



كلية الهندسة - قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا.د. علي عمران سليمان

محاضرات الأسبوع الثالث

فعالية الخوارزمية 3

Algorithm Efficiency 3

الفصل الأول 2025-2026

6-1- Introduction:

6-1-1- Classification methods

6-1-2-algorithm properties

6-1-3-algorithm types

6-2- Algorithm effectiveness:

6-3- Search algorithms:

6-3-1- search concept:

6-3-2- Linear (sequential) search algorithm:

6-3-3- Binary search algorithm:

6-4- Sort algorithms

sort concept, bubble sorting, swap sorting, selection sorting (selective sorting), insertion sorting, Bucket sorting, merging concept, merge sorting, quick sorting,.

1-6- مقدمة : Why Data Structures

1-1-6- طرق التصنيف.

2-1-6- خصائص الخوارزمية

3-1-6- أنواع الخوارزميات 1

2-6- فعالية الخوارزمية :

3-6- خوارزميات البحث:

1-3-6- مفهوم البحث:

2-3-6- خوارزمية البحث الخطي (التتابعي):

3-3-6- خوارزمية البحث الثنائي :

4-6- خوارزميات الترتيب.

مفهوم الترتيب، الترتيب الفقاعي، الترتيب بالتبديل، الترتيب بالاختيار (الترتيب الانتقائي)، الترتيب بالحشر، ترتيب Bucket Sort، مفهوم الدمج، الترتيب الدمج، الترتيب السريع.

- ADTs, Data Structures, and Problem Solving with C++: International Edition, by Larry R. Nyhoff, Publisher: Pearson , 2005
- Data Structures and Algorithms in Java, 6 th, [Roberto Tamassia](#), [Michael T. Goodrich](#), [Michael H. Goldwasser](#), Pub. Wiley 2014
- د.علي سليمان، بني معطيات بلغة JAVA، بني معطيات بلغة C++، بني معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

SWAP SORT ALGORITHM 1



3-4-6- خوارزمية الترتيب بالتبديل 1

SWAP SORT ALGORITHM

3-4-6- خوارزمية الترتيب بالتبديل :

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

SWAP SORT ALGORITHM

(* Algorithm to sort the list $A[1], \dots, A[n]$ 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 $A[\text{first}] > A[\text{second}]$ then do the following:
(* swap $A[\text{first}] \leftrightarrow A[\text{second}]$)
 - b11. set temp equal to $A[\text{first}]$ b12. set $A[\text{first}]$ equal to $A[\text{second}]$ b13. set $A[\text{second}]$ equal to temp
 - b2. set second equal to second +1
 - c. set first equal to first +1

SWAP SORT ALGORITHM 2

3-4-6- خوارزمية الترتيب بالتبديل 2

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

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

first=1;second=2

نبادل 1 و2

first=1;second=3

نبادل 1 و3

first=1;second=4

نبادل 1 و4

first=1;second=5

نبادل 1 و5

first=1;second=6

نبادل 1 و6

first=1;second=7

نبادل 1 و7

first=1;second=8

نبادل 1 و8

first=2;second=9

نهاية المسح الأول

الجدول (9-6) 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:

SWAP SORT ALGORITHM 3

3-4-6- خوارزمية الترتيب بالتبديل 3

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

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

first=2;second=3

نبادل 2 و3

first=2;second=4

نبادل 2 و4

first=2;second=5

نبادل 2 و5

first=2;second=6

نبادل 2 و6

first=2;second=7

نبادل 2 و7

first=2;second=8

نبادل 2 و8

first=3;second=9

نهاية المسح الثاني

الجدول (10-6) المسح الثاني لخوارزمية الترتيب بالتبديل

هكذا نجد أنه في نهاية المسح الثاني أصبح ثاني أصغر العناصر 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 (11-6):

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

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

first=2;second=3

لنبادل 3 و2

first=2;second=4

نبادل 4 و2

first=2;second=5

نبادل 5 و2

first=2;second=6

نبادل 6 و2

first=2;second=7

نبادل 7 و2

first=2;second=8

نبادل 8 و2

first=3;second=9

نهاية المسح الثاني

الجدول (10-6) المسح الثاني لخوارزمية الترتيب بالتبديل

هكذا نجد أنه في نهاية المسح الثاني أصبح ثاني أصغر العناصر 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 (11-6):

SWAP SORT ALGORITHM 4

3-4-6- خوارزمية الترتيب بالتبديل 4

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

نهاية المسح الثالث

نهاية المسح الرابع

نهاية المسح الخامس

نهاية المسح السادس

نهاية المسح السابع

الجدول (6-11) نهايات المسح الباقية

هكذا يتم في كل مسح انتقال أصغر عناصر النسق - بدءاً 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 حتى تصبح مقارنة وتبديل واحد.

