كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع الثاني

فعالية الخوارزمية 2

Algorithm Efficiency 2

الفصل الثاني 2023-2024

4-6- خوارزميات الترتيب 1

1-4-6- مفهوم الترتيب:

لتكن لدينا المجموعات التالية من البيانات :

1- {25,40,2,-6,20,100,1,11,0,15,11}

2- { أحمد , كرم , اسماعيل , مريم , نادين , خالد , حسن , معروف , أمل }

3- {1.4,10e-7,3.14,70.0,23.5,1e-5,2.1e+4}

حيث تمثل كل مجموعة منها سلسلة من البيانات المتجانسة في النوع (أعداد صحيحة , سلاسل حرفية , أعداد كسرية –على الترتيب).

نقصد بالترتيب(Sorting or Ordering) الحصول على عناصر السلسلة ذاتها بترتيب جديد تكون فيه العناصر متتالية وفق معيار معين حيث أن علاقة ترتيب معينة معرفة على مجموعة العناصر بحيث يمكن الحديث ضمن المجموعة الواحدة عن مفهوم -العنصر التالي- ومفهوم العنصر السابق- حيث يمكن ترتيب القيم العددية تصاعدياً (ascending order) وفق تسلسل القيم المتزايدة للعدد وقد يكون الترتيب تنازليا (descending order) وفق القيم المتناقصة للأعداد. وترتيب عناصر المجموعة الثانية ترتيباً أبجدياً (alphabetical order) (ترتيب تصاعدي) أو ترتيباً أبجديا معكوساً (ترتيب تنازلي) لتكون نتيجة الترتيب التصاعدي للمجموعات الثلاثة السابقة كالآتي:

4-6- خوارزميات الترتيب 2

-{-6,0,1,2,11,11,15,20,25,40,100}

2- { أحمد , إسماعيل ,حسن , خالد , مازن , مريم , معروف , نادر, نادين }

3- {10e-7,1.8e-5,1.4, 3.14,23.5,70.0,2.1e+4}

وتعتبرعملية الترتيب هذه من أهم الأعمال التي تقوم بها الحواسيب فحسب بعض المراجع كانت الحواسيب المركزية الضخمة المعروفة بـ Mainframe تصرف ربع زمن تشغيلها في عمليات ترتيب البيانات.

وعادة ما يتخرج طلاب المعلوماتية من كلياتهم وقد درسوا خوارزميات الترتيب الأساسية في مقرركمقررمن قبيل "مقدمة إلى البرمجة" ثم في مقررآخرمن قبيل "بنى المعطيات" Data Structures ومن ثم قد يدرسونها بشكل مستقل في مقررات الخوارزميات , فلماذا كل هذا الاهتمام بخوارزميات.

وفي معرض الإجابة على هذا التساؤل نجد النقاط التالية:

1-تعتبرخوارزميات الترتيب لبنة أساسية في بناء العديد من الخوارزميات الأخرى الهامة, وبفهم خوارزميات الترتيب يحصل الطالب على قدر مذهل من المهارة والقدرة لحل مشاكل أخرى.

SORT ALGORITHMS 3

4-6- خوارزميات الترتيب 3

2-كما ذكرنا سابقاً فتاريخياً كان جزء لا يستهان به (الربع تقريباً) من الوقت الذي تصرفه الحواسيب على حل المسائل المتنوعة يذهب لعمليات الترتيب.

3-خوارزميات الترتيب هي من أكثر المسائل التي تمت دراستها وتحليلها بشكل كامل في علوم الحاسب , ويمكننا القول جازمين أن هناك عدداً كبيراً من خوارزميات الترتيب المتنوعة المعروفة, ويتفوق بعضها على البعض الآخر في حالات بعينها, ويكفي القارئ أن يعود إلى أحد المراجع المتخصصة بخوارزميات الترتيب فقط ليجد الكم الكبير منها.

4-في سياق الحديث عن تصميم وتحليل خوارزميات الترتيب يتم التعامل مع العديد من الأفكار المثيرة والمشوقة كـ "فرِّق تسُدْ" divide-and-conquer و"الخوارزميات العشوائية" randomized algorithms و"الخوارزميات العودية" recursive algorithms وخوارزميات الترتيب بالدمج mirage sort، الترتيب بالحشر insert sort.

وسنقوم فيما يلي من فقرات بدراسة بعض أهم خوارزميات الترتيب الأساسية مستفيضين في شرح بعضها ومختصرين في البعض الآخر وذلك بما يناسب شريحة الطلاب الذين نوجه هذا المساق اليهم, وسيتم التركيز على حسابات BON لكل خوارزمية,
فخوارزميات الترتيب تشكل ساحة واسعة لعمليات الحساب هذه مع عمليات المقارنة بين الخوارزميات المختلفة في درجة تعقيدها والحالات الأسوأ والأفضل.

BUBBLE SORT ALGORITHM 1

6-4-2- خوارزمية الترتيب الفقاعي 1

تعتبر خوارزمية الترتيب الفقاعي من أبسط الخوارزميات المستخدمة في الترتيب على الإطلاق (و أقلها كفاءة) وتفترض هذه الخوارزمية وجود المعطيات المراد ترتيبها في بنية خطية ذات وصول حسب الدليل (نسق أحادي البعد مثلاً) ولترتيب عناصر النسق ذي الـ N عنصرا يتم المرور على النسق (N-1) مرة ابتداء من العنصر الأول وحتى العنصر ما قبل الأخير في كل مرور وتتم مقارنة كل عنصر أثناء المرور بالعنصر التالي له فإذا كان العنصران اللذان تتم مقارنتهما بحاجة تبديل يتم تبديل العنصرين .
والنص التالي يشرح هذه الخوارزمية (6-4) حتى يتسنى لنا حساب درجة تعقيدها:

BUBBLE SORT ALGORITHM

(* Algorithm to sort the list A[1],...A[n] in ascending order. *)

1. set scan equal to 1

2. while scan less or equal to n-1 do the following:

a. set pos equal to 1.

b. while pos less or equal to n-1 do the following:

b1. if A[pos] > A[pos+1] then do the following:

(* swap A[pos] ↔ A[pos+1])

BUBBLE SORT ALGORITHM 2

6-4-2- خوارزمية الترتيب الفقاعي 2

b11. set temp equal to A[pos]

b12. set A[pos] equal to A[pos+1]

b13 set A[pos+1] equal to temp

b2.set pos equal to pos+1

c. set scan equal to scan+1

وسنرى تأثير هذه الخوارزمية على النسق التالي كمثال:

25	20	16	12	7	5	2	1
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

- المسح الأول (scan = 1 ; pos = 1 .. N-1) مبين في الجدول (3-6)

BUBBLE SORT ALGORITHM 3

6-4-2- خوارزمية الترتيب الفقاعي 3

1	2	3	4	5	6	7	8		
25	20	16	12	7	5	2	1	scan=1;pos=1	نبادل 1و2
20	25	16	12	7	5	2	1	scan=1;pos=2	نبادل 2و3
20	16	25	12	7	5	2	1	scan=1;pos=3	نبادل 3و4
20	16	12	25	7	5	2	1	scan=1;pos=4	نبادل 4و5
20	16	12	7	25	5	2	1	scan=1;pos=5	نبادل 5و6
20	16	12	7	5	25	2	1	scan=1;pos=6	نبادل 6و7
20	16	12	7	5	2	25	1	scan=1;pos=7	نبادل 7و8
20	16	12	7	5	2	1	25	scan=1;pos=8	نهاية المسح الأول

بين الجدول (6-3) المسح الأول لخوارزمية الترتيب الفقاعي

وهكذا نجد أنه في نهاية المسح الأول أصبح أكبر العناصر وهو 25 في موقعه الصحيح وقد قمنا بـ N-1=7 عمليات تبديل في الحالة الأسوأ حيث كان العنصر الأكبر في بداية النسق. وسنتابع مع المسح الثاني:

- المسح الثاني (scan = 2 ; pos = 1 .. N-1) مبين في الجدول (6-4):

BUBBLE SORT ALGORITHM 4

6-4-2- خوارزمية الترتيب الفقاعي 4

1	2	3	4	5	6	7	8		
20	16	12	7	5	2	1	25	scan=2;pos=1	نبادل 1و2
16	20	12	7	5	2	1	25	scan=2;pos=2	نبادل 2و3
16	12	20	7	5	2	1	25	scan=2;pos=3	نبادل 3و4
16	12	7	20	5	2	1	25	Scan=2;pos=4	نبادل 4و5
16	12	7	5	20	2	1	25	scan=2;pos=5	نبادل 5و6
16	12	7	5	2	20	1	25	scan=2;pos=6	نبادل 6و7
16	12	7	5	2	1	20	25	scan=2;pos=7	لا نبادل 7و8
16	12	7	5	2	1	20	25	scan=2;pos=8	نهاية المسح الثاني

بين الجدول (6-4) المسح الثاني لخوارزمية الترتيب الفقاعي

وهكذا نجد أنه في نهاية المسح الثاني أصبح ثاني أكبر العناصر وهو 20 في موقعه الصحيح وقد قمنا بـ N-2=6 عمليات تبديل في الحالة الأسوأ حيث كان العنصر في بداية النسق.

وسنتابع مع المسوحات الباقية الثالث والرابع ....حتى السابع كما في الجدول (6-5)

BUBBLE SORT ALGORITHM 5

6-4-2- خوارزمية الترتيب الفقاعي 5

	1	2	3	4	5	6	7	8
نهاية المسح الثالث	12	7	5	2	1	16	20	25
نهاية المسح الرابع	7	5	2	1	12	16	20	25
نهاية المسح الخامس	5	2	1	7	12	16	20	25
نهاية المسح السادس	2	1	5	7	12	16	20	25
نهاية المسح السابع	1	2	5	7	12	16	20	25

الجدول (6-5) يبين نهايات المسوحات الباقية

وهكذا وبعد (N-1)*(N-1) عملية مقارنة (مع عدد من عمليات التبديل يكون في أسوأ الاحتمالات يبدأ بـ (N-1) ثم ينقص واحداً في كل مرة) يكون النسق قد أصبح مرتبا.

يعود سبب تسمية هذه الخوارزمية بـ"خوارزمية الترتيب الفقاعي" Bubble Sort Algorithm حيث نلاحظ أنه في كل مرور كامل على النسق يصل أحد العناصر الكبيرة إلى موقعه الصحيح بدءاً بالأكبر بحركة تشبه حركة فقاعات الماء عند بدء غليانه حيث تتشكل فيه بعض الفقاعات الهوائية في الأسفل ثم ما تلبث بالصعود إلى السطح وتكون الفقاعات الأكبر هي الواصلة أولا إلى السطح.

وسنقوم الآن بحساب درجة تعقيد هذه الخوارزمية: نيبين عدد مرات تكرار كل عبارة من الخوارزمية المذكورة وفق الحالة الأسوأ

BUBBLE SORT ALGORITHM 6

6-4-2- خوارزمية الترتيب الفقاعي 6

statement	# of times executed
1	1
2	N
a	N-1
b	$N \times (N-1)$
b1	$(N-1) \times (N-1)$
b11	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b12	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b13	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b2	$(N-1) \times (N-1)$
c	N-1
total:	$\frac{9}{2}N^2-\frac{7}{2}N+1$

BUBBLE SORT ALGORITHM 6

statement	# of times executed
1	1
2	N
a	N-1
b	$N \times (N-1)$
b1	$(N-1) \times (N-1)$
b11	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b12	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b13	$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$
b2	$(N-1) \times (N-1)$
c	N-1
total:	$\frac{9}{2}N^2-\frac{7}{2}N+1$

6-4-2- خوارزمية الترتيب الفقاعي 6

الجدول (6-6) يبين عدد مرات تكراركل عبارة من خوارزمية الفقاعي.
ونستطيع التعبيرعن زمن التنفيذ باستخدام تدوين BON كما يلي:

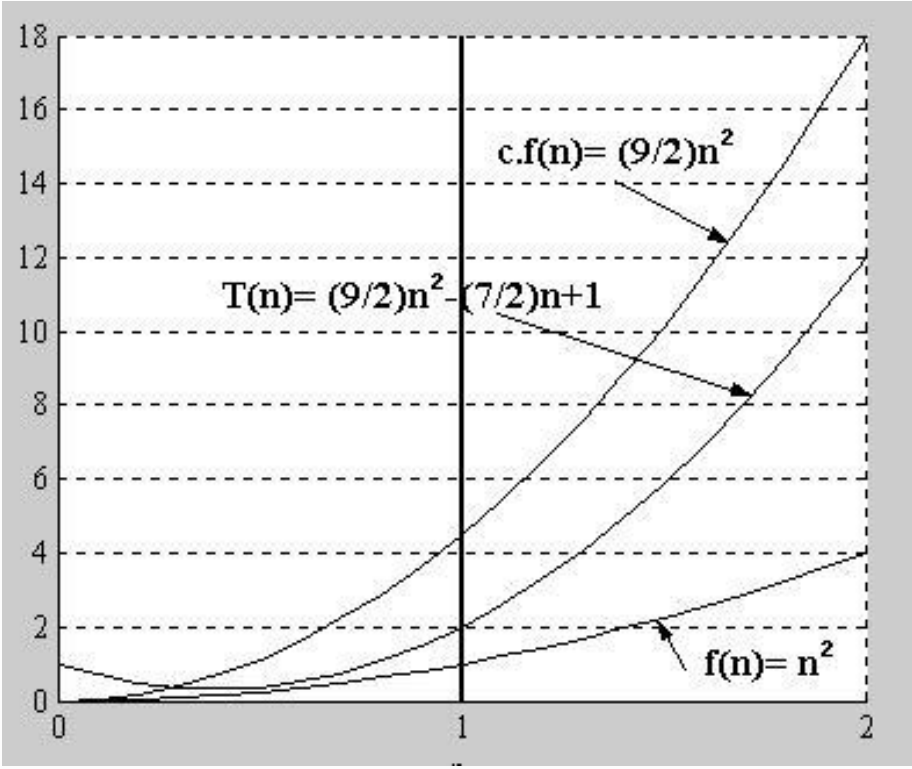
$$T(n)=\frac{9}{2}N^2-\frac{7}{2}N+1 \Rightarrow T(n)=O(n^2)$$

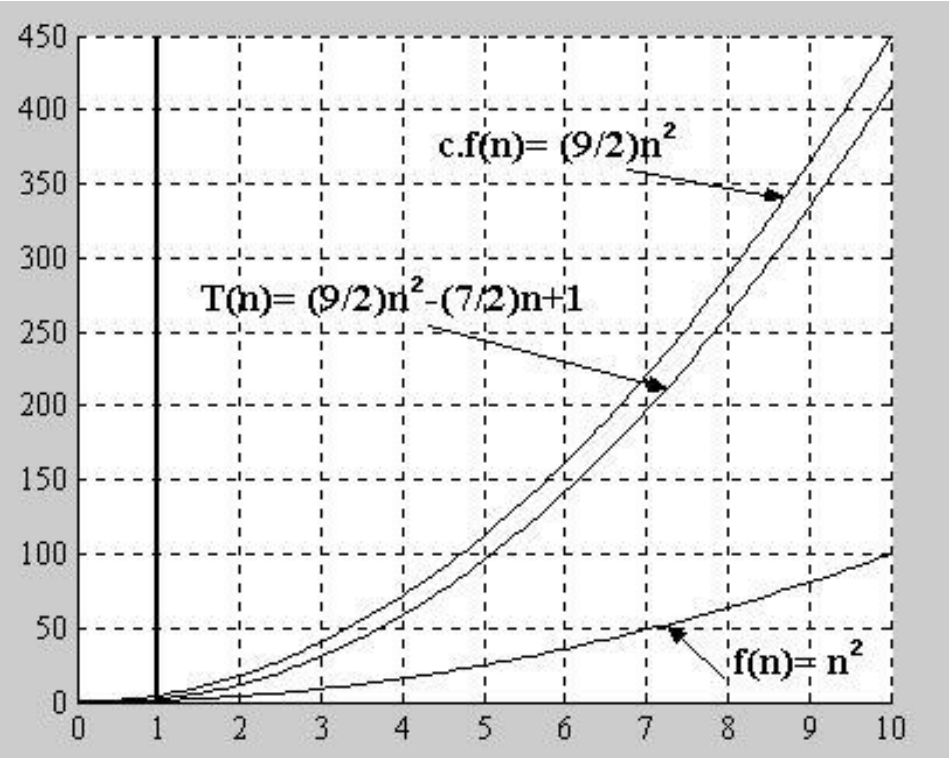
$$\frac{9}{2}N^2-\frac{7}{2}N+1 \le \frac{9}{2}N^2 \; for \;\; all \;\; N \ge 1$$

أي أن $f(n)=n^2$, $C=9/2$, $n_0=1$ ويبين الشكل (6-6) الخط البياني لكل من التوابع f(n),T(n),c.f(n):

BUBBLE SORT ALGORITHM 7

6-4-2- خوارزمية الترتيب الفقاعي 7





الشكل (6-6) يبين أن $T(n) <= 9/2.f(n)$ عندما $n >= 1$

والبرنامج (6-4) التالي يحقق هذه الخوارزمية

BUBBLE SORT ALGORITHM 8

6-4-2- خوارزمية الترتيب الفقاعي 8

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void main(void)
{    const int n=8;
     int A[n]={ 25,20,16,12,7,5,2,1};
     cout<<"A before sorting : ";
     for (int i=0;i<n;i++)   cout<<setw(4)<<A[i];

     cout<<endl;
     int scan,pos;
     scan = 1;
```

BUBBLE SORT ALGORITHM 9

6-4-2- خوارزمية الترتيب الفقاعي 9

```
while(scan<=n-1)
    {   pos=0;
        while(pos<n-1)
        {  if(A[pos]> A[pos+1])
            { int temp = A[pos]  ;  A[pos]  = A[pos+1];
                A[pos+1]= temp    ;
            }pos++;
        }scan ++ ;
    }
    cout<<"A after  sorting : ";
    for (int i=0;i<n;i++) cout<<setw(4)<<A[i];
    cout<<endl;
system("pause");
}
```

BUBBLE SORT ALGORITHM 10

6-4-2- خوارزمية الترتيب الفقاعي 10

وستقدم الآن بعض التطويرات البسيطة على خوارزمية الترتيب الفقاعي :

لقد وجدنا أن الترتيب الفقاعي يتم بمسح على (N-1) عنصراً وبتكرار عملية المسح (N-1) مرة ولكن لو عدنا إلى الجدول (3-6) لوجدنا أنه عند نهاية المسح الأول يكون أكبر العناصر قد توضع في مكانه الصحيح (في آخر النسق) ولذلك لا داعي لمقارنة العنصرين (N,N-1) عند المسح الثاني وعند نهاية المسح الثاني يصبح ثاني أكبر العناصر في موضعه الصحيح ولذلك لا داعي لمقارنة العنصرين (N-1,N-2) عند المسح الثالث وهكذا دواليك الأمر الذي يمكننا من إعادة صياغة الخوارزمية (5-6) بشكل فعال أكبر كالتالي:

يستبدل السطر التالي: **b. while pos less or equal to n-1 do the following** بالسطر

b. while pos less or equal to n-scan do the following

ولكن لو عدنا إلى حساب درجة التعقيد لوجدنا أن التغيير سيصبح في عدد مرات تنفيذ كل من العبارتين **b,b1** فقط, حيث (في الحالة الأسوأ) سيصبح :لكل من **b** و **b1** علي الترتيب

$$(N)+(N-1)+....+2=\sum_{scan=1}^{scan=N-1}(N-scan+1)=\frac{(N+2)\times(N-1)}{2}$$

$$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$$

6-4-2- خوارزمية الترتيب الفقاعي 11

وعند حساب العدد الإجمالي للعمليات المنفذة سنحصل على تابع من الشكل: $T(n)=a.N^2+b.N+c$

وهو لا يختلف عن التابع السابق إلا بقيم الثوابت أي مازالت الخوارزمية من الدرجة N^2 أي $O(n)=f(N^2)$ وعمليا عند عدد عناصر كبير سنلاحظ تحسنا بسيطا في زمن التنفيذ عن الحالة السابقة (عندما يكون النسق في حالته الأسوأ أو الحالات القريبة منها فقط) حيث إن التوفير تم في زمن تنفيذ عبارات تتضمن عمليات مقارنة فقط وهي عمليات تحتاج لزمن بسيط في معظم الآلات خاصة إذا ما قارنا هذا الزمن بعمليات نسب المعطيات والتي ستتكرر ثلاث مرات لتأمين عملية تبديل عنصرين والتي ستكون أكبر بكثير في البرامج الفعلية حيث ستكون المعطيات سجلات من البيانات بدلا من أعداد بسيطة كما في مثالنا السابق.

ولتعديل البرنامج الذي ينفذ الخوارزمية ليشمل التحسين (A) الذي درسناه ما علينا إلا استبدال السطر:

while(pos<<u>n-1</u>) بالسطر: while(pos<<u>n-scan</u>)

يمكن تقديم تحسين آخر على خوارزمية الترتيب الفقاعي السابقة تماشيا مع حقيقة أن الحالة الأسوأ نادرة الحدوث (ذات احتمال صغير) حيث أنه يكفي في بعض الحالات القيام بعدد أقل من المسوحات ليصبح النسق مرتبا وعندها يجب التوقف عن عمليات المسح المتبقية, فمثلا لو كان النسق المعطى يملك القيم التالية كما في الجدول (6-7) :

BUBBLE SORT ALGORITHM 12

6-4-2- خوارزمية الترتيب الفقاعي 12

	1	2	3	4	5	6	7	8
النسق الأصل(حالة ليست بالأسوأ)	2	1	25	12	7	16	20	5
نهاية المسح الأول	1	2	12	7	16	20	5	25
نهاية المسح الثاني	1	2	7	12	16	5	20	25
نهاية المسح الثالث	1	2	7	12	5	16	20	25
نهاية المسح الرابع	1	2	7	5	12	16	20	25
نهاية المسح الخامس	1	2	5	7	12	16	20	25
نهاية المسح السادس	1	2	5	7	12	16	20	25
نهاية المسح السابع	1	2	5	7	12	16	20	25

الجدول (6-7) يبين حالة يتم فيها ترتيب النسق بعدد قليل من المسوحات

حيث نلاحظ أن النسق أصبح مرتبا اعتبارا من نهاية المسح الرابع مما جعل بقية المسوحات تتم دون القيام بأية عملية تبديل ولو لاحظنا مثالا آخر خاصة إذا كان النسق في حالته الفضلى (عندما يكون مرتبا منذ البداية) أو قريباً من الحالة الفضلى (شبه مرتب) كالمثال التالي الجدول (6-8):

BUBBLE SORT ALGORITHM 13

6-4-2- خوارزمية الترتيب الفقاعي 13

	1	2	3	4	5	6	7	8
النسق الأصل(حالة ليست بالأسوأ)	1	2	7	5	12	25	16	20
نهاية المسح الأول	1	2	5	7	12	16	20	25
نهاية المسح الثاني	1	2	5	7	12	16	20	25
نهاية المسح الثالث	1	2	5	7	12	16	20	25
نهاية المسح الرابع	1	2	5	7	12	16	20	25
نهاية المسح الخامس	1	2	5	7	12	16	20	25
نهاية المسح السادس	1	2	5	7	12	16	20	25
نهاية المسح السابع	1	2	5	7	12	16	20	25

الجدول (6-8) يبين حالة يتم فيها ترتيب النسق بعدد اقل من المسوحات

حيث نلاحظ هنا أن النسق أصبح مرتبا اعتباراً من نهاية المسح الأول ولم تتم بعد ذلك أية عملية تبديل, ويمكن تحسين الخوارزمية لتضمن التوقف عن مسح النسق عندما يصبح مرتبا ويتم كشف كون النسق مرتبا بعد عملية مسح لا يجري فيها أي تبديل ولذلك عند بداية كل مسح نضع قيمة منطقية swap_have_made بالقيمة false وعندما يتم أي تبديل نغيرها إلى true ويتم إدخالها في شرط استمرار المسح كما تبين سطور الخوارزمية (6-6)

BUBBLE SORT ALGORITHM 14

6-4-2- خوارزمية الترتيب الفقاعي 14

BUBBLE SORT ALGORITHM (B)

(* Algorithm to sort the list A[1],....A[n] in ascending order. *)

1. set scan equal to 1

2. set swap_have_made equal to true

3. while (scan less or equal to n-1) and (swap_have_made) do the following:

- **a. set pos equal to 1.**
- **b. set swap_have_made equal to false**
- **c. while pos less or equal to n-scan do the following:**
  - **c1. if A[pos] > A[pos+1] then do the following:**
    - **(* swap A[pos] ←→ A[pos+1])**
    - **c11. set temp equal to A[pos]**
    - **c12. set A[pos] equal to A[pos+1]**
    - **c13. set A[pos+1] equal to temp**
    - **c14. set swap_have_made equal to true**
  - **c2.set pos equal to pos+1**
- **d. set scan equal to scan+1**

6-4-2- خوارزمية الترتيب الفقاعي 15

ملاحظة: العبارة 2 تم وضعها لتأمين الدخول إلى الحلقة عند البداية فقط.

وأما من ناحية حساب درجة التعقيد فما تزال الخوارزمية من مرتبة N^2 كما أنها ستكون أقل فعالية في الحالة الأسوأ أو الحالات القريبة منها وذلك بسبب إضافة اختبار شرط جديد مع إضافة عملية and المنطقية وإضافة العبارة c14 التي ستنفذ عند كل تبديل يتم, ولذلك يفضل اللجوء إلى هذه الخوارزمية فقط في الحالات التي يكون احتمال كون العناصر مرتبة أو شبه مرتبة كبيرا (كمثال نذكر حالة إضافات متكررة لعنصر إلى لائحة تم ترتيبها مسبقاً مع إعادة الترتيب بعد كل إضافة للمحافظة على الترتيب).

والبرنامج(6-5) التالي ينفذ الخوارزمية المحسنة (B)مع تزويده بتابعين input و output لتمكين المستخدم من إدخال القيم كما يقوم البرنامج بطباعة النسق قبل و بعد الترتيب وبعد نهاية كل مسح:

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[])
{    for(int i=0;i<n;i++) {cout<<"input "<<name<<"["<<i+1<<"] : ";cin>>a[i];}}
```

