertionSort$ أي الحلقتين $while$ و for لذلك فإن الـ $big-oh$ للخوارزمية هو جداء الـ $big-oh$ للحلقات الثلاث:

- بسبب وجود التقسيم على 2 في كل مرة في حلقة $while$ الخارجية فتابع نموها $O(\log n)$
- حلقة for تدور n مرة فتابع النمو لها هو $O(N)$.
- أما في حلقة $while$ الداخلية فنلاحظ أن عدد العمليات للحلقة يتوقف على توزع القيم للمصفوفات الفرعية فنفرض أن الحلقة تدور A مرة حيث أن A مجهول.

فيكون الـ $Big-o$:

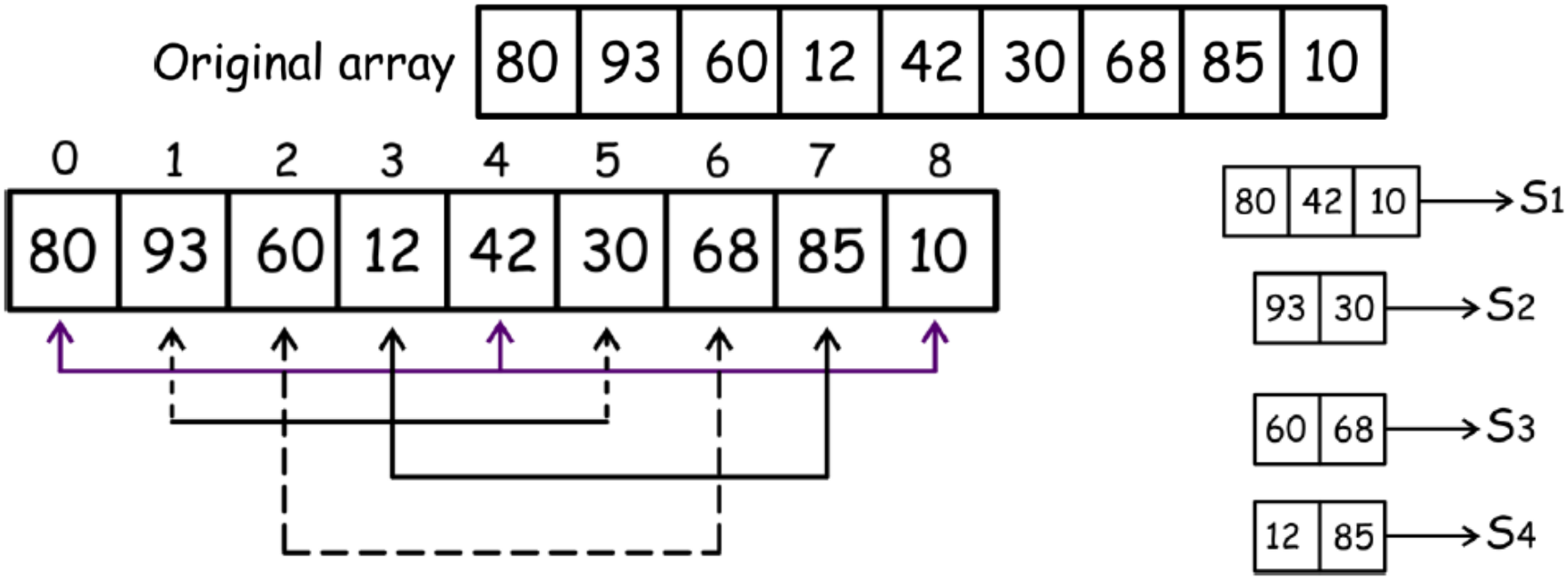
$$O(A * N * logN)$$

حيث أن A هو معامل مجهول.