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 مرتباً ترتيباً عكسياً لما هو مطلوب):

SWAP SORT ALGORITHM 5

3-4-6- خوارزمية الترتيب بالتبديل 5

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

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

SWAP SORT ALGORITHM 6

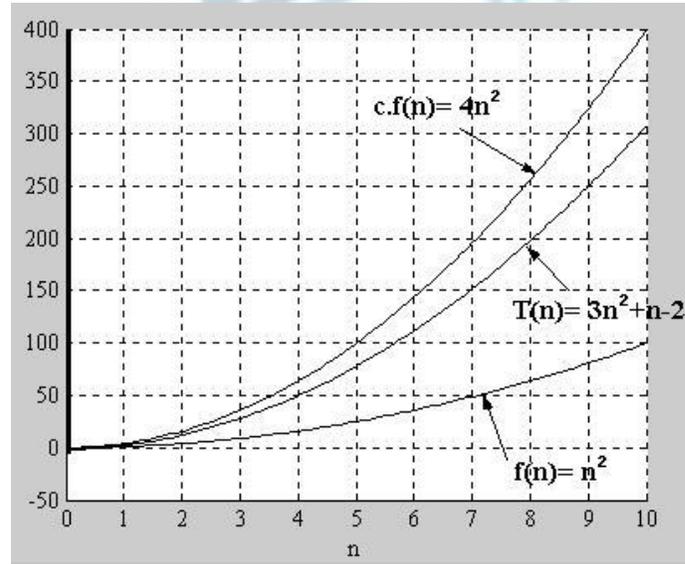
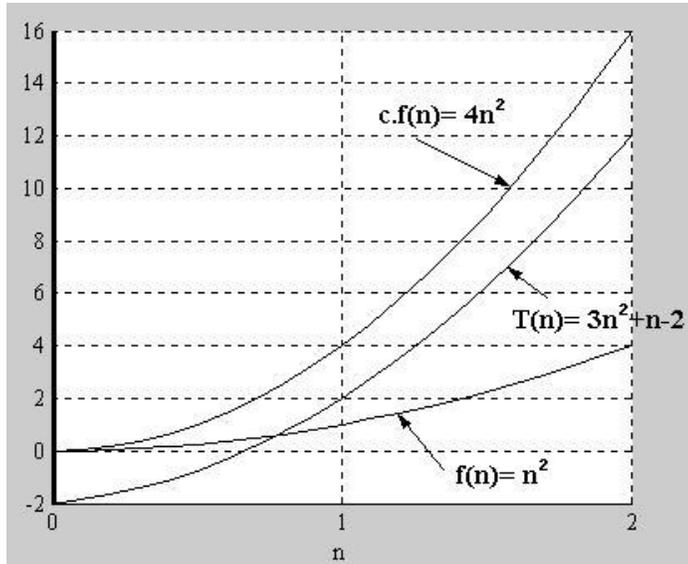
6-4-3- خوارزمية الترتيب بالتبديل

وعليه نستطيع التعبير عن زمن التنفيذ باستخدام تدوين BON كما يلي: $T(n) = 3N^2 + N - 2 \Rightarrow T(n) = O(n^2)$

$$3N^2 + N - 2 \leq 4N^2 \text{ for all } N \geq 0$$

حيث أن:

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



الشكل (7-6) يبين أن $4.f(n) \geq T(n)$

عندما $n \geq 0$

والبرنامج (8-6) التالي يحقق هذه الخوارزمية

حيث التابعان input و output هما كما في

البرنامج السابق تماماً:

```
#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



8-3-4-6 خوارزمية الترتيب بالتبديل

```
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++; }
```

```

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 ليطبّع 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

4-4-6- خوارزمية الترتيب بالاختيار (خوارزمية الترتيب الانتقائي):

يقوم مبدأ هذه الخوارزمية على عدة مسوحات للنسق (N-1) مرة، حيث يتم في المسح الأول المرور على العناصر جميعها لمعرفة العنصر الأصغر بينها (عملية بحث خطي عن أصغر العناصر بدءاً من A[1] وانتهاءً بـ A[N]) ثم يتم مبادلتها بالعنصر الأول، وفي المسح الثاني يتم البحث عن أصغر العناصر بدءاً من A[2] وانتهاءً بـ A[n] وتتم مبادلتها بالعنصر الثاني وهكذا دواليك، الخوارزمية (8-6) 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] \leftrightarrow 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

