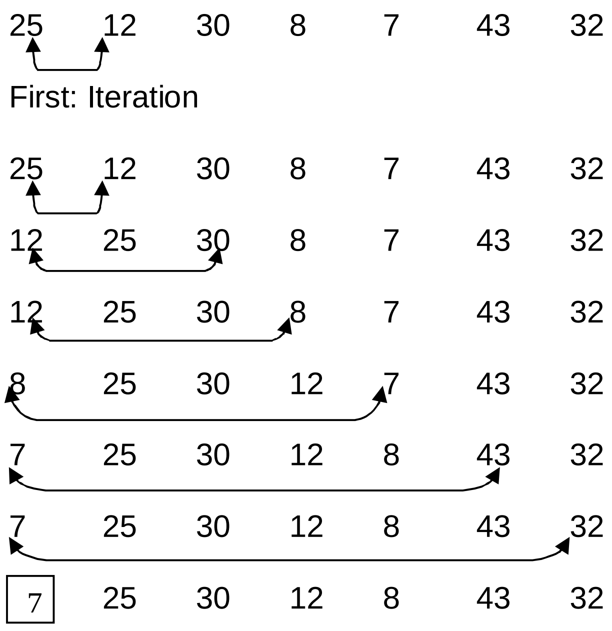
SORTING

Sorting is a process of arranging a set of data in some order. Usually, sorting will be either in ascending order or in descending order. Sorting technique can be mainly divided into two categories viz. internal sorting and external sorting. If all the data to be sorted all stored in the main memory, then it is called as internal sorting. If the data are stored in the auxiliary storage i.e. in floppy, tape etc, then the sorting is said to be external sorting. Let us discuss about different internal sorting techniques one by one.

Selection Sort

Selection sort is a simplest method of sorting technique. To sort the given list in ascending order, we will compare the first element with all the other elements. If the first element is found to be greater then the compared element, then they are interchanged. Thus at the end of first interaction, the smallest element will be stored in first position, which is its proper position. Then in the second interaction, we will repeat the procedure from second element to last element. The algorithm is continued till we get sorted list. If there are n elements, we require (n-1) iterations, in general.

Consider the example----



Second Iteration

$$\begin{tikzcd}
\boxed{7} & 30 \arrow[r, bend right, leftrightarrow] & 12 & 8 & 43 & 32 \\
\boxed{7} & 25 \arrow[rr, bend right, leftrightarrow] & 30 & 12 & 8 & 43 & 32 \\
\boxed{7} & 12 \arrow[rrr, bend right, leftrightarrow] & 30 & 25 & 8 & 43 & 32 \\
\boxed{7} & 30 \arrow[rrr, bend right, leftrightarrow] & 25 & 12 & 43 & 32 \\
\boxed{7} & 8 \arrow[rrrrr, bend right, leftrightarrow] & 30 & 25 & 12 & 43 & 32 \\
\boxed{7} & \boxed{8} & 30 & 25 & 12 & 43 & 32
\end{tikzcd}$$

Third Iteration

$$\begin{tikzcd}
\boxed{7} & \boxed{8} & 30 \arrow[r, bend right, leftrightarrow] & 25 & 12 & 43 & 32 \\
\boxed{7} & \boxed{8} & 25 \arrow[rr, bend right, leftrightarrow] & 30 & 12 & 43 & 32 \\
\boxed{7} & \boxed{8} & 12 \arrow[rrr, bend right, leftrightarrow] & 30 & 25 & 43 & 32 \\
\boxed{7} & \boxed{8} & 12 \arrow[rrrr, bend right, leftrightarrow] & 30 & 25 & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & 30 & 25 & 43 & 32
\end{tikzcd}$$

Fourth Iteration

$$\begin{tikzcd}
\boxed{7} & \boxed{8} & \boxed{12} & 30 \arrow[r, bend right, leftrightarrow] & 25 & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & 25 \arrow[rr, bend right, leftrightarrow] & 30 & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & 25 \arrow[rrr, bend right, leftrightarrow] & 30 & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & \boxed{25} & 30 & 43 & 32
\end{tikzcd}$$