BUBBLE SORT ALGORITHM 16

6-4-2- خوارزمية الترتيب الفقاعي 16

```
void output(int a[],int n)
{    cout<<"[ " ;
    for (int i=0;i<n;i++)cout<<setw(4)<<a[i];    cout<<" ]"<<endl;}
void main(void)
{   const int n=8;      int A[n];
    input(A,n,"A");
    cout<<"Before sorting : ";    output(A,n);    cout<<endl;
    int scan,pos,temp,swap_have_made;
    scan = 1;     swap_have_made = 1 ; // true
    while((scan<=n-1) && (swap_have_made))
    {   pos=0;        swap_have_made = 0 ; // false
        while(pos<n-scan)
        { if(A[pos]> A[pos+1]) {   temp    = A[pos]  ; A[pos]  = A[pos+1];
                A[pos+1]= temp    ;          swap_have_made = 1 ; // true
        } pos++;
     }
```

BUBBLE SORT ALGORITHM 17

6-4-2- خوارزمية الترتيب الفقاعي 17

```
cout<<"scan = "<<scan<<" : ";
        output(A,n);
        scan ++ ;
    }    cout<<endl;
    cout<<"After  sorting : "; output(A,n);    cout<<endl;
system("pause");
}
```

ولفهم أعمق يطلب من الطالب تعديل البرنامج ليطبع النسق في بداية كل مقارنة يقوم بها حيث نستطيع معاينة عمليات التبديل التي تتم كما يتم طباعة النسق في نهاية كل مسح مع طباعته قبل الترتيب وبعد انتهاء الترتيب يمكن العودة للكتاب لنص البرمجي (6-6) .

SWAP SORT ALGORITHM 1

6-4-3- خوارزمية الترتيب بالتبديل 1

6-4-3- خوارزمية الترتيب بالتبديل : SWAP SORT ALGORITHM

تعتمد خوارزمية الترتيب بالتبديل على المرور على جميع العناصر الموجودة في المو اقع ذوات الأرقام من 1 وحتى n-1 , ويتم من أجل كل موقع منها مقارنة هذا العنصر مع جميع العناصر التالية له ضمن النسق وإجراء المبادلة بين العنصرين الجارية مقارنتهما عندما يكونان متوضعين بترتيب غير صحيح. وهكذا نستطيع كتابة نص الخوارزمية (6-7) كالتالي مفترضين أن العناصر مخزنة في نسق أحادي ويراد ترتيبها تصاعديا , كما فعلنا عند دراستنا لخوارزمية الترتيب الفقاعي:

SWAP SORT ALGORITHM

(* Algorithm to sort the list A[1],...A[n] in ascending order. *)

1. set first equal to 1
2. while (first less or equal to n-1) do the following:
    a. set second equal to first +1 .
    b. while second less or equal to n do the following:
        b1. if A[first] > A[second] then do the following:
            (* swap A[first] ⇔ A[second])
            b11. set temp equal to A[first] b12. set A[first] equal to A[second] b13. set A[second] equal to temp
        b2.set second equal to second +1
    c. set first equal to first +1

6-4-3- خوارزمية الترتيب بالتبديل 2

- المسح الأول (first = 1 ; second = 2 .. N): الجدول(6-9)

1	2	3	4	5	6	7	8		
25	20	16	12	7	5	2	1	first=1;second=2	نبادل 1و2
20	25	16	12	7	5	2	1	first=1;second=3	نبادل 1و3
16	25	20	12	7	5	2	1	first=1;second=4	نبادل 1و4
12	25	20	16	7	5	2	1	first=1;second=5	نبادل 1و5
7	25	20	16	12	5	2	1	first=1;second=6	نبادل 1و6
5	25	20	16	12	7	2	1	first=1;second=7	نبادل 1و7
2	25	20	16	12	7	5	1	first=1;second=8	نبادل 1و8
1	25	20	16	12	7	5	2	first=2;second=9	نهاية المسح الأول

الجدول(6-9) المسح الأول لخوارزمية الترتيب بالتبديل

وهكذا نجد أنه في نهاية المسح الأول أصبح أصغر العناصر وهو 1 في موقعه الصحيح وقد قمنا بـ N-1=7 من عمليات المبادلة في الحالة الأسوأ حيث كان العنصر الأصغر في نهاية النسق. وسنتابع مع المسح الثاني:

6-4-3- خوارزمية الترتيب بالتبديل 3

- المسح الثاني (first = 2 ; second = 3 .. N): الجدول(6-10)

1	2	3	4	5	6	7	8		
25	20	16	12	7	5	2	1	first=1;second=2	نبادل 1و2
20	25	16	12	7	5	2	1	first=1;second=3	نبادل 1و3
16	25	20	12	7	5	2	1	first=1;second=4	نبادل 1و4
12	25	20	16	7	5	2	1	first=1;second=5	نبادل 1و5
7	25	20	16	12	5	2	1	first=1;second=6	نبادل 1و6
5	25	20	16	12	7	2	1	first=1;second=7	نبادل 1و7
2	25	20	16	12	7	5	1	first=1;second=8	نبادل 1و8
1	25	20	16	12	7	5	2	first=2;second=9	نهاية المسح الأول

الجدول(6-10) المسح الثاني لخوارزمية الترتيب بالتبديل

وهكذا نجد أنه في نهاية المسح الأول أصبح <u>ثاني أصغر العناصر</u> وهو 2 في موقعه الصحيح وقد قمنا بـ N-2=6 من عمليات المبادلة. وسنتابع مع المسوحات الباقية الثالث والرابع ....حتى السابع كما في الجدول(6-11):

SWAP SORT ALGORITHM 4

6-4-3- خوارزمية الترتيب بالتبديل 4

	8	7	6	5	4	3	2	1
نهاية المسح الثالث	7	12	16	20	25	5	2	1
نهاية المسح الرابع	12	16	20	25	7	5	2	1
نهاية المسح الخامس	16	20	25	12	7	5	2	1
نهاية المسح السادس	20	25	16	12	7	5	2	1
نهاية المسح السابع	25	20	16	12	7	5	2	1

الجدول(6-11) نهايات المسح الباقية

وهكذا يتم في كل مسح انتقال أصغر عناصر النسق –بدءا من موقع البداية الحالي وحتى نهاية النسق-إلى موقعه الصحيح مع عدد من عمليات المقارنة (وعدد مماثل من عمليات المبادلة في أسوأ الأحوال) يبدأ بـ (N-1) ثم ينقص واحداً في كل مرة.

وسنقوم الآن بحساب درجة تعقيد هذه الخوارزمية: الجدول(6-12) التالي يبين عدد مرات تكرار كل عبارة من الخوارزمية المذكورة سابقاً وذلك في الحالة الأسوأ(أي عندما يكون النسق أصلاً مرتبا ترتيبا عكسيا لما هو مطلوب):

SWAP SORT ALGORITHM 5

6-4-3- خوارزمية الترتيب بالتبديل 5

statement	# of times executed
1	1
2	N
a	N-1
b	$N+(N-1)+(N-2)+....+2=\frac{(N+2)\times(N-1)}{2}$
b1	$(N-1)+(N-2)+....+2+1=\frac{(N)\times(N-1)}{2}$
b11	$(N-1)+(N-2)+....+2+1=\frac{(N)\times(N-1)}{2}$
b12	$(N-1)+(N-2)+....+2+1=\frac{(N)\times(N-1)}{2}$
b13	$(N-1)+(N-2)+....+2+1=\frac{(N)\times(N-1)}{2}$
b2	$(N-1)+(N-2)+....+2+1=\frac{(N)\times(N-1)}{2}$
c	N-1
total:	$3N^2+N-2$

الجدول(6-12) زمن التنفيذ لخطوات خوارزمية الترتيب بالتبديل.

6-4-3- خوارزمية الترتيب بالتبديل 6

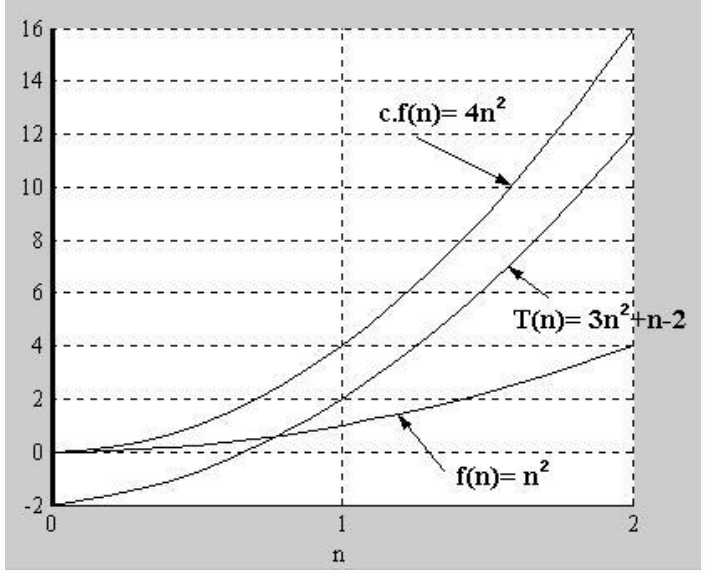
وعليه نستطيع التعبير عن زمن التنفيذ باستخدام تدوين BON كما يلي:

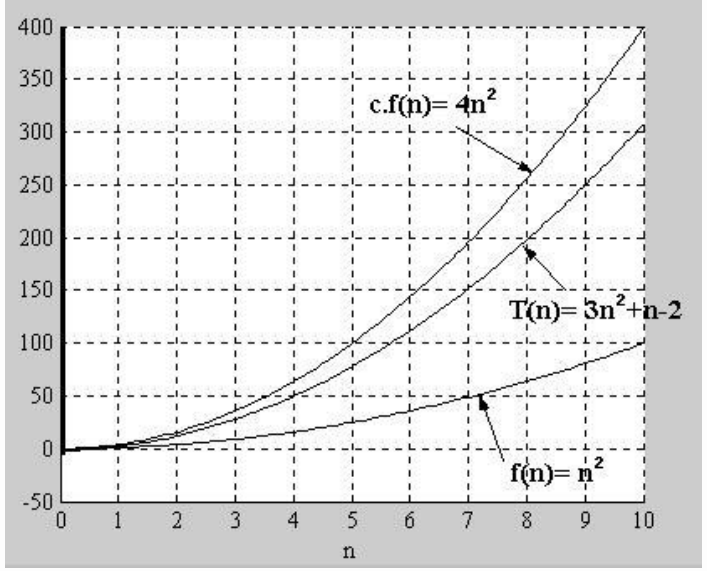
$$T(n) = 3N^2 + N - 2 \Rightarrow T(n) = O(n^2)$$

حيث أن :

$$3N^2 + N - 2 \le 4N^2 \; for \quad all \quad N \ge 0$$

أي أن $f(n) = n^2$, $C = 4$, $n_0 = 0$ ويبين الشكل (7-6) الخط البياني لكل من التوابع f(n),T(n),c.f(n):





الشكل (7-6)يبين أن 4.f(n) >= T(n) عندما n>= 0

والبرنامج(6-8) التالي يحقق هذه الخوارزمية حيث التابعان input وoutput هما كما في البرنامج السابق تماماً :