بالتدقيق في هذه الخوارزمية نجدها لا تختلف 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 نسبياً من العناصر.
- سيظهر هذا الفارق بشكل أوضح عندما تكون العناصر المخزنة في النسق عبارة عن سجلات ذات حجم بيانات كبير نسبياً تحوي على حقل مفتاح Key Field يتم الترتيب وفقه وعندها ستشكل عمليات المبادلة هدراً كبيراً 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 حقيقيين عدة أضعاف الزمن اللازم لمقارنة عددين صحيحين ومقارنة اسمين يتألف كل منهما من 10 أحرف عشرة أضعاف الزمن اللازم لمقارنة عددين حقيقيين.

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 مثلاً وتتألف المعطيات من الحقول التالية (أيضاً 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.

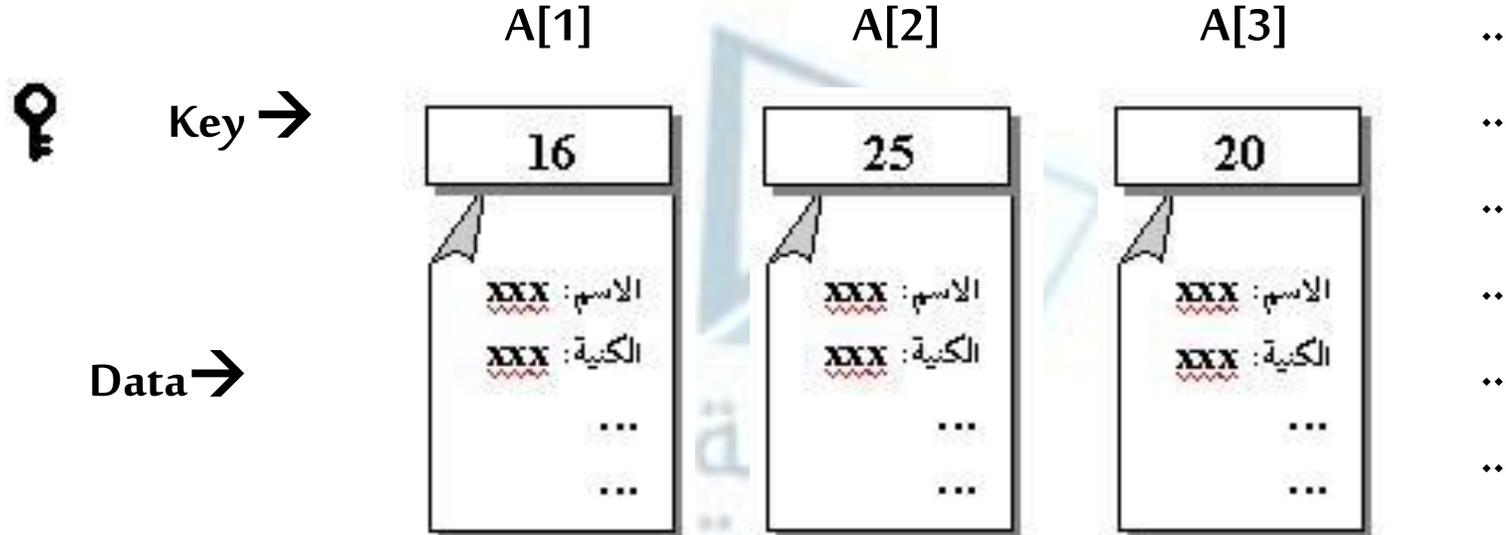
مكان الميلاد : بطول أعظمي 20 حرف مثلاً.

تاريخ الميلاد : السنة : عدد صحيح

: الشهر : عدد صحيح

: اليوم : عدد صحيح

السنة الدراسية : رقم صحيح



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

فإذا افترضنا أن العدد الصحيح يحتل 2Byte والحرف 1Byte فإن السجل الواحد سيحتل ما مقداره $(5 \times 2 + 5 \times 20 \times 1 + 1 \times 150 \times 1) = 260 \text{ byte}$ 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

ضعفاً من الزمن اللازم لمبادلة عددين صحيحين، ويستطيع القارئ هنا أن يتنبأ بالوفّر الزمني الذي ستقدمه خوارزمية الترتيب بالاختيار عن خوارزمية الترتيب بالتبديل عند ترتيب نسق من العناصر يحوي مثلاً 100 سجل. حيث ستقوم خوارزمية الترتيب بـ $\frac{(N-1) \cdot N}{2} = 4950$ عملية مقارنة مع العدد ذاته من عمليات المبادلة في أسوأ الاحتمالات، أما خوارزمية الاختيار فتقوم بـ 4950 عملية مقارنة و 100 عملية مبادلة في أسوأ الاحتمالات، أي أن هناك توفيراً لـ 4850 عملية مبادلة كل منها توفر 130 ضعفاً 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 الترتيب بالتبديل هي عندما يكون النسق مرتباً 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 لو كان 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 5