Fifth Iteration

$$\begin{tikzcd}
\boxed{7} & \boxed{8} & \boxed{12} & \boxed{25} & 30 \arrow[r, bend right, leftrightarrow] & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & \boxed{25} & 30 \arrow[rr, bend right, leftrightarrow] & 43 & 32 \\
\boxed{7} & \boxed{8} & \boxed{12} & \boxed{25} & \boxed{30} & 43 & 32
\end{tikzcd}$$

Sixth Iteration

$$\begin{tikzcd}\boxed{7} & \boxed{8} & \boxed{12} & \boxed{25} & \boxed{30} & 43 \arrow[r, leftrightarrow, bend right] & 32\end{tikzcd}$$

7 8 12 25 30 32 43

Thus sorted list is:
7, 8, 12, 25, 30, 32, 43

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],n,i,temp,j;
        clrscr();
        printf("Enter the size of the array:");
        scanf("%d",&n);
        printf("\nEnter array elements:\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        for(i=0;i<n-1;i++)
        {
                for(j=i+1;j<n;j++)
                {
                        if(a[ i]>a[j])
                        {
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
                }
        }
        printf("\nSorted list is:\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        getch();
}
```

Bubble Sort

The bubble sort technique for sorting a list of data in ascending order is as follows: In the first iteration, the first element of the array is compared with the second element. If the first element is found to be greater than the second element, they are interchanged. Now, the second element is compared with the third and interchanged if required. In the same way, comparison is done till the last element. At the end of first iteration, the largest element will be stored at the last position. In the second iteration, again the comparison is done from the first element to last-but-one element. At the end of this iteration, the second largest element will be placed in its proper position. If there are 'n' elements in the given list, then after (n-1) iterations, the array gets sorted.

Consider the following list of integers to be sorted:

25 12 30 7 32 8

1st Iteration:

25	12	30	7	32	8	25>12 interchange
12	25	30	7	32	8	25<30 no interchange
12	25	30	7	32	8	30>7 interchange
12	25	7	30	32	8	30<32 no interchange
12	25	7	30	32	8	32>8 interchange
12	25	7	30	8	$\boxed{32}$	32 placed in its position

2nd Iteration:

12	25	7	30	8	$\boxed{32}$	12<25 no interchange
12	25	7	30	8	$\boxed{32}$	25>7 interchange
					$\boxed{32}$	

12	7	25	30	8		25<30 no interchange
12	7	25	30	8	$\boxed{32}$	30>8 interchange
12	7	25	8	$\boxed{30}$	$\boxed{32}$	30 also placed

3rd Iteration:

12	7	25	8	$\boxed{30}$	$\boxed{32}$	12>7 interchange
7	12	25	8	$\boxed{30}$	$\boxed{32}$	12<25 no interchange
7	12	25	8	$\boxed{30}$	$\boxed{32}$	25>8 interchange
7	12	8	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	25 is placed now

4th Iteration:

7	12	8	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	7<12 no interchange
7	12	8	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	12>8 interchange
7	8	$\boxed{12}$	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	12 is placed now

5th Iteration:

7	8	$\boxed{12}$	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	7<8 no interchange
7	$\boxed{8}$	$\boxed{12}$	$\boxed{25}$	$\boxed{30}$	$\boxed{32}$	8 is placed

Thus, the sorted list is:

7 8 12 25 30 32

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],n,i,temp,j;
        clrscr();
        printf("Enter the size of the array:");
```

```
	scanf("%d",&n);
	printf("\nEnter array elements:\n");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);

	for(i=0;i<n;i++)
	{
		for(j=0;j<n-i-1;j++)
		{
			if(a[ j]>a[j+1])
			{
				temp=a[j];
				a[j]=a[j+1];
				a[j+1]=temp;
			}
		}
	}
	printf("\nSorted list is:\n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);
	getch();
}
```

Quick Sort

As the name suggests, Quick Sort is a technique that will sort a list of data significantly faster than any other sorting techniques. This algorithm is based on the fact that – it is always easier and faster to sort two small arrays than one single big array.