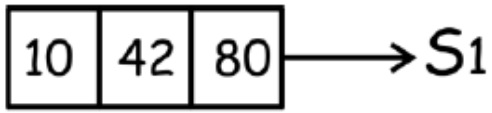
تجريبياً: نفرض أن $A = \log n$

$$O(n) = \log n * n * \log n = (n(\log n)^2) = n^{\frac{3}{2}}$$

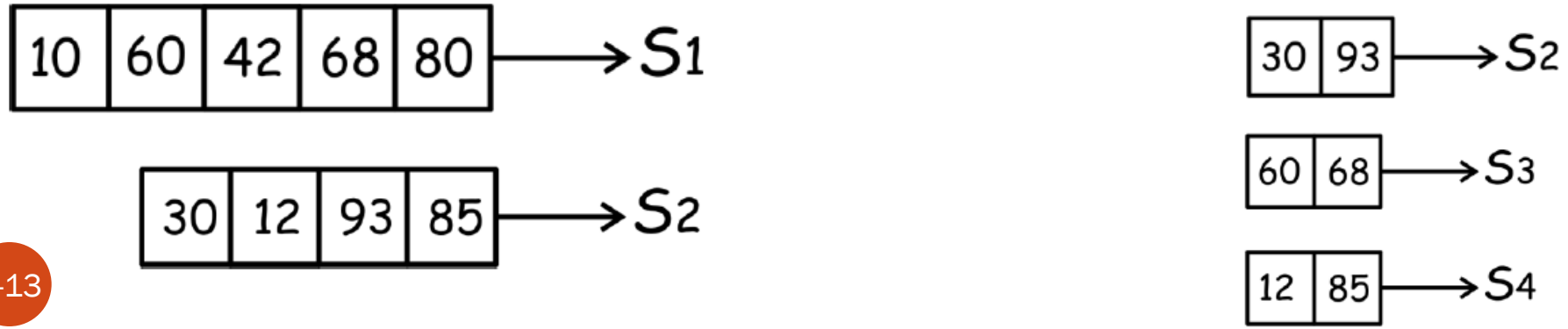
الشكل التالي يوضح الـ ***Shell Sort*** بشكل كبير:

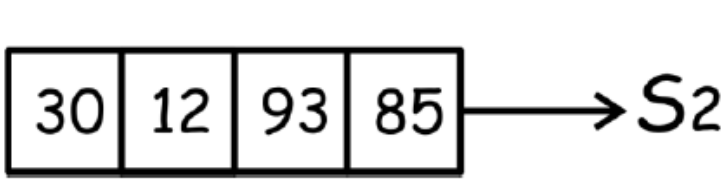


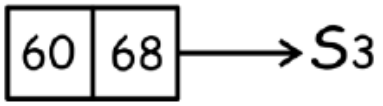
وبعد ترتيب المصفوفات الفرعية الصغيرة تصبح بالشكل:

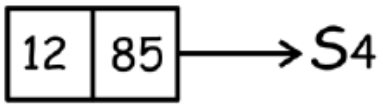


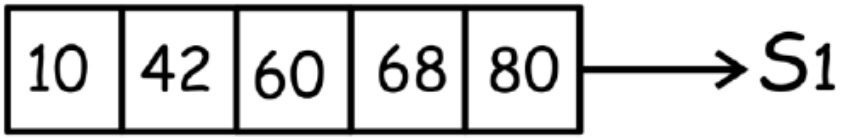
ويعاد تقسيم المصفوفة إلى مصفوفات أقل بعدد عناصر أكبر:



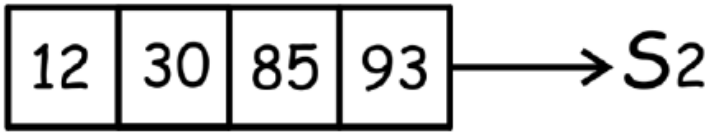




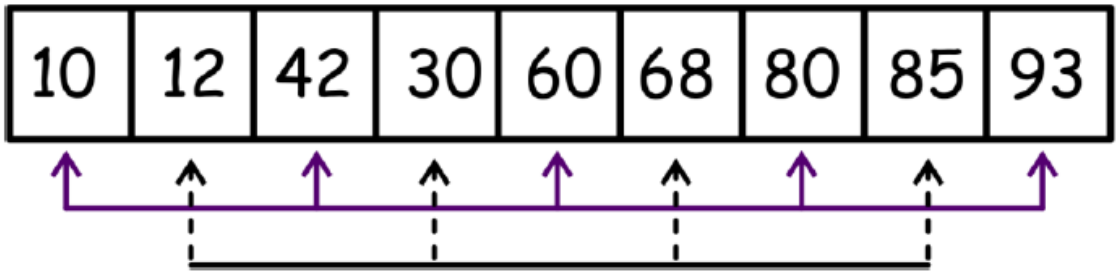




وبعد ترتيب المصفوفات الفرعية الصغيرة للتقسيم الثاني تصبح بالشكل:



ويعاد تقسيم المصفوفة عدة مرات إلى أن تترتب بشكل كامل:



حيث:

في المصفوفة الفرعية الأولى: الحد الأول هو $A[1]$، الثاني هو $A[k+1]$، الثالث $A[2k+1]$.