4-4-6- خوارزمية الترتيب الانتقائي 5

```
#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 6



4-4-6- خوارزمية الترتيب الانتقائي 6

```
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 7



4-4-6- خوارزمية الترتيب الانتقائي 7

```
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");
}
```

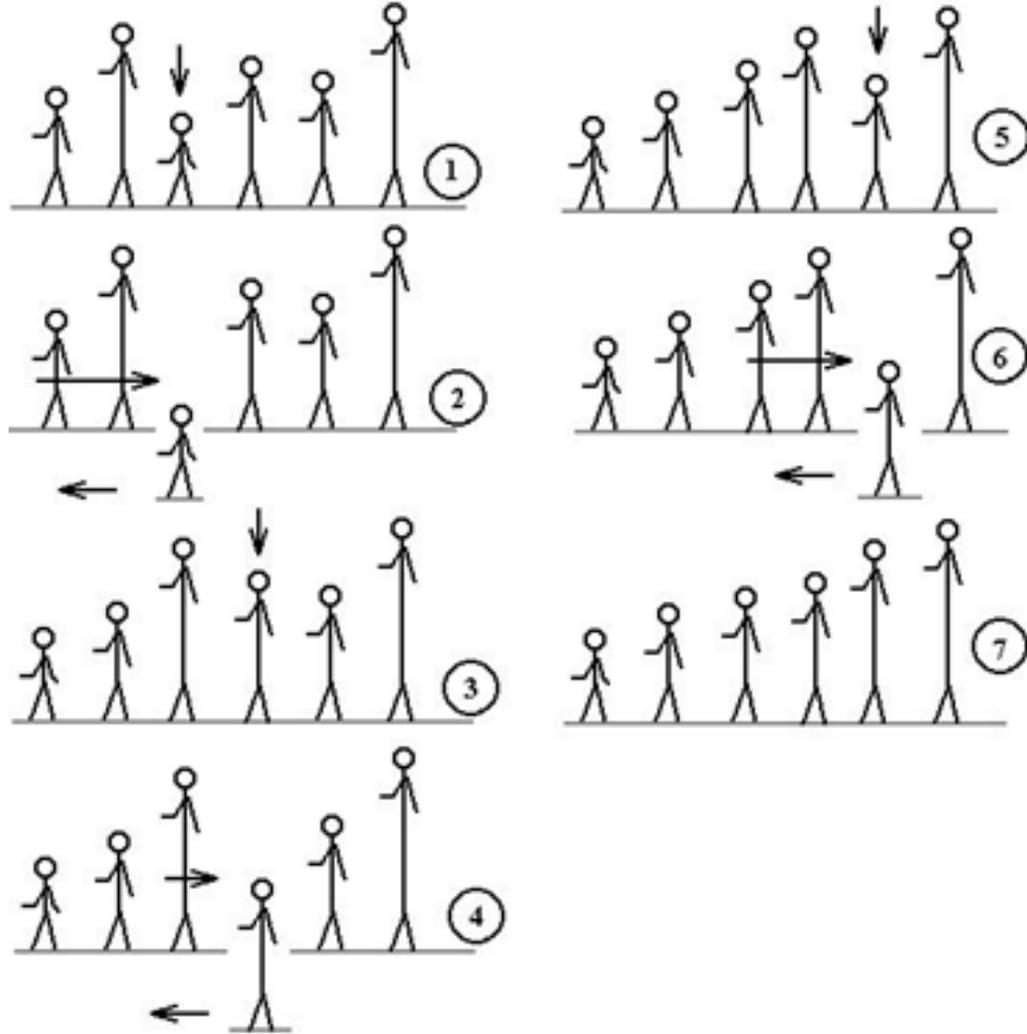
تعتبر خوارزمية الترتيب بالحشر من خوارزميات الترتيب الشائعة الاستخدام فهي مثلاً الطريقة التي نقوم بها بترتيب أوراق اللعب بشكل فطري وهي تعتمد على الوصول إلى العناصر الثاني فالثالث فالرابع ووضعه في مكانه الصحيح (حشره في المكان الصحيح) وذلك بإزاحة العناصر اللازم إزاحتها (وهي العناصر السابقة له والأكبر منه عند الترتيب التصاعدي) لإتاحة مكان لوضع ذلك العنصر فيه كما نستطيع تبين 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 سابقاً 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