Here, the given array is divided into two sub-arrays such that the elements at the left-side of some key element are less than the key element and the elements at the right-side of the key element are greater than the key element.

The dividing procedure is done with the help of two index variables and one key element as explained below –

i) Usually the first element of the array is treated as *key*. The position of the second element is taken as the first index variable *left* and the position of the last element will be the index variable *right.*

ii) Now the index variable *left* is incremented by one till the value stored at the position *left* is greater than the *key*.

iii) Similarly *right* is decremented by one till the value stored at the position *right* is smaller than the *key*.

iv) Now, these two elements are interchanged. Again from the current position, *left* and *right* are incremented and decremented respectively and exchanges are made appropriately, if required.

v) This process is continued till the index variables crossover. Now, exchange *key* with the element at the position *right.*

vi) Now, the whole array is divided into two parts such that one part is containing the elements less than the *key* element and the other part is containing the elements greater than the *key*. And, the position of *key* is fixed now.

vii) The above procedure (from step i to step vi) is applied on both the sub-arrays. After some iteration we will end-up with sub-arrays containing single element. By that time, the array will be stored.

Let us illustrate this algorithm using an example. Consider an array

a[7] ={25, 12, 30, 8, 7, 43, 32}

Let key =25
left = 1, the position of 12
right =6, the position of 32

First step: Compare *key* with *a[left]*

key	*left*					*right*
25	12	30	8	7	43	32

Now, *key* > *a[left]* (i.e. 25 > 12) is true.
So increment *left.* So, now *left* will be at the element 30.

Second step: Compare *key* with *a[left]*

key		*left*				*right*
25	12	30	8	7	43	32

Now, *key* > *a[left]* (i.e. 25 > 30) is false.
So stop incrementing *left.*

Third step: Compare *key* with *a[right].*

key		*left*				*right*
25	12	30	8	7	43	32

Now, *key* < *a[right]* (i.e. 25 < 32) is true.
So, decrement *right.* Thus, *right* will be at the element 43 now.

Fourth step: Compare *key* with *a[right].*

key		*left*			*right*	
25	12	30	8	7	43	32

Now, *key* < *a[right]* (i.e. 25 < 43) is true.
So, decrement *right.* Thus, *right* will be now at 7.

Fifth step: Compare *key* with *a[right].*

key		*left*		*right*		
25	12	30	8	7	43	32

Now, *key* < *a[right]* (i.e. 25 < 7) is false.
So, stop decrementing *right.*

Sixth step: Exchange the values of *a[left]* (30) and *a[right]* (7)*.* Thus, array will be –

key		*left*		*right*		
25	12	7	8	30	43	32

Seventh step: Again start the procedure from beginning. That is, compare *key* with *a[left].*

key		*left*		*right*		
25	12	7	8	30	43	32

Now, *key* > *a[left]* (i.e. 25 > 7) is true.
So, increment *left.* Now, *left* will be at 8.

Eighth step: Compare *key* with *a[left].*

key			*left*	*right*		
25	12	7	8	30	43	32

Now, *key* > *a[left]* (i.e. 25 > 8) is true.
So, increment *left.* Now, *left* will be at 30.

Ninth step: Compare *key* with *a[left].*

key				*left, right*		
25	12	7	8	30	43	32

Now, *key* > *a[left]* (i.e. 25 > 30) is false.
So, stop incrementing *left.*

Tenth step: Compare *key* with *a[right].*

key				*left, right*		
25	12	7	8	30	43	32

Now, *key* < *a[right]* (i.e. 25 < 30) is true.
So, decrement *right.* Thus, *right* will be at 8 now.