SWAP SORT ALGORITHM 7

6-4-3- خوارزمية الترتيب بالتبديل 7

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[]){......}
void output(int a[],int n){ ......}
void main(void)
{    const int n=8;
    //int A[n];input(A,n,”A”);
    int A[n]={ 25,20,16,12,7,5,2,1};
        cout<<"Before sorting :    ";
    output(A,n);
    cout<<endl;
```

SWAP SORT ALGORITHM 8

6-4-3- خوارزمية الترتيب بالتبديل 8

```
int first,second,temp;
    first=0;
     while(first<n-1)
    { second = first +1;
       while(second<n)
        { //cout<<"first= "<<first<<" second= "<<second<<": ";
         //output(A,n);
            if( A[first] > A[second] )
            {temp = A[first]; A[first]  = A[second] ;
                A[second] = temp ;
            } second++;
        }//cout<<endl<<"end of scan :"<<first+1<<" ";
        //output(A,n);cin.get();
         first++; }
```

SWAP SORT ALGORITHM 9

6-4-3- خوارزمية الترتيب بالتبديل 9

```
cout<<endl;
    cout<<"After  sorting :    ";
    output(A,n);
    cout<<endl;
system("pause");
}
```

ولجعل البرنامج يطبع النسق بعد كل عملية مقارنة أزل رمز التعليق // من أسطر التعليقات الواردة ضمن حلقتي while ثم أعد تنفيذه.

كما يمكن جعل البرنامج يدخل المعطيات من المستخدم بدلا من استخدام القيم الثابتة للعناصر بحذف رمز التعليق من السطر: //int A[n];input(A,n,"A");

ووضع رمز التعليق أمام السطر التالي له ,وسيقوم البرنامج عندها بطباعة النسق عددا من المرات في كل مرور وينتظر منك مفتاح الإدخال Enter للانتقال إلى المرحلة التالية.

ولفهم أعمق يطلب من الطالب تعديل البرنامج ليطبع النسق في حال كان النسق شبة مرتب وعدم هدر الزمن في المتابعة غير الضرورية أو العودة للكتاب لمعرفة النص البرمجي (6-6) .

SELECTION SORT ALGORITHM 1

6-4-4- خوارزمية الترتيب بالانتقائي 1

6-4-4- خوارزمية الترتيب بالاختيار(خوارزمية الترتيب الانتقائي) : SELECTION SORT ALGORITHM

يقوم مبدأ هذه الخوارزمية على عدة مسوحات للنسق (N-1) مرة,حيث يتم في المسح الأول المرورعلى العنصرجميعها لمعرفة العنصرالأصغربينها (عملية بحث خطي عن أصغرالعناصربدءاً من A[1] و انتهاء بـ A[N]) ثم يتم مبادلته بالعنصرالأول , وفي المسح الثاني يتم البحث عن أصغرالعناصربدءاً من A[2] و انتهاءً بـ A[n] وتتم مبادلته بالعنصرالثاني وهكذا دواليك, الخوارزمية(6-8) على الشكل التالي:

SELECTION SORT ALGORITHM (* Algorithm to sort the list A[1],....A[n] in ascending order. *)

1. set start equal to 1
2. while (start less or equal to n-1) do the following:
    - a. set pos equal to start+1.
    - b. set MINPOS equal to start;
    - c. while pos less or equal to n do the following:
        - c1. if A[pos] < A[MINPOS] then do the following:
            - (* store pos in MINPOS *)
            - c11. set MINPOS equal to pos
        - c2.set pos equal to pos +1
    - (* swap A[start] ←→A[MINPOS]*)
    - d. set temp equal to A[start] e. set A[start] equal to A[MINPOS] f. set A[MINPOS] equal to temp g. set start equal to start +1

SELECTION SORT ALGORITHM 2

6-4-4- خوارزمية الترتيب بالانتقائي 2

بالتدقيق في هذه الخوارزمية نجدها لا تخلف كثيرا عن خوارزمية الترتيب بالتبديل ولو حسبنا درجة تعقيدها لوجدناها من المرتبة N^2 أي أن:

$$T(N)=O(N^2)$$

- تقوم بالعدد ذاته من عمليات المقارنة والمسح وتختلف في عدد عمليات المبادلة فهي تقوم في كل مسح بمبادلة واحدة فقط.
- تقوم خوارزمية الترتيب بالتبديل بمبادلات عديدة ليست بالضرورة مبادلات ذات فائدة (واحدة منها فقط هي المفيدة) .
- هكذا تكون خوارزمية الاختيار قد وفرت الزمن اللازم للمبادلات غير الضرورية واستعاضت عنها بحفظ موقع العنصر الأصغر مما يجعل هذه الخوارزمية أكثر كفاءة من سابقتها عند العدد ذاته من العناصر ويظهر هذا الفارق خصوصاً عند عدد كبير نسبيا من العناصر.
- سيظهر هذا الفارق بشكل أوضح عندما تكون العناصر المخزنة في النسق عبارة عن سجلات ذات حجم بيانات كبير نسبيا تحوي على حقل مفتاح Key Field يتم الترتيب وفقه وعندها ستشكل عمليات المبادلة هدراً كبيرا للزمن مقارنة بالزمن اللازم لعمليات المقارنة والذي لا يشكل فعليا إلا جزءاً من الزمن اللازم لعملية نسخ بيانات واحد خاصة إذا ما تم اختيار الحقل المفتاح بحيث يكون معطيات رقمية صحيحة (وليس قيماً حقيقية أو "سلاسل محارف" (نصوص, كاسم الطالب أو كنيته أو مكان إقامته) ففي معظم الآلات تستغرق عملية مقارنة عددين حقيقين عدة أضعاف الزمن اللازم لمقارنة عددين صحيحين ومقارنة اسمين يتألف كل منهما من 10 أحرف عشرة أضعاف الزمن اللازم لمقارنة عددين حقيقين.

6-4-4- خوارزمية الترتيب بالانتقائي 3

ولتبيان هذا الفارق نفترض الحالة التي تكون العناصر مؤلفة من سجل يحوي معلومات عن الطالب إضافة إلى الحقل المفتاح كما في الشكل (6-8) حيث الحقل المفتاح هو رقم الطالب مثلا وتتألف المعطيات من الحقول التالية (أيضا كمثال لا حصر):

الرقم : رقم صحيح

الاسم : بطول أعظمي 20 حرف مثلاً.

الكنية : بطول أعظمي 20 حرف مثلاً.

اسم الأب : بطول أعظمي 20 حرف مثلاً.

اسم الأم : بطول أعظمي 20 حرف مثلاً.

مكان الإقامة : بطول أعظمي 150 حرف مثلاً.

مكان الميلاد : بطول أعظمي 20 حرف مثلاً.

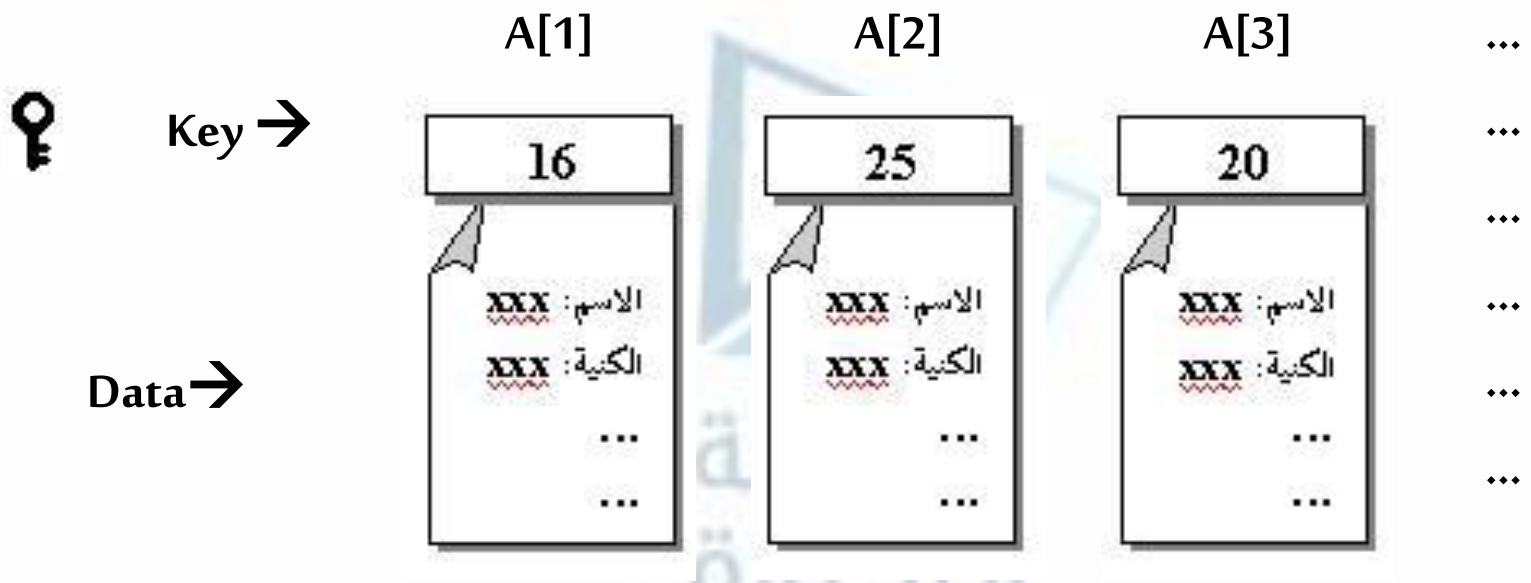
تاريخ الميلاد : السنة :عدد صحيح

: الشهر:عدد صحيح

: اليوم :عدد صحيح

السنة الدراسية : رقم صحيح

6-4-4- خوارزمية الترتيب بالانتقائي 4



الشكل (6-8) حالة عناصر النسق على شكل سجلات ذات حقل مفتاح

فإذا افترضنا أن العدد الصحيح يحتل 2Byte والحرف 1Byte فإن السجل الواحد سيحتل ما مقداره (5x2+5x20x1+1x150x1)=260byte والذي سيحتاج زمنا للنسخ في الذاكرة من موقع إلى آخر قدره 130t حيث t هو الزمن اللازم لنسخ عدد صحيح واحد ونعلم أن عملية مبادلة واحدة تحتاج لثلاث عمليات نسخ (أو نسب) فإن الزمن يصبح 390t أي أنه يساوي 130

6-4-4- خوارزمية الترتيب بالانتقائي 5

ضعفا من الزمن اللازم لمبادلة عددين صحيحين, ويستطيع القارئ هنا أن يتنبأ بالوفر الزمني الذي ستقدمه خوارزمية الترتيب بالاختيار عن خوارزمية الترتيب بالتبديل عند ترتيب نسق من العناصر يحوي مثلاً 100 سجل.حيث ستقوم خوارزمية التبديل بـ 4950=2/(N-1).(N) عملية مقارنة مع العدد ذاته من عمليات المبادلة في أسوأ الاحتمالات, أما خوارزمية الاختيار فستقوم بـ 4950 عملية مقارنة و 100 عملية مبادلة في أسوأ الاحتمالات, أي أن هناك توفيراً لـ 4850 عملية مبادلة كل منها توفر 130 ضعفا من زمن مبادلة عددين صحيحين.

كما تجب الملاحظة أن الحالة الأسوأ للنسق في حالة الترتيب بالاختيار ليست كالحالة الأسوأ لخوارزمية الترتيب بالتبديل فالحالة الأسوأ لخوارزمية التبديل هي عندما يكون النسق مرتبا بشكل عكسي في حين أن الحالة الأسوأ لخوارزمية الاختيار هي عندما تكون العناصر متوضعة في ترتيب خاص بحيث إذا اتجهنا من الموقع الأخير إلى الموقع الأول يرد العنصر الأصغر في آخر النسق ثم يرد العنصر الأكبر ثم ثاني أصغر العناصر وبعده ثاني أكبر العناصر وهكذا دواليك كما لوكان التالي:

12	7	16	5	20	2	25	1
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

ويقدم البرنامج(6-10) التالي تطبيقاً لخوارزمية الترتيب بالاختيار مع الحالة الأسوأ:

SELECTION SORT ALGORITHM 6

6-4-4- خوارزمية الترتيب بالانتقائي 6

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[])
{     for (int i=0;i<n;i++)
    { cout<<"input "<<name<<"["<<i+1<<"] : ";cin>>a[i];    }
}

void output(int a[],int n)
{     cout<<"[ " ;
    for (int i=0;i<n;i++)cout<<setw(4)<<a[i]; cout<<" ]"<<endl;
}
void main(void)
```