5-4-6- خوارزمية الترتيب بالحشر 2



الشكل (9-6)- 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 (9-6) الترتيب بالحشر كما يأتي:

INSERTION SORT ALGORITHM 3



5-4-6- خوارزمية الترتيب بالحشر 3

INSERTION SORT ALGORITHM

(* Algorithm to sort the list $A[1], \dots, A[n]$ 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

5-4-6- خوارزمية الترتيب بالحشر 4

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

الجدول (6-13) زمن تنفيذ خطوات الترتيب بالحشر

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

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

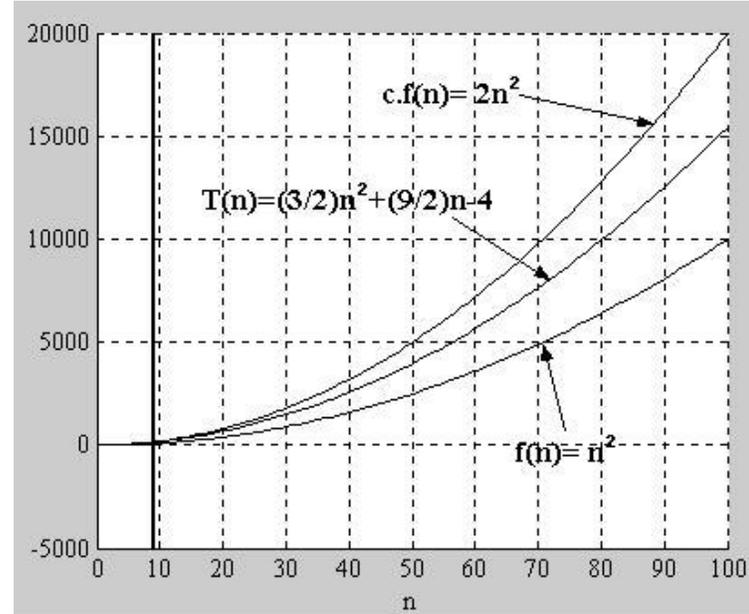
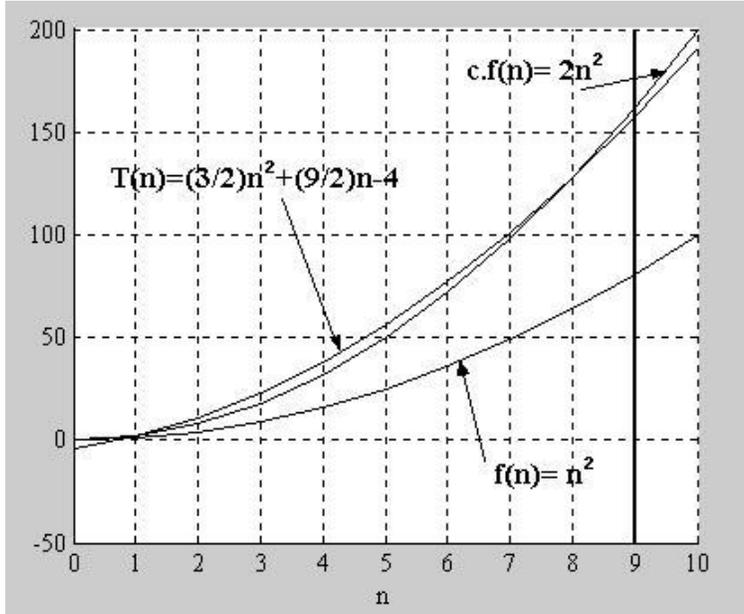
حيث أن:

$$\frac{3}{2} N^2 + \frac{9}{2} N - 4 \leq 2N^2 \text{ for all } N \geq 9$$

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

INSERTION SORT ALGORITHM 5

5-4-6- خوارزمية الترتيب بالحشر 5



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

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:

- لأول : $A[1] \leq A[2]$ وعندها لا تتم أية إزاحة وتنتهي الحلقة الداخلية وتم بعدها الانتقال لمعالجة العنصر $A[3]$ وما بعده دون أية مشاكل.
- الثاني : $A[1] > A[2]$ وعندها سيتم الدخول إلى الحلقة الداخلية وستبدأ عملية إزاحة من غير المعلوم متى تنتهي وذلك لأنه سيتم تجاوز حدود النسق (سيتم الوصول إلى العناصر $A[0], A[-1], \dots$) 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:

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.