في المصفوفة الفرعية الثانية: الحد الأول هو $A[2]$، الثاني هو $A[k+2]$، الثالث $A[2k+2]$.

⋮

في المصفوفة الفرعية k: الحد الأول هو $A[k]$،الثاني هو $A[k+k]$، الثالث $A[2k+k]$.

حيث أن k في كل مرة تتغير ففي أول مرة تكون $\frac{n}{2}$ ثم $\frac{n}{4}$ ثم $\frac{n}{8}$ ثم $\frac{n}{16}$... إلخ.

الفرز السريع Quick sort

الترتيب السريع quicksort هي طريقة ترتيب من اختراع هوار (C.A.R.Hoare) في ١٩٦٢.

تعتمد الخوارزمية على وضع العنصر الأول (يسمى مؤشر) في مكانه النهائي ثم وضع العناصر الأكبر من المؤشر من جهة اليمين و العناصر الأصغر من جهة اليسار. و تسمى هذه العملية تجزئة. و بالنسبة لكل -جدول، نحدد مؤشرا جديدا و نعيد عملية التجزئة. تتكرر هذه العملية إلى أن نحصل على مجموعة مرتبة.

إذا تم اختيار المؤشر بطريقة صحيحة، نحصل على الطريقة الأسرع للترتيب في الحالة المتوسطة، مع تعقيد بـ $O(n \log n)$ و التي قد تتحول إلى $O(n^2)$ في الحالة الأصعب، و هي حالة جدول عناصره مرتبة أصلا. و لكن هذه الحالة بديهية لأن المجموعة مرتبة أصلا.

Quick Sort

في الناحية العملية، بالنسبة للتجزئات مع عدد قليل لا يتجاوز بضع عشرات من العناصر، يتم اللجوء عادة إلى الترتيب بالإدراج الذي يكون أفضل من الترتيب السريع. و بصفة عامة يعتبر الترتيب السريع الأكثر شيوعا (شعبية) من بين جميع خوارزميات الترتيب، و المشكلة الوحيدة تتمثل في كيفية اختيار المؤشر.

تعتبر من الخوارزميات الأسرع بشكل عام،و الأكثر تعقيدا" .

عند استعمال الترتيب السريع لترتيب مجموعة ذات عناصر كبيرة، يمكن تغيير تقنية الترتيب عند الوصول إلى مجموعة جزئية غير مرتبة عدد عناصرها صغير، ١٠ أو أقل. الترتيب بالاختيار مناسب في هذه الحالة.

Quick Sort

تتألف هذه الطريقة من أربع خطوات :

في حال وجود عنصر واحد فقط مراد ترتيبه ضمن اللائحة يتم إعادته مباشرة.

اختيار عنصر من اللائحة ليكون نقطة pivot للمصفوفة و عادة يكون أول عنصر من يسار اللائحة.

تقسيم اللائحة إلى نصفين أحدهما ذات قيم أصغر من القيمة المرجعية لنقطة pivot و النصف الآخر يحتوي على العناصر الأكبر من القيمة المرجعية لنقطة pivot.

تكرار العملية مع كل من النصفين اللذين تم تشكيلهم من المرحلة السابقة .

Quick Sort

خطوات خوارزمية الفرز السريع

1. اختيار مؤشر "pivot "و وضعه في مكانه النهائي من القائمة " في الوسط تقريبا ".
2. ترتيب بقية عناصر المصفوفة بحيث تكون العناصر الأكبر على يمين المؤشر و العناصر الأصغر على يسار المؤشر ، بنهاية هذه الخطوة سيكون المؤشر في مكانه الصحيح
4. نجزيء المصفوفة إلى مصفوفتين يسار المؤشر و يمين المؤشر .
4. نكرر الخطوات 1 و 2 و 4 على كلا المصفوفتين ، و هكذا حتى يتم ترتيب المصفوفة .

