PRACTICAL 2

AIM: Implement and analyse algorithms given below. (Compare them)

2.1 Bubble Sort
2.2 Selection Sort
2.3 Insertion Sort

SOFTWARE REQUIREMENT:

C++ Compiler, MS Word.

HARDWARE REQUIREMENT:

Desktop Computer.

Code:

```
#include<bits/stdc++.h>
using namespace std;
int count11=0;
void display(int *a,int n)
{
   int i;
   for(i=0;i<n;i++)
   {
      cout<<a[i]<<" ";
   }
}
int *BubbleSort(int *arr,int n)
{
   for(int i=0;i<n;i++){
      count11++;
      int swap=0;
      for(int j=0;j<n-1;j++){
         count11++;
         if(arr[j]>arr[j+1]){
            count11++;
            int temp=arr[j];count11++;
            arr[j]=arr[j+1];count11++;
            arr[j+1]=temp;count11++;
            swap++;count11++;
         }

      }
```

```
        if(swap==0){
                count11++;
                return arr;
            }
    }
    return arr;

}

void SelectionSort(int *a,int n)
{
    int i,j,minindex,temp,count1=0;
    for(i=0;i<=n-2;i++)
    {
        count1++;
            minindex=i;
            for(j=i+1;j<=n-1;j++)
            {count1++;
                if(a[j]<a[minindex])
                {
                    count1++;
                    minindex=j;count1++;
                }

            }
            if(minindex!=i)
            {
                count1++;
                temp=a[i];count1++;
                a[i]=a[minindex];count1++;
                a[minindex]=temp;count1++;
            }

    }
        cout<<"\nSorted array:\n";
        display(a,n);
        cout<<"\nNumber of Instructions are:";
        cout<<count1<<"\n\n";
}

void InsertionSort(int a[],int n)
{
    int i,j,key,count1=0;
    for(i=1;i<=n-1;i++)
    {count1++;
```

```
        key=a[i];count1++;
        j=i-1;count1++;
        while((j>=0) && (a[j]>key))
        {
           count1++;
           a[j+1]=a[j];count1++;
           j=j-1;count1++;
        }
        a[j+1]=key;count1++;

    }
    cout<<"\nSorted array:\n";
        display(a,n);

        cout<<"\nNumber of Instructions are:";
        cout<<count1<<"\n\n";
}

int main()
{
    int *array,choice=0,ch,n,i,*a;


    cout<<"Enter the number of elements:";
    cin>>n;
    array=new int(n);
    cout<<"Enter the elements:";
    for(i=0;i<n;i++)
    {
      cin>>array[i];
    }
    display(array,n);
    cout<<"\n---->Options<----\n";
    cout<<"\n1)Bubble Sort\n2)Selection Sort\n3)Insertion Sort\n4)Exit";

    cout<<"\nEnter your choice:";
    cin>>ch;
    choice=ch;
    switch(ch)
    {
       case 1:
          a=BubbleSort(array,n);
          cout<<"\nSorted array:\n";
          display(a,n);
          cout<<"\nNumber of Instructions are:";
```

```
        cout<<count11<<"\n\n";
        count11=0;
         break;
    case 2:
        SelectionSort(array,n);
         break;
    case 3:
        InsertionSort(array,n);
         break;
    case 4:
         exit(1);
    default:
          cout<<"Wrong choice";
          break;
  }

}
```

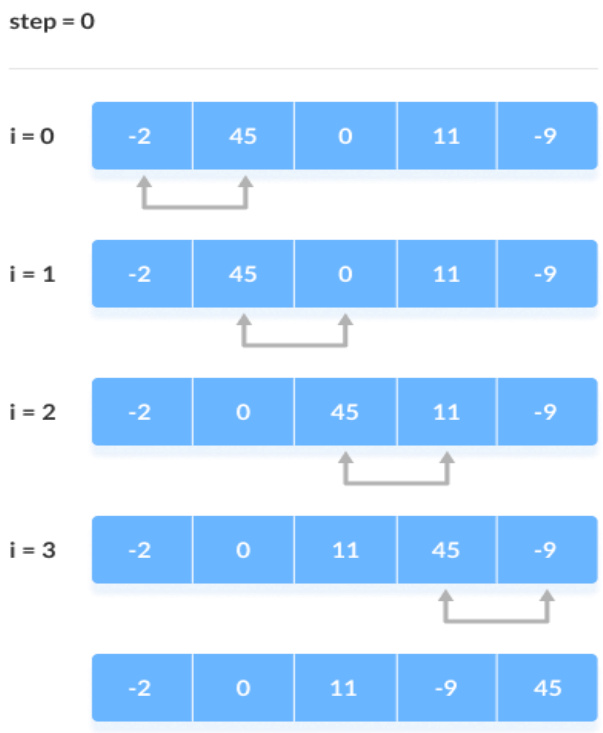
PRACTICAL 2.1

AIM:

Implement and analyse Bubble Sort.

THEORY:

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared, and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst-case complexity are of O(n2) where n is the number of items.