INSERTION SORT ALGORITHM 8

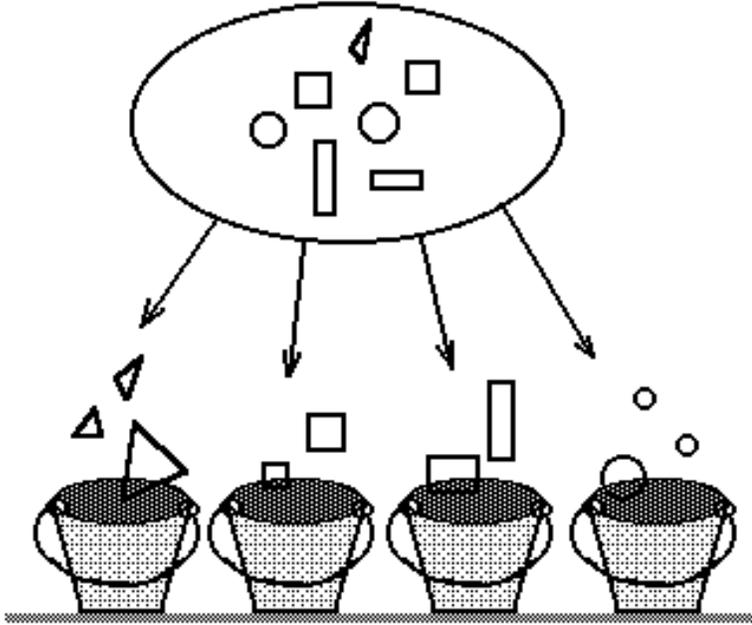


5-4-6- خوارزمية الترتيب بالحشر 8

والبرنامجان التاليان (6-11) و(6-12) يحققان خوارزمية الترتيب بالحشر أولهما يستخدم شرطاً إضافياً 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;
```

```
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");
}
```



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 الأحاد ثم العشرات وانتهاءً 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 أعمدها يساوي n (عدد العناصر المراد ترتيبها) مرقمة من 0 وحتى $n-1$ وللقيام بالترتيب نقوم بالتالي:

1. نأخذ أحد الأعداد من النسق الأصلي ونضعه في السطر المقابل لخانة أحاده 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 9,2,1 بالتتالي، ونسمي 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 الأحاد):

BUCKET SORT ALGORITHM 1



8-4-6- خوارزمية ال 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											

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

الجدول (6-14) - نهاية طور التوزيع الأول

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

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

BUCKET SORT ALGORITHM 1



8-4-6- خوارزمية ال 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											

نجمّع من جديد فنحصل على:

الجدول (6-15)- نهاية طور التوزيع الثاني

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

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

BUCKET SORTALGORITHM 1



8-4-6- خوارزمية ال 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											

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) :

وهذا يكون قد تم ترتيب عناصر النسق تصاعدياً، أما الترتيب التنازلي فيمكن إنجازه بأن نقوم بالتجميع بدءاً بالسطر 9 وانتهاءً بالسطر 0، بعد إجراء 3 أطوار توزيع و 3 أطوار تجميع في كل منها n عملية وهذا ستكون درجة تعقيد هذه الخوارزمية $T(n) = O(n)$ أي أن $3 \times 2 \times n = 6n$ فالخوارزمية خطية وهذا تكون سريعة جداً ولا يزداد زمنها تربيعياً مع زيادة عدد العناصر، ولو اعتبرنا تابعة الزمن لعدد الخانات الأعظمي وليكن m فإن الزمن يصبح $T(n, m) = O(n \times m)$ 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: $T(n, \max) = O(\log_{10}(\max) \times n)$

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

ملاحظة 1:

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.

ملاحظة 2: 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 ال 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

$$\text{int}(\text{pow}(10, \text{pass}))$$

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

والهدف من هذه العمليات الحصول على الخانة رقم (pass+1) من العدد (بدءاً بالأحاد) الذي تتم معالجته، كما نستخدم النسق index ذي العناصر العشرة للاحتفاظ بالموقع الذي يجب الإضافة 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):

ملاحظة 2: يمكن اختصار الزمن اللازم لعملية التجميع وذلك بحجز ذاكرة إضافية مضاعفة حيث ننشئ مصفوفتين ثنائيتي البعد ونقوم بالتوزيع الأول في إحدهما ثم ننقل العناصر من إحدهما إلى الأخرى في كل عملية توزيع وفي النهاية تقوم بعملية تجميع واحدة.

يبين النص البرمجي (6-17) التنفيذ الفعلي لخوارزمية ال 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

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

والهدف من هذه العمليات الحصول على الخانة رقم (pass+1) من العدد (بدءاً بالأحاد) الذي تتم معالجته، كما نستخدم النسق index ذي العناصر العشرة للاحتفاظ بالموقع الذي يجب الإضافة 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):

انتهت محاضرات الأسبوع الثالث

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 مثلا 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]

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

- N_a هو طول النسق A و N_b هو طول النسق B .
- $N_c \geq N_a + N_b$ حيث N_c هو طول النسق C حيث لا يوجد عناصر مكررة في A ولا عناصر مكررة في B ولا عناصر مكررة في A و B معاً (بهدف التبسيط).

1. set l_a equal to 1.
2. set l_b equal to 1.
3. set l_c equal to 1.
4. while ($l_a \leq N_a$) and ($l_b \leq N_b$) do the following:
 - 4a. if $A[l_a] < B[l_b]$ then
 - 4a1. set $C[l_c]$ equal to $A[l_a]$
 - 4a2. set l_a equal to $l_a + 1$
 - else

MERGE SORT ALGORITHM 3



6-4-6- خوارزمية الترتيب بالدمج 3

4a3. set $C[lc]$ equal to $B[lb]$

4a4. set lb equal to $lb+1$

4b. set lc equal to $lc+1$

5. if ($la < lb$) then:

5a. while ($la \leq Na$) do the following:

5a1. set $C[lc]$ equal to $A[la]$

5a2. set la equal to $la+1$

5a3. set lc equal to $lc+1$

else

5b. while ($lb \leq Nb$) do the following:

5b1. set $C[lc]$ equal to $B[lb]$

5b2. set lb equal to $lb+1$

5b3. set lc equal to $lc+1$



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

