NAME:HARSHBHAI SOLANKI
ROLL-NO: 62
DIV:SE4-D

Experiment Number: 1

Aim: - Comparative analysis of Selection Sort and Insertion Sort.

Problem Statement:- Doing the comparative analysis of Selection sort & Insertion sort on the basis of comparisons required for sorting a list of large values of n.

Theory:-

Selection Sort

The selection sort works as follows. Initially the list is divided into two sublists, sorted and unsorted, which are divided by an imaginary wall. We find the smallest element from the unsorted sub list and swap it with the element at the beginning of the unsorted data. After each selection and swapping, the imaginary wall between the two sublists moves one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones. Each time we move one element from the unsorted sublist to the sorted sublist, we say that we have completed a sort pass. A list of n elements requires n-1 passes to completely rearrange the data.

Algorithm

```
n ← length[A]
for j ← 1 to n – 1
do smallest ← j
for i ← j + 1 to n
do if A[i] < A[smallest]
then smallest ← i
exchange A[j] ↔ A[smallest]
```

In Selection Sort for pass 1 there will be (n-1) comparison and as output only 1 element will get sorted. In second pass there will be (n-2) comparison and one more element will be sorted. So to sort n number of elements there will be (n-1) passes with (n-1), (n-2), (n-3)… 3, 2, 1 comparisons. Hence total number of comparisons is given as,

$$T(n) = (n - 1) + (n - 2) + (n - 3) + \dots + 3 + 2 + 1$$

$$\therefore T(n) = \sum_{j=1}^{n-1} j$$

$$\therefore T(n) = \frac{n(n-1)}{2} = O(n^2)$$

Insertion Sort

If the first few objects are already sorted, an unsorted object can be inserted in the sorted set in proper place. This is called insertion sort. An algorithm consider the elements one at a time, inserting each in its suitable place among those already considered (keeping them sorted). Insertion sort is an example of an **incremental** algorithm; it builds the sorted sequence one number at a time. This is perhaps the simplest example of the incremental insertion technique, where we build up a complicated structure on n items by first building it on n − 1 items and then making the necessary changes to fix things in adding the last item.

Algorithm	**Cost**	**Frequency**
for j ⍰ 2 to length [A]	c1	n
do Key ⍰ A[j]	c2	n-1
I ⍰ j-1	c3	n-1
while i > 0 and A[i] > Key	c4	$\sum_{j=2}^{n} t_j$
do A[i+1] ⍰ A[i]	c5	$\sum_{j=2}^{n} t_j - 1$
i⍰i-1	c6	$\sum_{j=2}^{n} t_j - 1$
A[i +1] ⍰ Key	c7	n-1

From above table we can write the equation for complexity of Selection Sort as follows:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} t_j - 1 + c_6 \sum_{j=2}^{n} t_j - 1 + c_7(n-1)$$

Now there will be three different cases for analysis in case of Insertion sort.

<u>Best Case:</u>

In case of best case the while loop will get executed (n-1) times whereas statements within the while will not be executed. Hence we can say that,

$$\sum_{j=2}^{n} t_j = (n-1)$$

$$\sum_{j=2}^{n} t_j - 1 = 0$$

Hence above equation can be re-written as follows,

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$\therefore T(n) = c_1 n + c_2 n + c_3 n + c_4 n + c_7 n - c_2 - c_3 - c_4 - c_5 - c_6 - c_7$$

$$\therefore T(n) = n\left(c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7\right) - \left(c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7\right)$$

$$\therefore T(n) = O(n) \qquad \ldots as\ \left(c_1, c_2, c_3, c_4, c_5, c_6, c_7\right)\ are\ constants.$$