ALGORITHM:

Step 1: Repeat Step 2 For i = 0 to N-1.

Step 2: Repeat For J = i + 1 to N – I.

Step 3: IF A[J] > A[i]

SWAP A[J] and A[i]

[END OF INNER LOOP]

[END OF OUTER LOOP

Step 4: End.

OUTPUT:

1. **Best Case**

```
Enter the number of elements:5
Enter the elements:1 2 3 4 5
1 2 3 4 5
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:1

Sorted array:
1 2 3 4 5
Number of Instructions are:6
```

2. **Average Case**

```
Enter the number of elements:5
Enter the elements:2 4 1 0 3
2 4 1 0 3
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:1

Sorted array:
0 1 2 3 4
Number of Instructions are:51
```

3. **Worst Case**

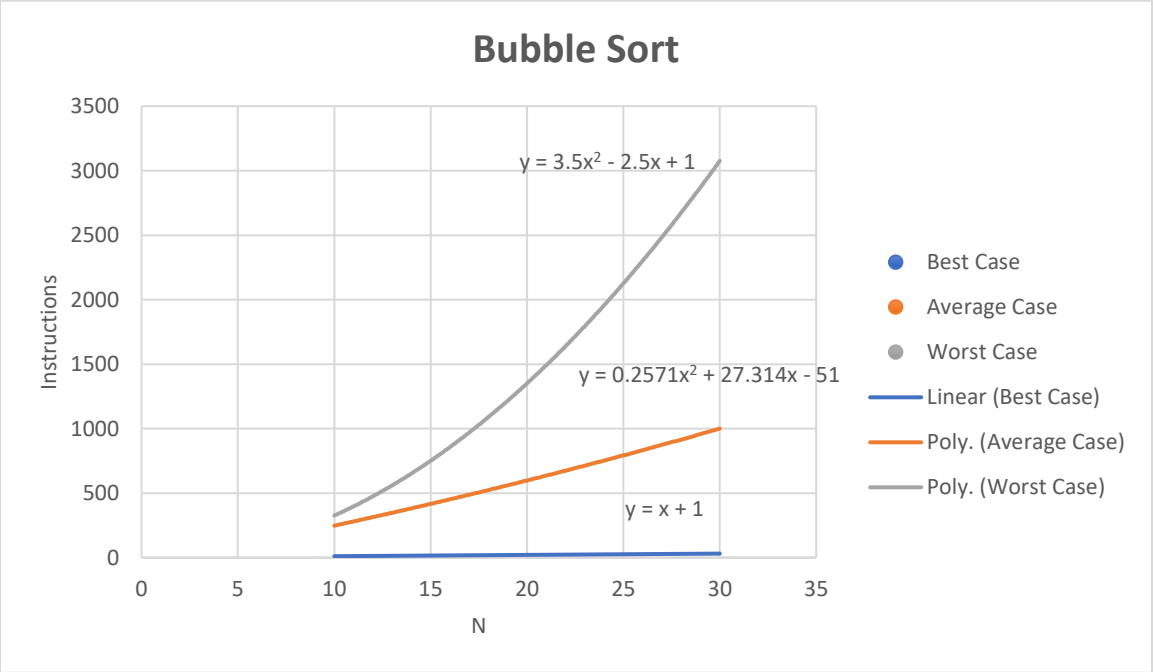
```
Enter the number of elements:5
Enter the elements:5 4 3 2 1
5 4 3 2 1
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:1

Sorted array:
1 2 3 4 5
Number of Instructions are:76
```

ANALYSIS OF ALGORITHM:

N	Best Case	Average Case	Worst Case
10	11	226	326
15	16	461	751
20	21	596	1351
25	26	751	2126
30	31	1021	3076



CONCLUSION

Performed bubble sort operation for different cases and found out the time complexities for each case. If all the elements are already sorted then it becomes the best case and if all elements are sorted in reverse order, then it becomes the worst case for this sorting algorithm

i. Best Case: O(n)
ii. Average Case: $O(n^2)$
iii. Worst Case: $O(n^2)$

PRACTICAL 2.2

AIM:

Implement and analyse Selection Sort.

THEORY:

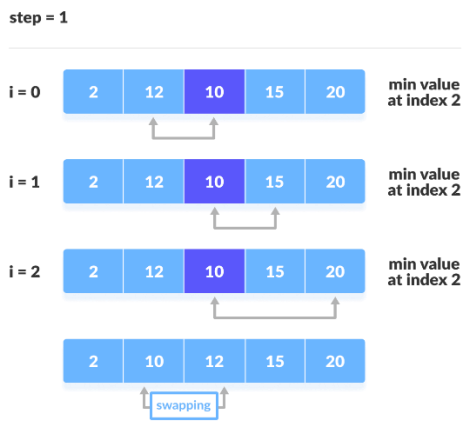
The code given from the mentors was of Selection Sort.

The **Selection Sort** algorithm sorts an array by repeatedly finding the minimum element (Considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted.

2. Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.



