5.1 Inserting Sort

The steps of this algorithm are summarized as follows:

1- We start with the second element i = 2 in the original list and compare it with the first element i = 1 and put them in order and be ascending to the top of the list.

2- We take the third element i = 3 in the original list and compare it with the introduction to the list that contains the first and second element and put it in its correct location with them.

3- We take the fourth element i = 4 in the original list and compare it with the introduction to the list that contains the three elements and put it in its correct position between them.

4- We continue in this process until the last component and we will get the list in order.

Example: sort the following list items in ascending order (8 3 9 7 2 6 4)

	5	4	3	2	1	القائمة الاصلية
2	2	2	3	3 ←	3	8 ←
3	3	3	7	8	8 ←	3
4	6	7 ←	8 ←	9	9	← 9
6	7	8	9	7	7	7
7	8	9	2	2	2	2
8	9	6	6	6	6	6
9	4	4	4	4	4	4

-The number of elements n = 7

- The number of stages n-1 = 6, the arrow mark (←) indicating the element that was added.

Notes: - The average number of comparisons is n ^ 2/4 where n represents the number of list item

- Average no. of exchanges n ^ 4/4

-Inserting Sort function

```
const size=20;

int line[size],int i,m;

void insertionsort (int data[size],int n;
{
 int i,j,item;
 i=1;
while(i<n)
 {
  j=i;
  while((j>=1) && (data[j]<data[j-1]))
     {
         item= data[j];
         data[j]=data[j-1];
         data[j-1]=item;
          j--;
        }
   i++;
    }
  }
```

5.2 quick sort

The algorithm for this sort depends on the idea of segmentation and pasting. In the sort of bubble, the contiguous elements are compared and exchanged. Therefore, if the element is far from its correct location, in this case, we will need a large number of comparisons.

The rapid sorting algorithm addresses this weakness and allows comparisons between elements in far-flung locations and the lowest number of comparisons, as the idea of segmentation and pasting is based. The steps of this algorithm are summarized as follows:

1- Choose one of the menu items in the middle, and let it be x, meaning that the list is divided into two parts.

2- The survey starts from both directions, i.e. we search in the first half (left) from the list for the element whose value is greater than x and we look in the second half (right) from the list for the element whose value is smaller than x. We replace these two elements by making the first half of the list contain elements greater than x.

3- We take the first half of the list and treat it with the same previous method (i.e. splitting and pasting) and so on with the second half i.e. we continue splitting and pasting successively until all the elements of the overall list are arranged.

```
#include<iostream>
const size=10;
int ar[size];
void swap(int *x,int *y)
{
 int temp;
 temp=*x;
 *x=*y;
 *y=temp;
}
```

```
void quicksort(int list[size],int f,int l)
{
 int i,j,x;
 i=f;
 j=l;
 x=list[(i+j)/2];
 do
  {
   while(list[i]<x)
   i++;
   while(x<list[j])
   j--;
   if(i<=j)
    {
     swap(&list[i],&list[j]);
     i++;
     j--;
    }
   }while(i<=j);
   if(f<j)
   quicksort(list,f,j);
   if(i<l)
   quicksort(list,i,l);
}
```

Example: Use the quick sort algorithm (by adopting the center element as the order axis) to sort the following set of values in ascending order (20, 85, 60, 75, 70, 88, 50, 90, 33, 95.

A- We assume that the elements are stored in the list matrix as follows:

F=1, l=10, x= list (5)=70, i=1, j=10.

1	2	3	4	5	6	7	8	9	10
20	85	60	75	**70**	88	50	90	33	95
i									j

B. I=2, j=9, list(2) <x,x< list (9), i=j

1	2	3	4	5	6	7	8	9	10
20	85	60	75	**70**	88	50	90	33	95
	I=2								j=9

C.

1	2	3	4	5	6	7	8	9	10
20	33	60	75	**70**	88	50	90	85	95
		I=3					j=8		

D. I=4,j=7, list (4)< x , x < list (7), i<=j

1	2	3	4	5	6	7	8	9	10
20	33	60	75	**70**	88	50	90	85	95
			I=4			j=7			

E.

1	2	3	4	5	6	7	8	9	10
20	33	60	50	**70**	88	75	90	85	95
				I=5	j=6				

F. I=5, j=5, list (5) <x, x< list (j)

1	2	3	4	5	6	7	8	9	10
20	33	60	50	**70**	88	75	90	85	95
				I=j=5					

At this step, the list was divided into two parts, the left section contains all the numbers whose value is less than 70, and the right section contains all the numbers whose value is greater than 70.

The next step is to implement the algorithm repeatedly on each part in the same way, i.e. calling quick sort (1.4) in relation to the part whose elements are in locations 1 to 4 and calling quick sort (6,10) in relation to the part whose elements in the site from 6-10 Thus the process of repeat segmentation and arrangement continues.

5.3 .The merging sort is balanced two –way merge

This method is one of the types of external sort. The algorithm is summarized in the following steps:

1- Divide the original list (data) into two almost equal lists, and make it a, b

2- We compare the first item from list a with the first item from list b and put them in order in list c.

3- We compare the second component of list a with the second component of list b and put them in order in list d

4- Repeat steps 2,3 and we will get string of length 2 in both c, d.

5- In the same way we combine the elements of the lists c and d and put them in the lists a, b and we will be the elements of them with a length of 4.

6- We return the method by combining the elements of the lists a and b and placing them in the lists c, d and their elements will be of length 8.

7- We continue with this method until the final ranked list is obtained.

Example: Sort this list in ascending order using the merging algorithm ((18, 23, 02, 50, 42, 63, 20, 28, 33, 47, 3

The solution: The number of list items (n = 1) is divided into two lists approximately equal in number

A: 18, 23, 2, 50, 42

B: 63, 20, 28, 33, 47, 3

C: 18, 63, 2, 28, 42, 47

D: 20, 23, 33, 50, 3

A:18, 20, 23, 63, 3, 43, 47

B: 2, 28, 33, 50

C: 2, 18, 20, 23, 28, 50, 63

D: 3, 42, 47

A: 2,3,18,20,23,28,42,47,50,63

بعد سلسلة الخطوات اعلاه حصلنا على على عناصر القائمة مرتبة تصاعدية كما في A