```
#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");
}
```

```
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++; }
    }
}
```

MERGE SORT 1

2-6-4-6- 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 أولاً 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 دمجياً في B ثم إعادة الناتج إلى A وتم مثل هذا التقسيم والترتيب باستخدام تابع عودي (Recursive Function) يستدعي نفسه أولاً ليرتب نصفي النسق ثم يدمجهما ولكنه لترتيب كل نصف سيقوم أيضاً باستدعاء نفسه من جديد لترتيب كل جزء على حدة ثم يقوم بالدمج ... وهكذا دواليك . والبرنامج (6-14) التالي ينفذ هذه الخوارزمية:

MERGE SORT 3

```
#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 4



4-6-4-2- خوارزمية الدمج 4

```
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 5



5-2-6-4-6 خوارزمية الدمج

```
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 6



6-4-6-2- خوارزمية الدمج 6

```
{   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];
}
```

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 (مساو في الطول للنسق الأصلي) كما أنها خوارزمية عودية الأمر الذي لا يمكن تحقيقه مع بعض لغات البرمجة التي لا تسمح بالاستدعاءات العودية.

ندرس الآن إحدى أهم خوارزميات الترتيب وربما الخوارزمية الأكثر استخداماً 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 مكس صغير نسبياً) وتتملك 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 معقداً 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.

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 بدءاً 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: لا شيء

يتم في البداية استدعاء الإجراء العودي مع قيم أولية $L=1$ 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 2,3,4 حتى يتقاطع مؤشر 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 يتلوه 1 مؤشر المسح من جهة اليمين.