<u>Worst Case and Average Case:</u>

In case of average case and worst case the while loop will get executed (n-1) times whereas statements within the while will not be executed. Hence we can say that,

$$\sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} + 1$$

$$\sum_{j=2}^{n} t_j - 1 = \frac{n(n-1)}{2}$$

Hence above equation can be re-written as follows,

$$T(n) = c_1 n + c_2(n - 1) + c_3(n - 1) + c_4\left(\frac{n(n+1)}{2} + 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n - 1)$$

$$\therefore T(n) = O\left(n^2\right)$$

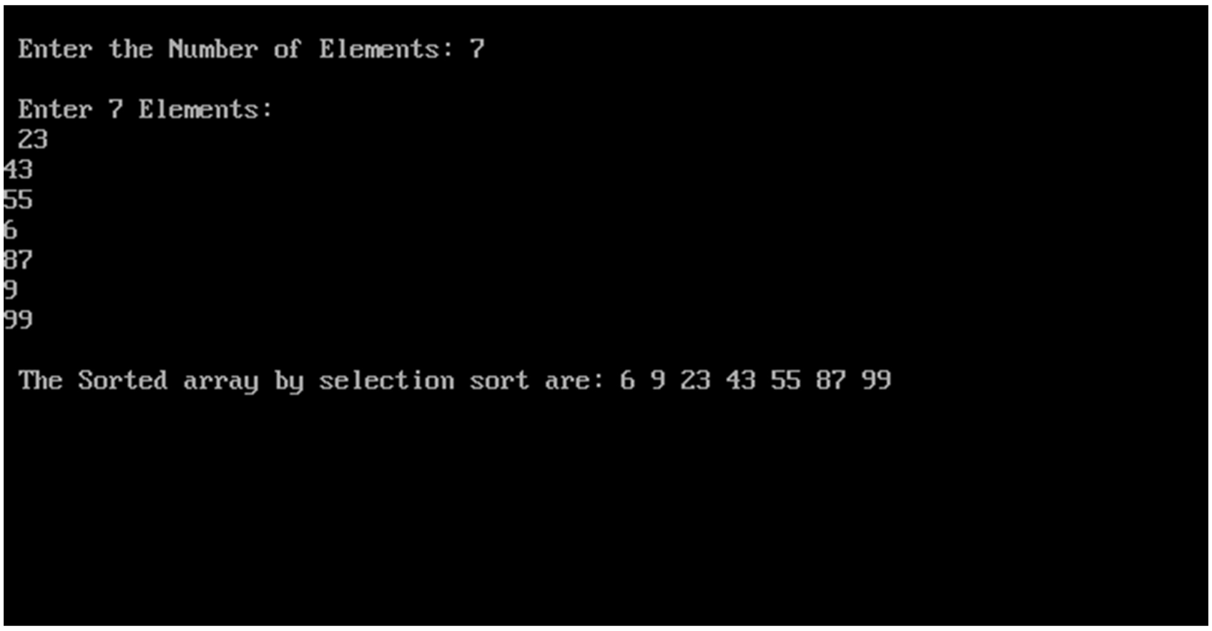
CODE:

SELECTION SORT:

```
#include<stdio.h>
#include<conio.h>
void selection(int a[],int n)
{
   int i,j,min,temp;
   for(i=0;i<n-1;i++)
   {
      min=i;
      for(j=i+1;j<n;j++)
      {
         if(a[min]>a[j])
         min=j;
      }
      if(min!=i)
      {
         temp=a[i];
         a[i]=a[min];
         a[min]=temp;
      }
   }
}
int main()
{
   int n,i;
   printf("\n Enter the Number of Elements: ");
```

```
    scanf("%d",&n);
    int a[n];
    printf("\n Enter %d Elements:\n ",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    selection(a,n);
    printf("\n The Sorted array by selection sort are: ");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```

OUTPUT:



```
 Enter the Number of Elements: 7

 Enter 7 Elements:
 23
43
55
6
87
9
99

 The Sorted array by selection sort are: 6 9 23 43 55 87 99
```

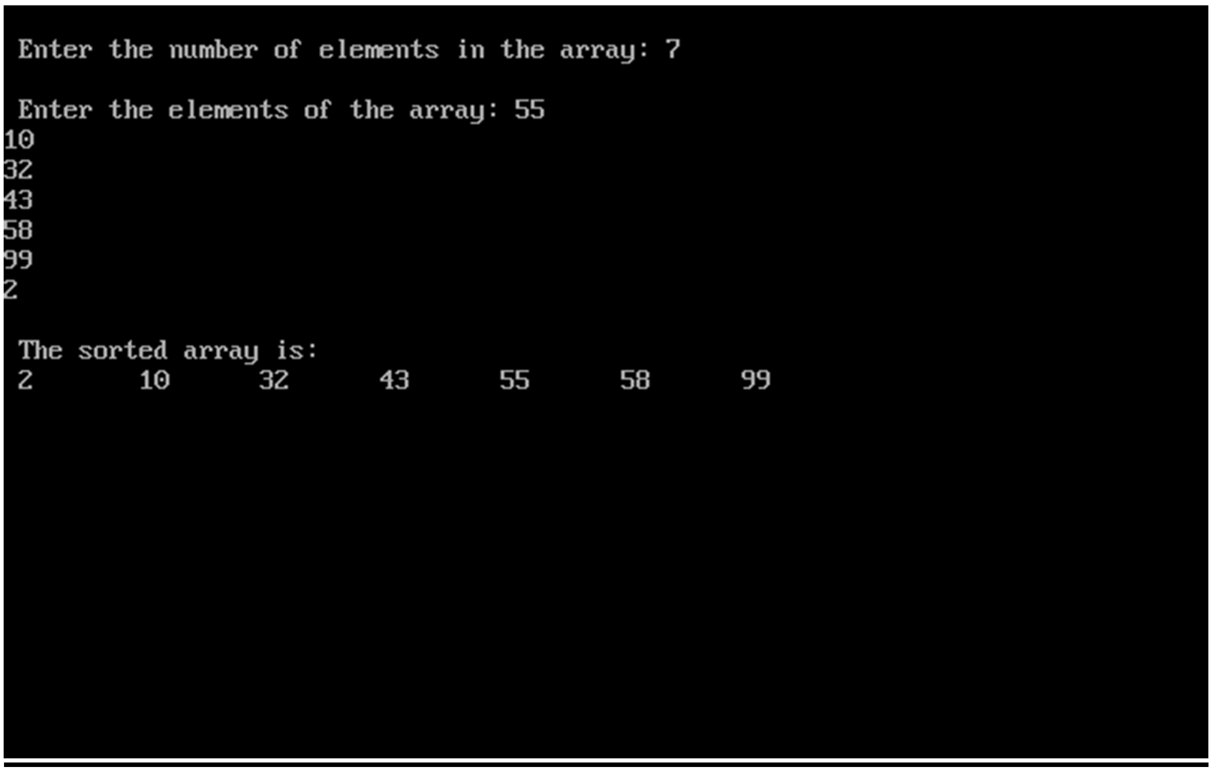
INSERTION SORT:

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

```
#include <conio.h>
void insertion_sort(int arr[], int n);
void main()
{
    int i, n;
    printf("\n Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("\n Enter the elements of the array: ");
    for(i=0;i<n;i++)
    {
        scanf("%d", &arr[i]);
    }
    insertion_sort(arr, n);
    printf("\n The sorted array is: \n");
    for(i=0;i<n;i++)
        printf(" %d\t", arr[i]);
    getch();
}
void insertion_sort(int arr[], int n)
{
    int i, j, temp;
    for(i=1;i<n;i++)
    {
         temp = arr[i];
         j = i-1;
         while((temp < arr[j]) && (j>=0))
```

```
            {
              arr[j+1] = arr[j];
              j--;
            }
            arr[j+1] = temp;
      }
}
```

OUTPUT:



```
 Enter the number of elements in the array: 7

 Enter the elements of the array: 55
10
32
43
58
99
2

 The sorted array is:
 2       10      32      43      55      58      99
```

CONCLUSION:

By performing above practical we can conclude that insertion sort is much more stable than selection sort. Insertion sort is a live sorting technique where the arriving elements are immediately sorted in the list whereas selection sort cannot work well with immediate data. For selection sort the best case for time complexity is Ω(n^2) and the average case is θ(n^2) and the worst case is O(n^2) .For insertion sort the best case for time complexity is Ω(n) and the average case is θ(n^2) and the worst case is O(n^2).Space complexity of selection sort is 1 and for space complexity insertion sort is also 1