SELECTION SORT ALGORITHM 7

4-4-6- خوارزمية الترتيب بالانتقائي 7

```
void main(void)
{     const int n=8;
    int A[n]={ 12,7,16,5,20,2,25,1};
    cout<<"Before sorting :     ";   output(A,n);    cout<<endl;

         int start,pos,MINPOS,temp ;

    start=0;

    while(start<n-1)  {   pos = start +1;        MINPOS= start ;
        while(pos<n)   { if( A[pos] < A[MINPOS] ) MINPOS = pos ;
         cout<<"start="<<start<<"MINPOS="<<MINPOS<<"pos="<<pos<<":";
            output(A,n); pos++;
        }
```

SELECTION SORT ALGORITHM 8

6-4-4- خوارزمية الترتيب بالانتقائي 8

```
      temp = A[start] ; A[start]  = A[MINPOS]; A[MINPOS] = temp;
         cout<<endl<<"end of scan : "<<start+1<<"     ";
         output(A,n);
         cin.get();
         start++;
    }
      cout<<“ \n After  sorting :    ";
    output(A,n);    cout<<endl;
system("pause");
}
```

INSERTION SORT ALGORITHM

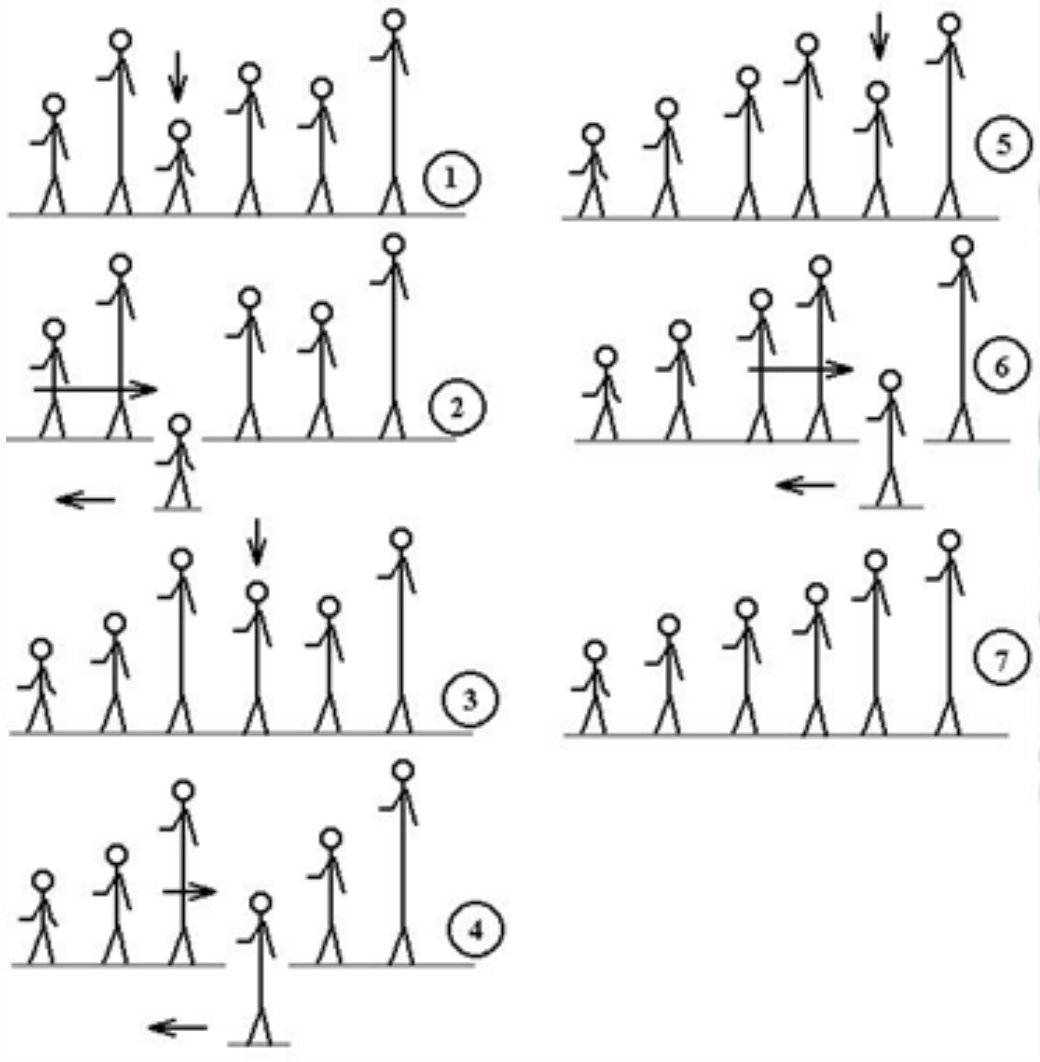
6-4-5- خوارزمية الترتيب بالحشر1

تعتبر خوارزمية الترتيب بالحشر من خوارزميات الترتيب الشائعة الاستخدام فهي مثلا الطريقة التي نقوم بها بترتيب أوراق اللعب بشكل فطري وهي تعتمد على الوصول إلى العناصر الثاني فالثالث فالرابع ووضعه في مكانه الصحيح (حشره في المكان الصحيح) وذلك بإزاحة العناصر اللازم إزاحتها(وهي العناصر السابقة له والأكبر منه عند الترتيب التصاعدي) لإتاحة مكان لوضع ذلك العنصر فيه كما نستطيع تبيان هذه الخوارزمية في عملية ترتيب رتل من الأفراد وفق أطوالهم المتزايدة كما يظهر الشكل (6-9) التمثيلي التالي:

ونجد في الرسم التالي أنه لا حاجة بنا إلى تغيير مكان العنصر الثاني فهو أكبر من جميع العناصر السابقة له (لا نقوم بأية إزاحة) ثم نصل إلى العنصر الثالث فنجد أنه بحاجة للحشر في موقع سابق فقد تم حشره في المكان الأول عن طريق إزاحة جميع العناصر التي هي أطول منه خطوة واحدة إلى الخلف فأتحنا له مكاناً في بداية النسق كما في الرسم (2) ثم في الرسم (3) تم الوصول إلى العنصر التالي الذي يخالف الترتيب وهو العنصر الرابع ثم تمت إزاحة العناصر السابقة له والأطول منه (هنا هو العنصر الثالث فقط) لإتاحة مكان له في الموقع الثالث وحشرنا هذا العنصر في مكانه الصحيح الرسم (4) وفي الرسم (5) وصلنا إلى عنصر جديد يخالف الترتيب وهو العنصر الخامس ولوضعه (لحشره) في مكانه المناسب وهو الموقع الثالث نزيح العناصر الثالث والرابع خطوة واحدة للخلف ونضعه في الموقع الثالث.

مما يميز هذه الخوارزمية بساطتها وفعاليتها في المتوسط (وليس في الحالة الأسوأ) وبخاصة عندما يراد ترتيب أنساق مرتبة سابقا بعد إضافة عنصر جديد لها حيث يمكن إضافة هذا العنصر إلى نهاية النسق وإعادة الترتيب وعندها نحتاج عملية إزاحة لـ N عنصراً على الأكثر.

INSERTION SORT ALGORITHM 2



6-4-5- خوارزمية الترتيب بالحشر2

الشكل (6-9)- رسم تمثيلي لخوارزمية الترتيب بالحشر

والآن على فرض لدينا النسق التالي:

25	20	16	12	7	5	2	1
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

نكتب خوارزمية (6-9) الترتيب بالحشر كما يأتي:

INSERTION SORT ALGORITHM 3

6-4-5- خوارزمية الترتيب بالحشر 3

INSERTION SORT ALGORITHM

(* Algorithm to sort the list A[1],....A[n] in ascending order. *)

1. set pos equal to 2
2. while (pos less or equal to n) do the following:
    a. set temp equal to A[pos]
    b. set i equal to pos
    c. while (A[i-1] greater than temp) do the following:
        c1. set A[i] equal to A[i-1]
        c2. set i equal to i-1
    d. set A[i] equal to temp
    e. set pos equal to pos+1

ولحساب التابع الزمني وفق BON لخوارزمية الترتيب بالحشر نضع الجدول (6-13) الذي يبين عدد مرات التنفيذ لعبارات الخوارزمية محسوبا على الحالة الأسوأ وهي حالة نسق مرتب ترتيبا عكسياً:

INSERTION SORT ALGORITHM 4

statement	# of times executed
1	1
2	N
a	N-1
b	N-1
c	$2+3+....+(N-1)+(N)=\frac{(N-1)\times(N+2)}{2}$
c1	$1+2+....+(N-2)+(N-1)=\frac{(N-1)\times(N)}{2}$
c2	$1+2+....+(N-2)+(N-1)=\frac{(N-1)\times(N)}{2}$
d	N-1
e	N-1
total:	$\frac{3}{2}N^2+\frac{9}{2}N-4$

6-4-5- خوارزمية الترتيب بالحشر 4

الجدول (6-13) زمن تنفيذ خطوات الترتيب بالحشر

وعليه نستطيع التعبيرعن زمن التنفيذ باستخدام تدوين BON كما يلي:

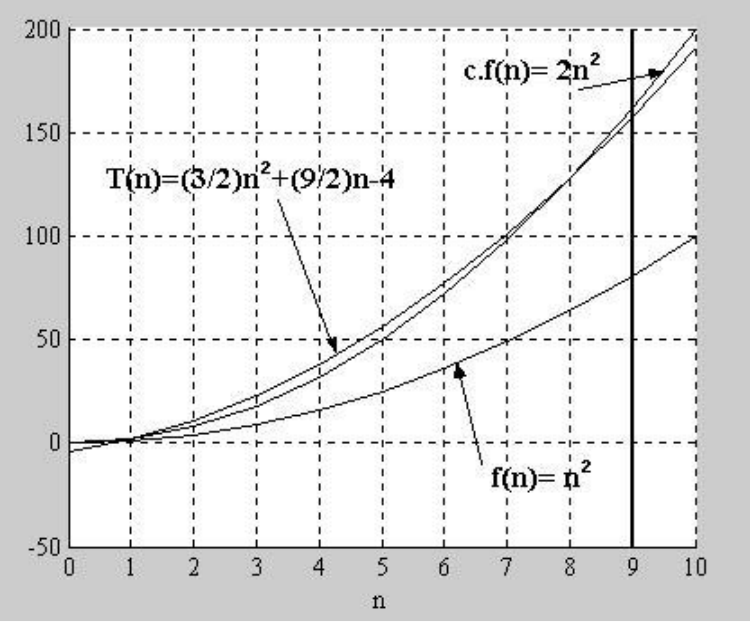
$$T(n)=\frac{3}{2}N^2+\frac{9}{2}N-4 \Rightarrow T(n)=O(n^2)$$

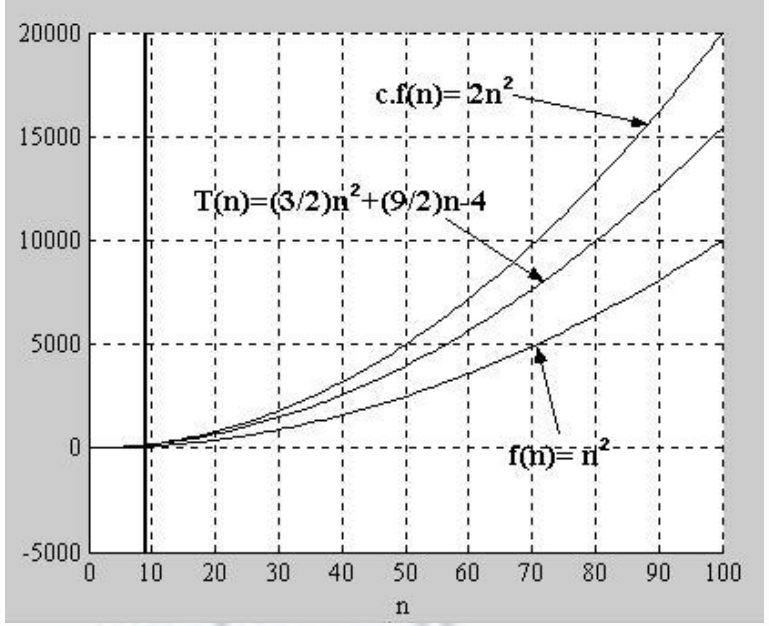
حيث أن :

$$\frac{3}{2}N^2+\frac{9}{2}N-4 \le 2N^2 \quad for \quad all \quad N \ge 9$$

أي أن $f(n)=n^2$, $C=2$, $n_0=9$ ويبين الشكل (6-10) الخط البياني لكل من التوابع f(n),T(n),c.f(n):