QUICK SORT ALGORITHM 4



7-4-6- خوارزمية الترتيب السريع 4

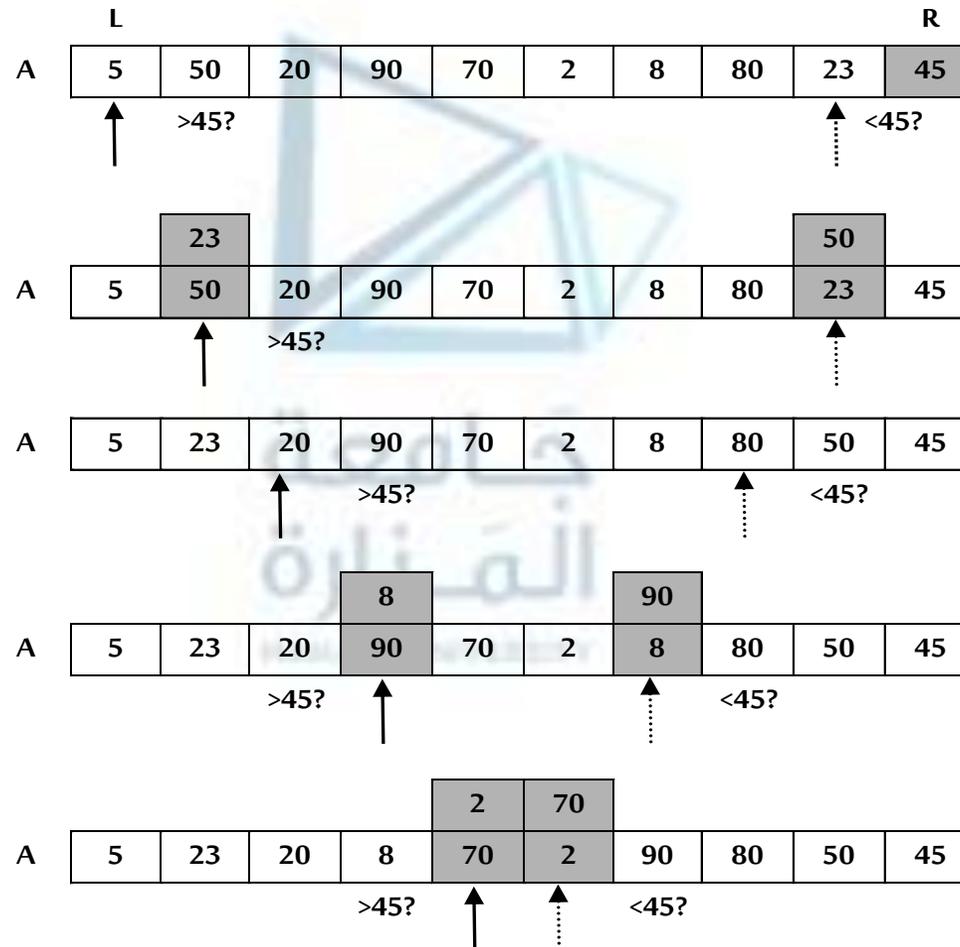
وهكذا نكتب الخوارزمية (6-13) كاملة باستخدام pseudo code:

```
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+1, 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:

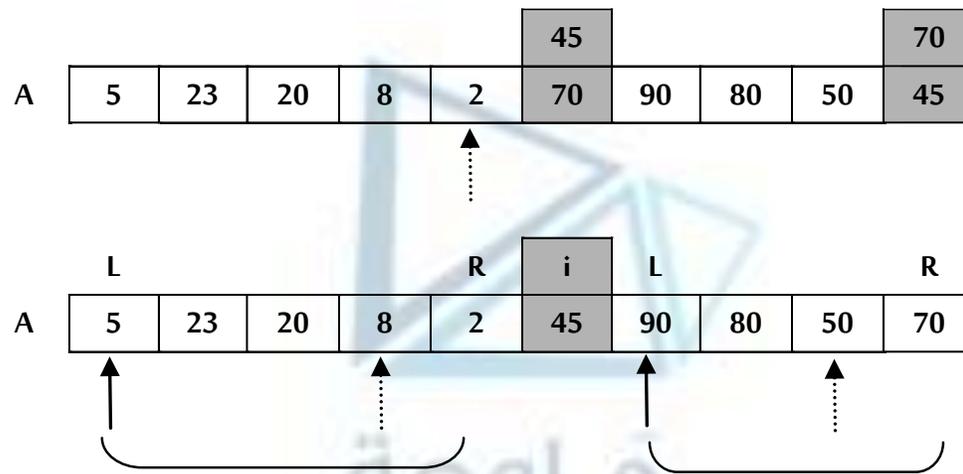
QUICK SORT ALGORITHM 5

7-4-6- خوارزمية الترتيب السريع 5



QUICK SORT ALGORITHM 6

6-4-7- خوارزمية الترتيب السريع



نرتب هذا الجزء بالطريقة ذاتها

نرتب هذا الجزء بالطريقة ذاتها

الشكل (6-7-13) 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); }
```

نقدم فيما يلي خوارزمية مميزة بما تقدمه من سرعة في عملية ترتيب عناصر النسق وفكرة هذه الخوارزمية تعتمد على فكرة بسيطة يمكن أن نشبهها بالمثل التوضيحي التالي:
ليكن لدينا مجموعة من القطع المعدنية بأشكال مختلفة (مثلثات - مستطيلات - مربعات - دوائر ...) موضوعة معا (مختلطة) ونرغب بفرز هذه العناصر حسب الشكل كلاً على حدة من الطرق الممكنة التالي:

نأخذ وعاء (جرديلاً) - (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):

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

== == == == == | == == == ==



7-4-6- خوارزمية الترتيب السريع 5

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

7-4-6- خوارزمية الترتيب السريع 5

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