Eleventh step: The array looks like –

key			*right*	*left*		
25	12	7	8	30	43	32

As the index variables *left* and *right* cross-over, exchange *key* (25) with *a[right]* (8)*.*

The array would be –

8	12	7	25	30	43	32

Thus, all the elements at the left-side of *key*(i.e. 25) are less than *key* and all the elements at the right-side of *key* are greater than *key*. Hence, we have got two sub-arrays as –

{8, 12, 7}   25   {30, 43, 32}

Now, the position of 25 will not get changed. But, we have to sort two sub-arrays separately, by referring the above explained steps.

Proceeding like this, we will get the sorted list.

Program:

```
#include<stdio.h>
quick_sort(int x[], int low, int high)          //Function to apply quick sort technique
{
        int pos;
        if (low < high)
        {
                pos = partition(x,low,high);
                quick_sort(x,low,pos-1);
                quick_sort(x,pos+1,high);
        }
        return;
}

int partition(int x[], int low, int high)       //Function for partitioning the array
{
        int key, temp, true = 1;
        int left, right;

        key = x[low];
        left = low +1;
        right = high;

        while(true)
        {
                while ((left < high) && (key >= x[left]))
                        left++;
                while(key < x[right])
                        right--;
                if(left < right)
                {
                        temp = x[left];
                        x[left] = x[right];
                        x[right] = temp;
                }
                else
                {
                        temp = x[low];
                        x[low] = x[right];
                        x[right] = temp;
                        return(right);
                }
        }
        return 0;
}
```

```
void main()
{
	int a[10],n,i,low,high;
	clrscr();

	printf("Enter array size\n");
	scanf("%d",&n);

	printf("Enter the elements\n");
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);

	low = 0;
	high = n-1;

	quick_sort(a,low,high);

	printf("The sorted list is \n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);

	getch();
}
```

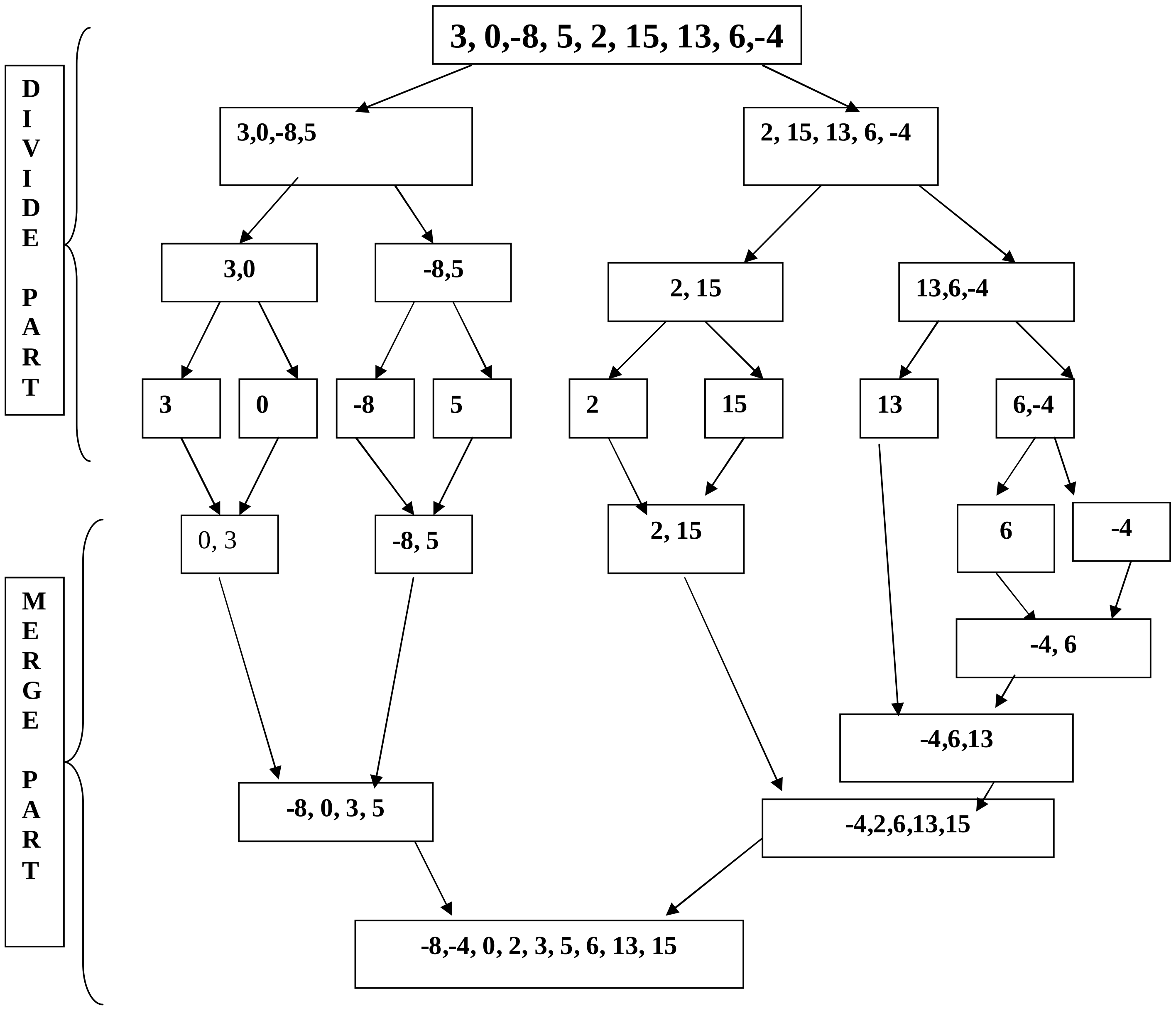
NOTE: The bubble sort and quick sort techniques are usually called as exchange sort techniques. Because, both of these involves the procedure of exchanging the elements in some or the other situation.