6-4-5- خوارزمية الترتيب بالحشر 5





الشكل (6-10) يبين أن $2.f(n) >= T(n)$ عندما $n>= 9$

ويجب أن نشير هنا إلى أن الخوارزمية المكتوبة سابقاً لم تعالج حالة خاصة حيث عند معالجة العنصر A[2] ستتم المقارنة مع العنصر A[1] وعندها سنكون أمام احتمالين:

6-4-5- خوارزمية الترتيب بالحشر 6

- لأول : A[2] ≤ A[1] وعندها لا تتم أية إزاحة وتنتهي الحلقة الداخلية وتم بعدها الانتقال لمعالجة العنصر A[3] وما بعده دون أية مشاكل.
- الثاني : A[2] > A[1] وعندها سيتم الدخول إلى الحلقة الداخلية وستبدأ عملية إزاحة من غير المعلوم متى تنتهي وذلك لأنه سيتم تجاوز حدود النسق (سيتم الوصول إلى العناصر A[0],A[-1],...) وتجاوز حدود النسق سيؤدي في بعض لغات البرمجة التي تقوم بفحص تجاوز الحدود إلى رسالة خطأ وفي البعض الآخر كلغة C++ لا يتم مثل هذا الفحص مما يجعل العمل يستمر خارج حدود النسق وإدخال عناصر أخرى من الذاكرة المجاورة للنسق إليه وإشراكها في عملية الترتيب وهذا ولا شك خطأ كبير سيؤدي على الأغلب إلى تعطل البرنامج في ما بعد (وحتى إن لم يتعطل فعلى الأقل لن يؤدي العمل المطلوب).
- لذلك لا بد من معالجة هذه المشكلة قبل وضع برنامج ينفذ هذه الخوارزمية وهذا الحل يكون بإضافة شرط لعدم تجاوز حدود النسق كما في الخوارزمية (6-10) التالية:

INSERTION SORT ALGORITHM 7

6-4-5- خوارزمية الترتيب بالحشر 7

1. set pos equal to 2
2. while (pos less or equal to n) do the following:
    a. set temp equal to A[pos]
    b. set i equal to pos
    c. while (A[i-1] greater than temp)
        AND (i greater than 1) do the following:
        c1. set A[i] equal to A[i-1]
        c2. set i equal to i-1
    d. set A[i] equal to temp
    e. set pos equal to pos+1

- هناك طريقة أخرى تهدف لتجنب إضافة شرط (بغية اختصار الزمن اللازم لهذا الاختبار وبالتالي زيادة فعالية الخوارزمية) وتقوم على إضافة عنصر قبل بداية النسق(في A[0]) يعمل كحاجز صد أو كحارس (sentinel) ويوضع به قيمة أصغر من كل القيم فيعمل كحارس يمنع الحلقة الداخلية من الوصول خارج حدود النسق.

6-4-5- خوارزمية الترتيب بالحشر 8

والبرنامجان التاليان(6-11) و(6-12) يحققان خوارزمية الترتيب بالحشر أولهما يستخدم شرطاً إضافيا لضمان عدم تجاوز حدود النسق والآخر يستخدم عنصر صد:

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[]){……}
void output(int a[],int n) { ……..}
void main(void)
{    const int n=8;
    int A[n]={ 25,20,16,12,7,5,2,1};
    cout<<"Before sorting :    ";
    output(A,n);    cout<<endl;
    int pos,i,temp;
```

INSERTION SORT ALGORITHM 9

6-4-5- خوارزمية الترتيب بالحشر 9

```
    pos = 1 ;
     while (pos < n)
     {    temp = A[pos] ;  i     = pos     ;
          while( (A[i-1]  > temp )&&( i > 0) )
          {    A[i] = A[i-1] ; i-- ; A[i] = temp    ;
          }
          pos  = pos + 1 ;
    }
      cout<<"\n After  sorting : ";
     output(A,n);
     cout<<endl;
  system("pause");
 }
```

6-4-6- خوارزمية الترتيب بالدمج 1

6-4-6-1- مفهوم الدمج:

- مفهوم الدمج Merging هو عملية تكوين لائحة مرتبة من العناصر انطلاقا من لائحتين منفصلتين كل منهما مرتبة وكثيرا ما نحتاج لعملية الدمج هذه لدمج بيانات مرتبة في ملفات منفصلة فمثلا عند دمج ملفات شركة مع شركة أخرى لتكوين شركة واحدة. والشكل (6-11) يبين مثالا بسيطا لعملية الدمج هذه حيث يمثل النسق C ناتج دمج النسقين المرتبين A,B:

3	7	16	20
A[1]	A[2]	A[3]	A[4]

4	6	11
B[1]	B[2]	B[3]

الشكل (6-11) – مفهوم الدمج

3	4	6	7	11	16	20
C[1]	C [2]	C [3]	C [4]	C [5]	C [6]	C [7]

MERGE SORT ALGORITHM 2

6-4-6- خوارزمية الترتيب بالدمج 2

ويبين النص التالي خوارزمية الدمج ذات الرقم (6-12) على فرض ما يلي:

- Na هو طول النسق A و Nb هو طول النسق B.
- Nc هو طول النسق C حيث Nc >= Na+Nb

لا يوجد عناصر مكررة في A ولا عناصر مكررة في B ولا عناصر مكررة في A و B معاً(بهدف التبسيط).

```
1. set Ia equal to 1.
2. set Ib equal to 1.
3. set Ic equal to 1.
4. while ( Ia <= Na ) and ( Ib <= Nb ) do the following:
      4a. if A[Ia] < B[Ib] then
            4a1. set C[Ic] equal to A[Ia]
            4a2. set Ia  equal to Ia+1
         else
```

MERGE SORT ALGORITHM 3

6-4-6- خوارزمية الترتيب بالدمج 3

4a3. set C[Ic] equal to B[Ib]

4a4. set Ib equal to Ib+1

4b. set Ic equal to Ic+1

5. if (Ia < Ib) then:

5a. while (Ia <= Na) do the following:

5a1. set C[Ic] equal to A[Ia]

5a2. set Ia equal to Ia+1

5a3. set Ic equal to Ic+1

else

5b. while (Ib <= Nb) do the following:

5b1. set C[Ic] equal to B[Ib]

5b2. set Ib equal to Ib+1

5b3. set Ic equal to Ic+1

6-4-6- خوارزمية الترتيب بالدمج 4

- ويقدم البرنامج (6-13) التالي تطبيقا عمليا لهذه الخوارزمية بلغة C++

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[]);
void output(int a[],int n);
void merge(int A[],int B[],int C[],int Na,int Nb,int Nc);
void main(void)
{    const int n=5;                int A[n]={1 ,4 ,7 ,20,70};
    int B[n]={2 ,3 ,80,90,95};  int C[2*n]={0};
    output(A,n);output(B,n);cout<<endl;merge(A,B,C,n,n,2*n);
output(C,2*n);cout<<endl;system("pause");
}
```

6-4-6- خوارزمية الترتيب بالدمج 5

```
void merge(int A[],int B[],int C[] ,int Na,int Nb,int Nc)
{     int Ia=0,Ib=0,Ic=0;
      while ( ( Ia < Na ) && ( Ib < Nb ) )
      {     if( A[Ia] < B[Ib] )   { C[Ic] = A[Ia];Ia++; }
            else { C[Ic] = B[Ib];Ib++; }Ic++;     }
      if ( Ia < Ib ) {
         while ( Ia < Na )
         { C[Ic] = A[Ia];Ia++;Ic++; }  }
      else {
         while ( Ib < Nb )
         { C[Ic] = B[Ib];Ib++;Ic++; }
      }
 }
```

6-4-6-2- خوارزمية الدمج 1

وتعتمد خوارزمية الترتيب بالدمج على عملية الدمج السابقة بشكل أساسي إضافة إلى اعتمادها على مفهوم الاستدعاء العودي حيث يمكن ترتيب نسق إذا كان لدينا نصفان مرتبان كما يبين الشكل (6-12) التالي:

A	5	7	20	45	70	2	8	10	23	90

عندئذ يمكن دمجهما ضمن نسق مساعد B ثم إعادة الناتج بعد الدمج إلى A:

B	2	5	7	8	10	20	23	45	70	90

الشكل (6-12)يبين مفهوم الدمج.

ولكن لترتيب كل من نصفي النسق A أولا لابد من تقسيم كل منهما إلى نصفين بحيث يتم ترتيب كل منهما في B وإعادة الناتج المرتب إلى A ويستمر التقسيم إلى أن نصل في النهاية إلى نصفين كل منهما هو عنصر واحد أو عنصرين على الأكثر يتم ترتيبهما دمجيا في B ثم إعادة الناتج إلى A وتم مثل هذا التقسيم والترتيب باستخدام تابع عودي (Recursive Function) يستدعي نفسه أولا ليرتب نصفي النسق ثم يدمجهما ولترتيب كل نصف سيقوم أيضا باستدعاء نفسه من جديد لترتيب كل جزء على حدة ثم يقوم بالدمج ... وهكذا دواليك . والبرنامج (6-14) التالي ينفذ هذه الخوارزمية.

MERGE SORT 2

2-6-4-6- خوارزمية الدمج 2

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[])
{    for (int i=0;i<n;i++)
    {cout<<"input"<<name<<"["<<i+1<<"]:";cin>>a[i];}}
void output(int a[],int n)
{    cout<<"[ " ;
   for(int i=0;i<n;i++)cout<<setw(4)<<a[i];cout<<"]"<<endl;}
void mergeSort(int A[], int B[], int n);
void m_sort(int A[], int B[], int left, int right);
void merge(int A[],int B[],int left,int mid,int right);
```

MERGE SORT 3

6-4-6-2- خوارزمية الدمج 3

```
void main(void)
{     const int n=8;
     int A[n]={25,5,1,7,12,2,16,20};
     int B[n]={0};

     cout<<"Before sorting :     ";
 output(A,n); cout<<endl;

     mergeSort(A , B , n);

     cout<<"After  sorting :     ";
output(A,n); cout<<endl;
      system("pause");
}
```

MERGE SORT 4

6-4-6-2- خوارزمية الدمج 4

```
void mergeSort(int A[],int B[], int n)
{     m_sort(A, B, 0, n - 1);  }
void m_sort(int A[], int B[],int left,int right)
{     int mid;
      if (right > left)
      {     mid = (right + left) / 2;
            m_sort(A, B, left, mid);
            m_sort(A, B, mid+1, right);
            merge(A, B, left, mid+1, right); } }
void merge(int A[],int B[],int left,int mid,int right)
{     int i_b =0 ;          int i_left  = left ;
      int i_right = mid  ;  int n =  right - left + 1 ;
      int  left_end = mid - 1;   int right_end = right  ;
      while((i_left<=left_end)&&(i_right<=right_end))
```

MERGE SORT 5

2-6-4-6- خوارزمية الدمج 5