ستواجهنا هنا اشكالية اختيار المؤشر ، حيث يمكن إن يكون المؤشر أصغر قيمة على أسوء حال ، في هذه الحالة سيكون معدل العمليات اللازمة هو n مرفوعة للأس 2 . و هي حالة سيئة .

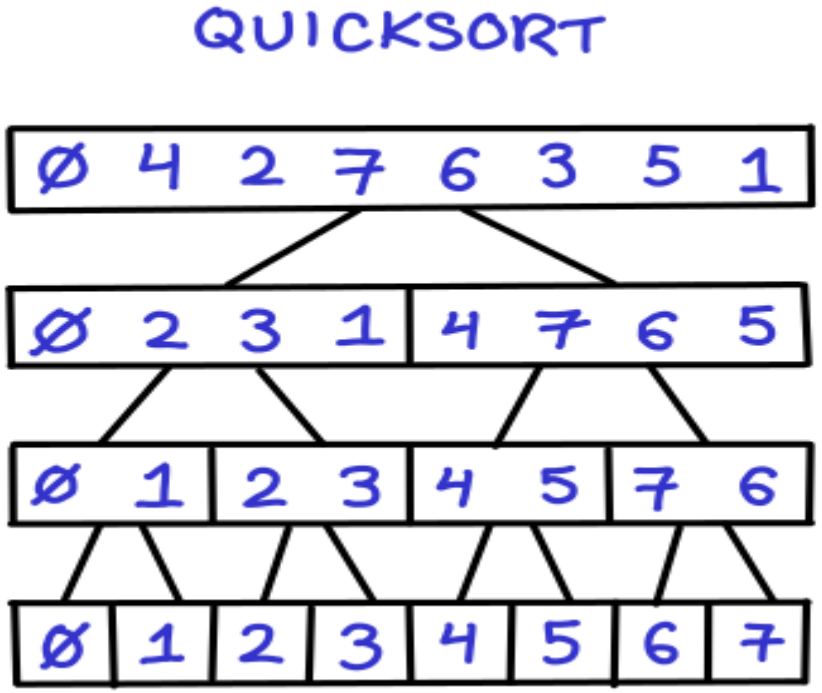
و يمكن التغلب على هذه المشكلة بعدة طرق منها :

1. اختيار المؤشر ليكون العنصر الأوسط في المصفوفة بدلا من الأول .

2 . اختيار المؤشر بشكل عشوائي في كل مرة ..

4 . اختيار المؤشر كمتوسط بين أول و أوسط و آخر عنصر في المصفوفة .

و الخيار الثالث هو الأمثل ..



- تعتمد هذه الخوارزمية على اختيار حد معين من المصفوفة ونسميه ***pivot*** (مثلاً) ثم نقوم بنقل جميع الأعداد التي أصغر منه قبله و نقل جميع الأعداد التي أكبر منه بعده وبالتالي يستقر ***pivot*** في موقعه الصحيح.
- وبعد هذه العملية يقوم التابع باستدعاء نفسه في المجالين "الأصغر من ***pivot*** والأكبر منه" ونستمر بهذه الاستدعاءات إلى أن نصل إلى نقطة خروج ونقطة الخروج هي أن تكون المجموعة الجديدة فارغة أو تحتوي على عنصر وحيد.

الكود لهذه الخوارزمية:

```
1. int quicksort (int a[], int first, int last)
2. {
4. if (first>=last)
4. return;
5. int split (partition (a, first, last));
6. quicksort (a, first, split-1);
7. quicksort (a, split+1, last);
8. }
```

```
1. int partition (int a[], int first, int last){
2. int lastsmall(first),i;
4. for (i=first+1;i<=1last;i++)
4. if (a[i]<=a[first]){
5. last small++;
6. swap Elements (a, lastsmall, i);}
7. swap Elements (a, first, lastsmall);
8. return lastsmall;
9. }
```

مثال : [67, 58, 38, 81, 90, 57, 54]