Merge Sort

The procedure for merge sort contains two main parts viz. divide and merge.
Divide: The original array is divided into two equal parts. Then each sub-array is divided into two equal parts. This method is continued till each sub array contains only one element.
Merge: The first element of first sub-array is compared with first element of the second sub-array. The lesser among these is put into result-array. The remaining element is compared with the second element of the other array. The procedure is continued till both the arrays get exhausted.

The divide and merge parts are done recursively on given array to get sorted list.

Consider an array: 3, 0, -8, 5, 2, 15, 13, 6, -4



Program:

```
#include<stdio.h>
#include<conio.h>

void Merge(int b[10], int c[10],int a[20],int p, int q)
{
        int i=0,j=0,k=0;

        while(i<p && j<q)
        {
                if(b[i]<c[j])
                {
                        a[k]=b[i];
                        i++;
                }
                else
                {
                        a[k]=c[j];
                        j++;
                }
                k++;
        }
        while(i<p)
        {
                a[k]=b[i];
                i++;
                k++;
        }
        while(j<q)
        {
                a[k]=c[j];
                j++;
                k++;
        }
}

void MergeSort(int a[20],int n)
{
        int b[20],c[20],i,j,p,q;

        if(n>1)
        {
                for(i=0;i<n/2;i++)
                        b[i]=a[i];
                p=i;

                for(i=n/2,j=0;i<n;i++,j++)
                        c[j]=a[i];
                q=j;

                MergeSort(b,p);
                MergeSort(c,q);
                Merge(b,c,a,p,q);
        }
        return;
}
```

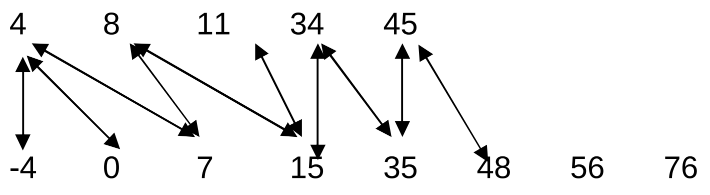
```
void main()
{
        int a[20],n,i;
        clrscr();
        printf("Enter the size of array:");
        scanf("%d",&n);
        printf("\nEnter elements:\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        MergeSort(a,n);

        printf("\nSorted list is:\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        getch();
}
```