```
{     if (A[i_left] < A[i_right])
      {B[i_b] = A[i_left];i_left++;}
      else
      {B[i_b] = A[i_right];i_right++;}
      i_b++;
}

while (i_left <= left_end)
{ B[i_b] = A[i_left];i_left++;i_b++; }
while (i_right <= right_end)
{ B[i_b] = A[i_right];i_right++;i_b++; }
for (i_b =0; i_b < n; i_b++)
     A[left+i_b] = B[i_b];
}
```

6-4-6-2- خوارزمية الدمج 6

حيث التابع m_sort هو التابع العودي الذي يقوم بالتقسيم واستدعاء نفسه لترتيب كل من القسمين على حدة ثم يدمجهما عن طريق استدعاء التابع merge الذي ينفذ عملية الدمج بمساعدة النسق B.

وتعتبر خوارزمية الترتيب بالدمج من الخوارزميات الفعالة لترتيب أنساق ذات عدد عناصر كبير نسبياً حيث يعطي حساب تعقيد الخوارزمية تابعاً من الشكل:

$$T(n)=O(\ n \log(n)\ )$$

ولكن يؤخذ على هذه الخوارزمية حاجتها إلى نسق إضافي (مساوي في الطول للنسق الأصلي) كما أنها خوارزمية عودية الأمر الذي لا يمكن تحقيقه مع بعض لغات البرمجة التي لا تسمح بالاستدعاءات العودية.

QUICK SORT ALGORITHM 1

6-4-7- خوارزمية الترتيب السريع 1

ندرس الآن إحدى أهم خوارزميات الترتيب وربما الخوارزمية الأكثر استخداما على نطاق واسع.

لقد تم تطوير الخوارزمية الأصل في عام 1960 من قبل C. A. R. Hoare ومذ ذاك تمت دراستها وإدخال تطويرات عديدة عليها من قبل العديد من الباحثين , وقد شاع استخدامها بكثرة وذلك لكونها سهلة التحقيق العملي ولأنها مناسبة لأغراض متنوعة وهي تحتاج إلى موارد أقل مما تحتاجه خوارزميات ترتيب أخرى ,ومن حسنات هذه الخوارزمية أنها تنفذ في المكان (in-place Algorithm) فهي لا تحتاج إلى ذاكرة إضافية للعمل (رغم حاجتها إلى حجم مكدس صغير نسبيا) وتملك درجة تعقيد في المتوسط O(n log(n)) لترتيب N عنصراً وتنفذ حلقة داخلية قصيرة للغاية.

بيد أن أهم سيئات هذه الخوارزمية كونها خوارزمية عودية (recursive algorithm) حيث يكون التحقيق العملي لها دون استخدام الاستدعاءات العودية معقدا جداً وذلك عند الحاجة لتنفيذ الترتيب على آلات أو باستخدام لغات برمجة لا تدعم الاستدعاء العودي كما أنها في الحالة الأسوأ تملك درجة تعقيد $O(N^2)$.

تم تهذيب الخوارزمية وتشذيبها حتى أضحت الخيار الأول للعديد من التطبيقات التي يطلب فيها الترتيب على عدد كبير من العناصر.

6-4-7- خوارزمية الترتيب السريع 2

تسلسل عمل الخوارزمية :

تعتمد خوارزمية الترتيب السريع على طريقة "فرق-تسد" divide-and-conquer في الترتيب حيث تعمل هذه الخوارزمية على تقسيم نسق العناصر التي يراد ترتيبها إلى قسمين ثم ترتيب كل قسم على حدة وموقع عنصر التقسيم يعتمد على عناصر النسق ذاتها وبذلك يكون للخوارزمية البنية العودية التالية:

لترتيب العناصر في النسق بدءا من L(left) وحتى R(right) قم بالتالي:

- إذا كان L<R عندئذ نفذ مايلي:
  - قسم العناصر من L وحتى R واحصل على موقع التقسيم i.
  - رتب العناصر من L وحتى i-1 بالطريقة ذاتها.
  - رتب العناصر من i+1 وحتى R بالطريقة ذاتها.
- وإلا : لاشيء

QUICK SORT ALGORITHM 3

6-4-7- خوارزمية الترتيب السريع 3

يتم في البداية استدعاء الإجراء العودي مع قيم أولية لـ L=1 و R=N حيث N هو عدد العناصر الكلي في النسق.

وتتم عملية التقسيم واختيار العنصر i بناء على ما يلي:

1. بحيث يكون العنصر A[i] في موقعه النهائي الصحيح.
2. جميع العناصر من A[L] وحتى A[i-1] أصغر أو تساوي A[i].
3. جميع العناصر من A[i+1] وحتى A[R] أكبر أو تساوي A[i].

ويمكن تحقيق مثل هذا التقسيم باتباع التالي:

1. اختر أحد العناصر ليكون العنصر الذي سيصبح في موقعه النهائي (عنصر التقسيم) وليكن العنصر A[R].
2. امسح النسق من اليسار وتوقف عند أول عنصر أكبر من A[R] أو يساويه.
3. امسح النسق من اليمين وتوقف عند أول عنصر أصغر من A[R] أو يساويه.
4. بادل العنصرين الذين توقفت عندهما وتقدم عنصراً واحداً من كل جهة.
5. كرر الخطوات 4,3,2 حتى يتقاطع مؤشرا المسح.
6. بادل العنصر A[R] بالعنصر الموجود في أقصى يسار الجزء الأيمن من العناصر ويكون موقع التقسيم هو i وهو أقصى يسار الجزء الأيمن ويكون أقصى يسار الجزء الأيمن يتلو بـ 1 مؤشر المسح من جهة اليمين.

6-4-7- خوارزمية الترتيب السريع 4

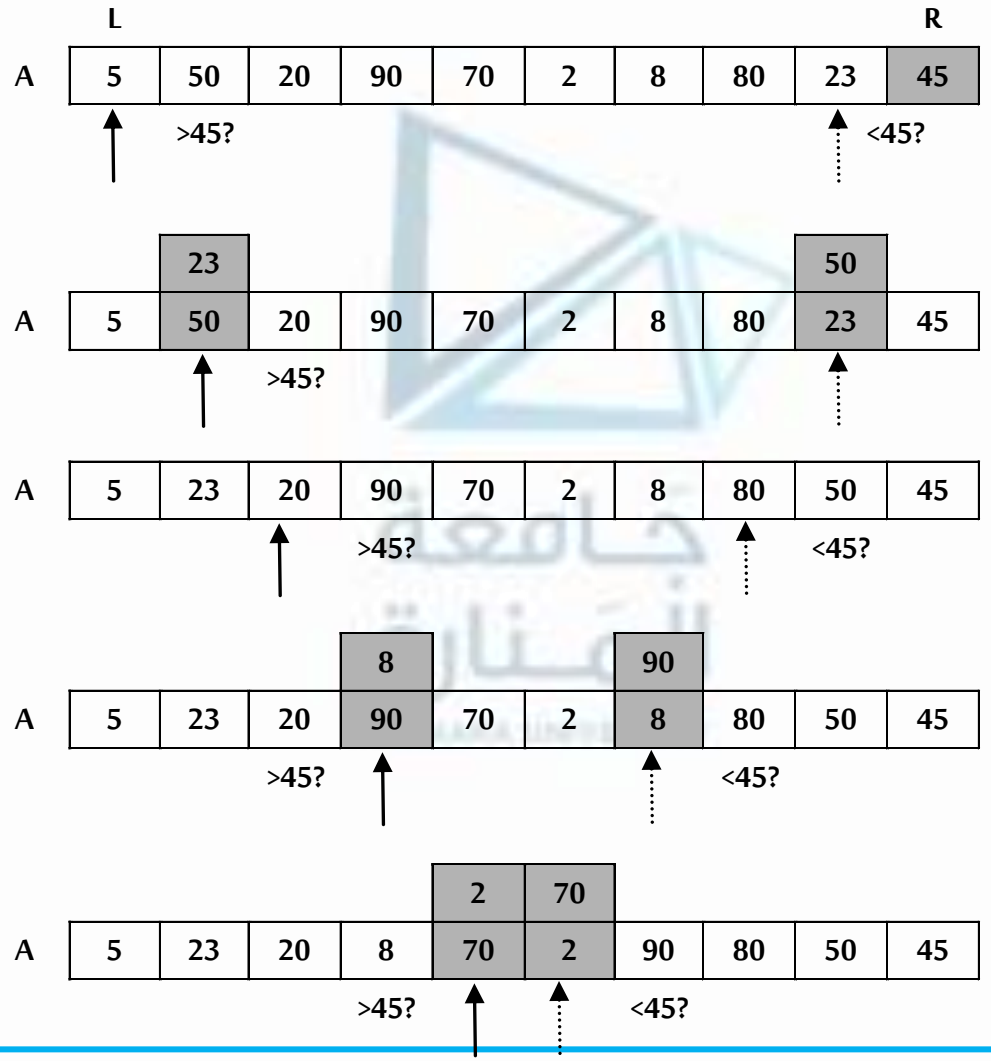
وهكذا نكتب الخوارزمية(6-12) كاملة باستخدام pseudo code:

```
procedure quicksort( L, R : integer);
begin
        i ← partition(L, R) ;
    if L < i-1 then
        quicksort (L, i - 1)  ;
    if i+1 < R then
        quicksort(i+l, R)    ;
end ;
```

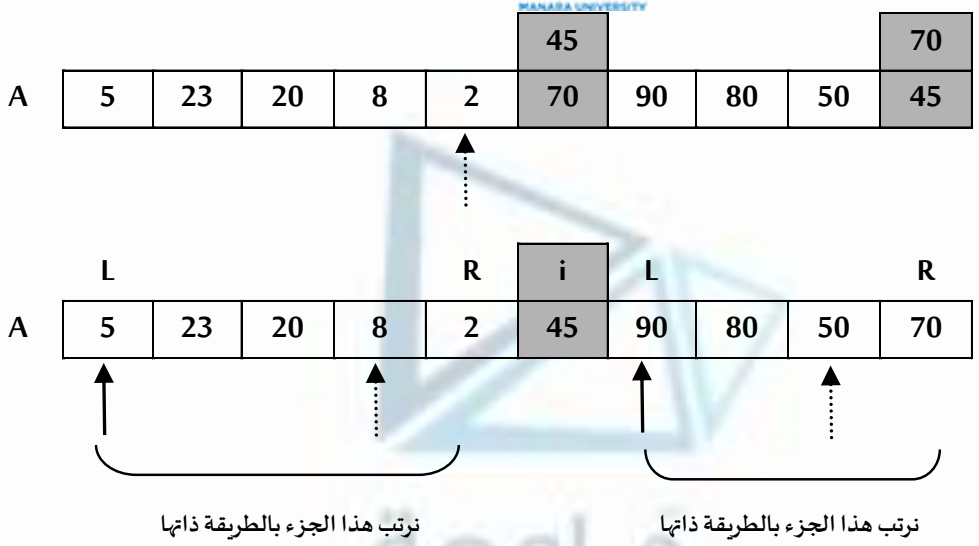
حيث partition هو تابع يقوم بعمليات المبادلة اللازمة ويعيد موقع عنصر التقسيم ولكن لأغراض الأداء الفعال يفضل عدم استدعاء تابع خاص للقيام بعملية التقسيم بل وضع شفرة التقسيم مباشرة ضمن الإجراء quicksort لتصبح الخوارزمية(6-14) كالتالي:

QUICK SORT ALGORITHM 5

6-4-7- خوارزمية الترتيب السريع 5



6-4-7- خوارزمية الترتيب السريع 6



الشكل (6 -13) يبين عملية التقسيم في خوارزمية الترتيب السريع

والنص البرمجي (6-15) يبين برنامجاً يستخدم خوارزمية الترتيب السريع لترتيب عناصر النسق كما يطبع النسق على الشاشة قبل الترتيب وبعده باستخدام التابع output كما في كل الأمثلة السابقة:

QUICK SORT ALGORITHM 7

7-4-6- خوارزمية الترتيب السريع 7

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[])
{    for (int i=0;i<n;i++)
   {cout<<"input"<<name<<"["<<i+1<<"]:";cin>>a[i]; }
}
void output(int a[],int n)
{    cout<<"[ " ;
     for (int i=0;i<n;i++)cout<<setw(4)<<a[i];
     cout<<" ]"<<endl;
}
```

QUICK SORT ALGORITHM 7

7-4-6- خوارزمية الترتيب السريع 7

```
void quicksort(int L,int  R ,int A[]);

void main(void)
{  const int n=10;
   int A[n]={ 5, 50, 20, 90, 70, 2, 8, 80, 23, 45};
    cout<<"Before sorting :"<<endl;
    output(A,n);
    cout<<endl;
    quicksort(0,n-1,A);    cout<<endl;
    cout<<"After  sorting :"<<endl;
    output(A,n); cout<<endl; system("pause");}
```

QUICK SORT ALGORITHM 7

7-4-6- خوارزمية الترتيب السريع 7

```
void quicksort(int L,int  R ,int A[])
{  int v,t,Rp,Lp; v= A[R]; Lp = L   ;  Rp = R-1 ;
    while ( Rp > Lp )
    { while ( (A[Lp] < v ) && ( Lp < Rp ) )  Lp++ ;
      while ( (A[Rp] > v ) && ( Lp < Rp ) )  Rp-- ;
        if ( Rp != Lp)
        {   // swap A[Lp] <==> A[Rp]
            t= A[Lp];  A[Lp]=A[Rp];  A[Rp]=t;  } }
    if ( A[R] < A[Lp] )
        {t = A[R] ;  A[R] = A[Lp];  A[Lp]= t; }
    else Lp = R;
    if (L<Lp-1) quicksort(L   , Lp-1 , A);
    if (Lp+1<R) quicksort(Lp+1, R    , A);  }
```