NOTE:

When there are two sorted arrays, then also we can sort them using merge sort. This technique is called as merging the sorted array. The procedure is explained as below using an example –

Sorted Array (i)	:	4, 8, 11, 34, 45
Sorted Array (ii)	:	-4, 0, 7, 15, 35, 48, 56, 76

Procedure:



Result:

-4	0	4	7	8	11	15	34	35	45	48	56	76

The program for implementing merge sort when two sorted arrays are given, is as below.

Program:

```
#include<stdio.h>
#include<conio.h>
```

```
void MergeSort(int a[10], int b[10],int c[20],int m, int n)
{
        int i=0,j=0,k=0;

        while(i<m && j<n)
        {
                if(a[i]<b[j])
                {
                        c[k]=a[i];
                        i++;
                        k++;
                }
                else
                {
                        c[k]=b[j];
                        j++;
                        k++;
                }
        }
        while(i<m)
        {
                c[k]=a[i];
                i++;
                k++;
        }
        while(j<n)
        {
                c[k]=b[j];
                j++;
                k++;
        }
}

void main()
{
        int a[10],b[10],c[20],m,n,i;
        clrscr();
        printf("Enter the size of first array:");
        scanf("%d",&m);
        printf("\nEnter first array(in sorted order):\n");
        for(i=0;i<m;i++)
                scanf("%d",&a[i]);
        printf("Enter the size of second array:");
        scanf("%d",&n);
        printf("\nEnter second array(in sorted order):\n");
        for(i=0;i<n;i++)
                scanf("%d",&b[i]);
        MergeSort(a,b,c,m,n);

        printf("\nSorted list is:\n");
        for(i=0;i<m+n;i++)
                printf("%d\t",c[i]);
        getch();
}
```

Insertion Sort

This sorting technique involves inserting a particular element in proper position. In the first iteration, the second element is compared with the first. In second iteration, the third element is compared with second and then the first. Thus in every iteration, the element is compared with all the elements before it. If the element is found to be greater than any of its previous elements, then it is inserted at that position and all other elements are moved to one position towards right, to create the space for inserting element. The procedure is repeated till we get the sorted list.

Consider an example

```
25      12      30      8       7       43      32
```

I iteration

```
25      12      30      8       7       43      32      (12<25, so insert 12 at first position)

12      25      30      8       7       43      32
```

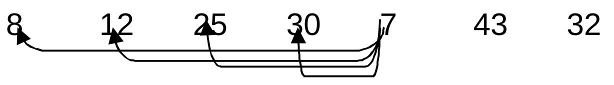
II iteration

```
12      25      30      8       7       43      32      (30>25, so don't compare 30 with 12)


12      25      30      8       7       43      32
```

III iteration

```
12      25      30      8       7       43      32      (8<30,25,12 so insert 8 at 1st position)
```

IV iteration


8 12 25 30 7 43 32 (7<30,25,12,8. So insert 7 at 1st pos)

V iteration

7 8 12 25 30 43 32 (43>30. So, don't compare 43 with other)

VI iteration

7 8 12 25 30 43 32 (32<43 but 32>30. So, insert in-between)

Sorted list:

7 8 12 25 30 32 43

Program:
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10],n,i,item,j;
        clrscr();
        printf("Enter the size of the array:");
        scanf("%d",&n);
        printf("\nEnter array elements:\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        for(i=1;i<n;i++)
        {
                item=a[i];
                for(j=i-1;j>=0 && item<a[j];j--)
                        a[j+1]=a[j];
                a[j+1]=item;
        }
        printf("\nSorted list is:\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        getch();
}
```