BUCKET SORTALGORITHM 1

6-4-8- خوارزمية الـ BUCKET SORT 1

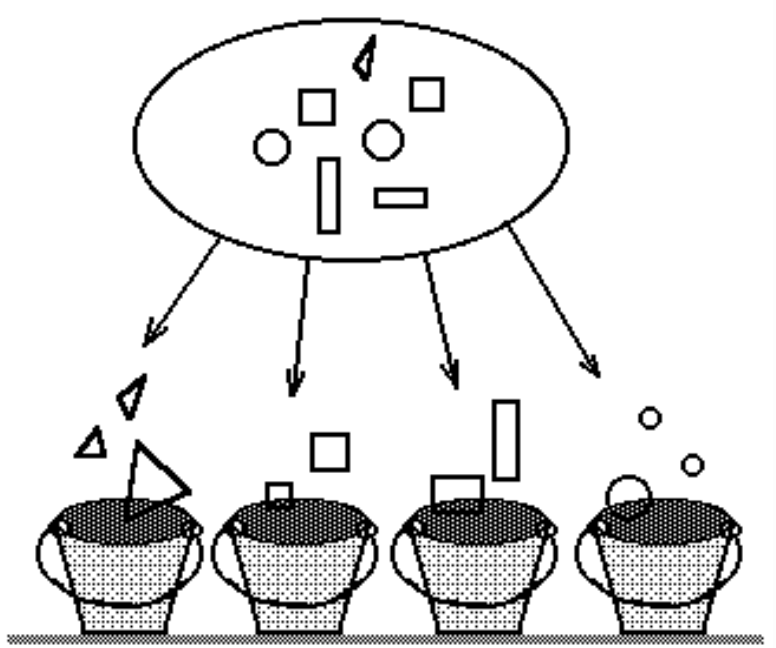
نقدم فيما يلي خوارزمية مميزة بما تقدمه من سرعة في عملية ترتيب عناصر النسق وفكرة هذه الخوارزمية تعتمد على فكرة بسيطة يمكن أن نشبهها بالمثال التوضيحي التالي:

ليكن لدينا مجموعة من القطع المعدنية بأشكال مختلفة (مثلثات– مستطيلات – مربعات – دوائر ...) موضوعة معا (مختلطة) ونرغب بفرزهذه العناصر حسب الشكل كلاً على حدة من الطرق الممكنة التالي:

نأخذ وعاءً (جردلاً) - (سطل-دلو-Bucket) لأجل كل من الأشكال المختلفة الموجودة في المجموعة فإذا كان لدينا أربع أشكال نحتاج إلى أربعة جرادل, ثم نبدأ بالقطع واحدة فأخرى فإذا كانت على شكل مثلث نضعها في الجردل الأول وإن كانت مربعة الشكل نضعها في الثاني وإن كانت مستطيلة الشكل نضعها في الثالث والدائرية نضعها في الرابع وهكذا دواليك مع الأشكال الأخرى ثم نكرر العملية مع كل القطع فنحصل بالتالي على الأشكال كلاً لوحده في أحد الأوعية كما في الشكل التمثيلي(6-14):

BUCKET SORTALGORITHM 2

6-4-8- خوارزمية الـ BUCKET SORT2



الشكل (6-14)- شكل تمثيلي لعملية فرز القطع المعدنية

أما في حال ترتيب الأعداد فإن العمل يتم بفرز الأعداد حسب تشابهها فإذا كانت كلها مؤلفة من خانة عشرية واحدة يتم فرز كل رقم على حدة من 0 وحتى 9

عندما تكون الأعداد مؤلفة من أكثر من خانة (وسنفترض بداية أنها جميعها أكبر أو تساوي الصفر) يتم الفرز وفق الخانات بالتتابع بدءاً بالخانة الأقل أهمية وهي خانة الآحاد ثم العشرات و انتهاء بالخانة الأخيرة (الأكثر أهمية) وهي الخانة الأخيرة في أكبر الأعداد (نعتبر الخانات غير الموجودة تحوي الرقم 0), ولتنفيذ عملية الفرز نحتاج إلى مصفوفة ثنائية البعد عدد أسطرها 10 مرقمة من 0 إلى 9 وعدد أعمدتها يساوي n (عدد العناصر المراد ترتيبها) مرقمة من 0 وحتى n-1 وللقيام بالترتيب نقوم بالخوارزمية (6-13) التاليه:

6-4-8- خوارزمية الـ BUCKET SORT:

1. نأخذ أحد الأعداد من النسق الأصلي ونضعه في السطر المقابل لخانة آحاده فمثلا الرقم 518 يوضع في السطر 8.
2. نكرر العمل السابق حتى انتهاء الأعداد, ونسمي هذا الطور من الخوارزمية بطور التوزيع distribution pass .
3. نقوم الآن بإعادة العناصر من المصفوفة الثنائية إلى النسق الأصلي بأن نبدأ بأول عنصر في السطر رقم 0 فالعناصر التالية له في السطر ذاته ثم ننتقل إلى عناصر الأسطر 1,2,..9 بالتتالي, ونسمى هذا الطور بطور التجميع gathering pass.
4. نكرر طوري التوزيع والتجميع عددا من المرات بعدد خانات العدد الأكبر في النسق (العدد الأعظم المتوقع وجوده في النسق), فنحصل في النهاية على العناصر مرتبة من الأكبر إلى الأصغر.

ولتكن لدينا العناصر التالية:

518	427	296	182	9	71	933	327	485	654	100	481

يبين الجدول (6-14) التالي نهاية طور التوزيع الأول (حسب خانة الآحاد):

6-4-8- خوارزمية الـ BUCKET SORT:

518 427 296 182 9 71 933 327 485 654 100 481

	0	1	2	3	4	5	6	7	8	9	10	n-1=11
0	100											
1	071	481										
2	182											
3	933											
4	654											
5	485											
6	296											
7	427	327										
8	518											
9	009											

الجدول (6-14) - نهاية طور التوزيع الأول

وبعد انتهاء التوزيع الأول نقوم بالتجميع الأول فنحصل على العناصر التالية مرتبة حسب خانة الآحاد:

100 71 481 182 933 654 485 296 427 327 518 9

وفي نهاية التوزيع الثاني(حسب خانة العشرات) نجد الجدول (6-15):

6-4-8- خوارزمية الـ BUCKET SORT:

100 71 481 182 933 654 485 296 427 327 518 9

	0	1	2	3	4	5	6	7	8	9	10	n-1=11
0	100	009										
1	518											
2	427	327										
3	933											
4												
5	654											
6												
7	071											
8	481	182	485									
9	296											

الجدول (6-15)- نهاية طور التوزيع الثاني

نجمّع من جديد فنحصل على:

100 9 518 427 327 933 654 71 481 182 485 296

وفي نهاية التوزيع الثالث(حسب خانة المئات): نجد الجدول (6-16) :

6-4-8- خوارزمية الـ BUCKET SORT:

100 9 518 427 327 933 654 71 481 182 485 296

	0	1	2	3	4	5	6	7	8	9	10	n-1=11
0	009	071										
1	100	182										
2	296											
3	327											
4	427	481	485									
5	518											
6	654											
7												
8												
9	933											

الجدول (6-16) -نهاية طور التوزيع الثالث

نجمّع من جديد فنحصل على:

9 71 100 182 296 327 427 481 485 518 654 933

وفي نهاية التوزيع الثالث(حسب خانة المئات): نجد الجدول (6-16) :

6-4-8- خوارزمية الـ BUCKET SORT:

وبهذا يكون قد تم ترتيب عناصر النسق تصاعدياً, أما الترتيب التنازلي فيمكن إنجازه بأن نقوم بالتجميع بدءاً بالسطر 9 و انتهاءً بالسطر 0 , بعد إجراء 3 أطوار توزيع و3 أطوار تجميع في كل منها n عملية وبهذا ستكون درجة تعقيد هذه الخوارزمية 3*2*n=6n أي أن T(n)=O(n) فالخوارزمية خطية وبهذا تكون سريعة جدا ولا يزداد زمنها تربيعياً مع زيادة عدد العناصر, ولو اعتبرنا تابعية الزمن لعدد الخانات الأعظمي وليكن m فإن الزمن يصبح T(n,m)=O(n*m) أي أنه خطي في n وكذلك في m ولكن بما أن m يساوي اللغاريتم العشري لأكبر العناصر وليكن هذا الرقم هو max فهو أصغر بكثير من n لهذا يصعب أن ترقى درجة التعقيد إلى n^2 ونستطيع أن نكتب: $T(n,max) = O(\log_{10}(max)*n)$

ولكن يؤخذ على هذه الخوارزمية حاجتها إلى الكثير من الذاكرة حيث تستهلك ذاكرة إضافية تساوي عشرة أضعاف الذاكرة اللازمة للعناصر الأصلية,وتقدم بذلك مثالاً شديد الوضوح على مقايضة زمن التنفيذ بالذاكرة.

ملاحظة1:

فرضنا بداية أن النسق لا يحوي أعداداً سالبة ولكن لو كانت الحالة كذلك فإننا نحتاج إلى مصفوفة ثنائية أخرى لترتيب الأعداد السالبة فيه ففي طور التوزيع نختبر إشارة العدد فإن كان موجبا وضعناه في المصفوفة الأولى وإن كان سالبا وضعناه في السطر المناسب من المصفوفة الثانية وعند التجميع نأخذ من الأولى بالترتيب من 0 إلى 9 أما من المصفوفة الثانية فنأخذ من 9 إلى 1 .

6-4-8- خوارزمية الـ BUCKET SORT:

ملاحظة2: يمكن اختصار الزمن اللازم لعملية التجميع وذلك بحجز ذاكرة إضافية مضاعفة حيث ننشئ مصفوفتين ثنائيتي البعد ونقوم بالتوزيع الأول في إحداهما ثم ننقل العناصر من إحداهما إلى الأخرى في كل عملية توزيع وفي النهاية تقوم بعملية تجميع واحدة.

يبين النص البرمجي (6-17) التنفيذ الفعلي لخوارزمية الـ Bucket Sort باستخدام لغة C++ وذلك في حالة عدم وجود أعداد سالبة, وفيه نستخدم التابع pow من المكتبة الرياضية math.h للقيام بعملية رفع الرقم 10 إلى القوة pass ولكن هذا التابع يعيد قيمة حقيقية من النوع (double) لذلك كان لابد من تحويل الناتج (قسرا - Type cast) إلى عدد صحيح باستخدام التعبير nt(pow(10,pass))

ليكون عامل القسمة / عامل قسمة صحيحة ثم يتم حساب باقي القسمة الصحيحة على 10 في التعبير k= (A[j] / int(pow(10,pass))) % 10;

والهدف من هذه العمليات الحصول على الخانة رقم (pass+1) من العدد(بدءاً بالآحاد) الذي تتم معالجته, كما نستخدم النسق index ذي العناصر العشرة للاحتفاظ بالموقع الذي يجب الإضافة في في كل سطر من المصفوفة B عند كل عملية توزيع ويقوم التابع MakeArrayEqualToZero بإعادة القيم في هذا النسق إلى الصفر قبل البداية بعملية التوزيع.

النص البرمجي (6-17):

6-4-7- خوارزمية الترتيب السريع 5b

```
Before sorting :
5   50   20   90   70    2    8   80   23   45

   L                                         R
   5   50   20   90   70    2    8   80   23   45

        l                              r
   5   50   20   90   70    2    8   80   23   45
   5   23   20   90   70    2    8   80   50   45

                 l              r
   5   23   20   90   70    2    8   80   50   45
   5   23   20    8   70    2   90   80   50   45

                      l    r
   5   23   20    8   70    2   90   80   50   45
   5   23   20    8    2   70   90   80   50   45

                         lr
   5   23   20    8    2   70   90   80   50   45

   5   23   20    8    2   45   90   80   50   70
  ==  ==   ==   ==   ==    |   ==   ==   ==   ==
```

تم العمل على كيفية إخراج مراحل الترتيب الثنائي ليصبح كما في الشكل المجاور ونترك للطالب تنفيذ ذلك كتدريب له.

6-4-7- خوارزمية الترتيب السريع 6b

```
 L                R
 5   23   20   8   2

lr
 5   23   20   8   2
_____________________
 2   23   20   8   5
 |   ==   ==  ==  ==
_____________________
 L            R
23   20   8   5
lr
23   20   8   5
_____________________
 5   20   8  23
 |   ==  ==  ==
_____________________
L        R
20   8  23

     lr
20   8  23
_____________
```

6-4-7- خوارزمية الترتيب السريع 7b

```
20    8   23
  ==   ==    |

_____________
  L     R
  20    8

_________
   8   20
   |   ==

_________

  L              R
  90   80   50   70

   l          r
  90   80   50   70
  50   80   90   70

       lr
  50   80   90   70

_____________________
  50   70   90   80
  ==    |   ==   ==

_____________________
```

```
  L    R
  90   80

_________
  80   90
   |   ==

_________

After  sorting :
2   5   8   20   23   45   50   70   80   90
```

انتهت محاضرة الأسبوع 2