ALGORITHM:

Step 1: Repeat Steps 2 and 3 for K = 1 to N-1.

Step 2: CALL SMALLEST(ARR, K, N, POS).

Step 3: SWAP A[K] with ARR[POS]

[END OF LOOP]

Step 4: End.

OUTPUT:

1. **Best Case**

```
Enter the number of elements:5
Enter the elements:1 2 3 4 5
1 2 3 4 5
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:2

Sorted array:
1 2 3 4 5
Number of Instructions are:14
```

2. **Average Case**

```
Enter the number of elements:5
Enter the elements:2 3 1 4 5
2 3 1 4 5
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:2

Sorted array:
1 2 3 4 5
Number of Instructions are:26
```

3. **Worst Case**

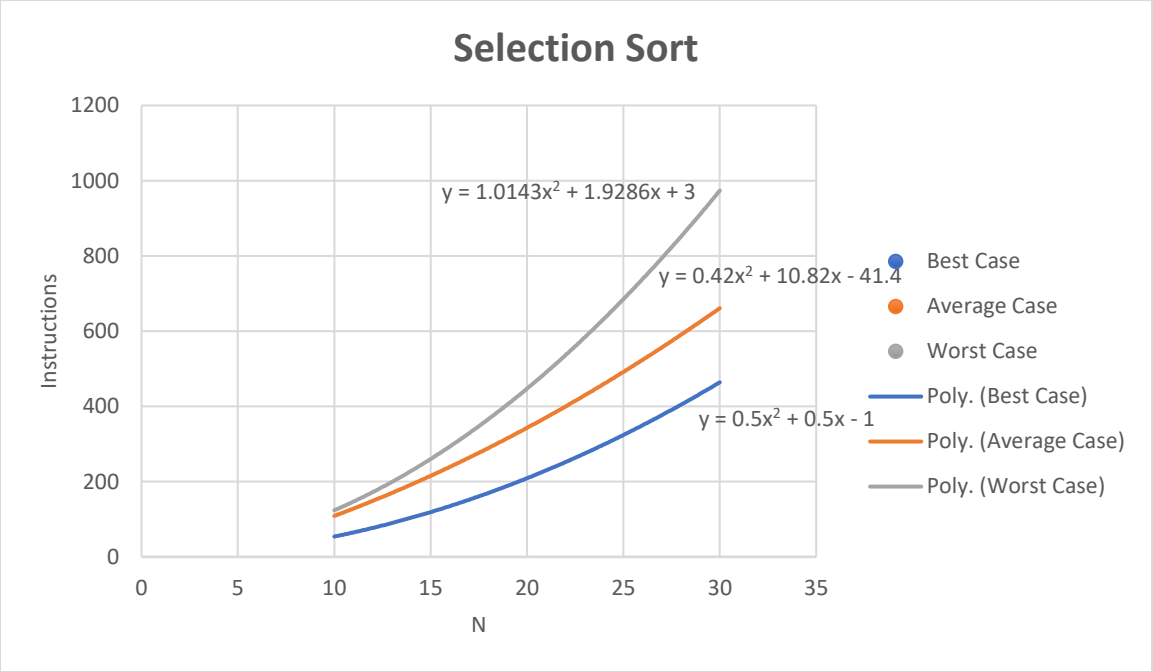
```
Enter the number of elements:5
Enter the elements:5 4 3 2 1
5 4 3 2 1
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:2

Sorted array:
1 2 3 4 5
Number of Instructions are:34
```

ANALYSIS OF ALGORITHM:

N	Best Case	Average Case	Worst Case
10	54	108	124
15	119	217	259
20	209	343	449
25	324	490	684
30	464	662	974



CONCLUSION

Performed selection sort operation for different cases and found out the time complexities for each case Here for selection sort in all the three cases, time complexity remains the same.

i. Best Case: $O(n^2)$
ii. Average Case: $O(n^2)$
iii. Worst Case: $O(n^2)$

PRACTICAL 2.3

AIM:

Implement and analyse Insertion Sort.

THEORY:

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially, and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst-case complexity are of O(n2), where n is the number of items.

ALGORITHM:

Step 1: Repeat Steps 2 to 5 for K = 1 to N-1.

Step 2: SET TEMP = ARR[K].

Step 3: SET J = K – 1.

Step 4: Repeat while TEMP <=ARR[J]

SET ARR[J + 1] = ARR[J]

SET J = J - 1

[END OF INNER LOOP]

Step 5: SET ARR[J + 1] = TEMP

[END OF LOOP]

Step 6: End.

OUTPUT:

1. **Best Case**

```
Enter the number of elements:5
Enter the elements:1 2 3 4 5
1 2 3 4 5
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:3

Sorted array:
1 2 3 4 5
Number of Instructions are:16
```

2. **Average Case**

```
Enter the number of elements:5
Enter the elements:2 4 3 1 5
2 4 3 1 5
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:3

Sorted array:
1 2 3 4 5
Number of Instructions are:28
```

3. **Worst Case**

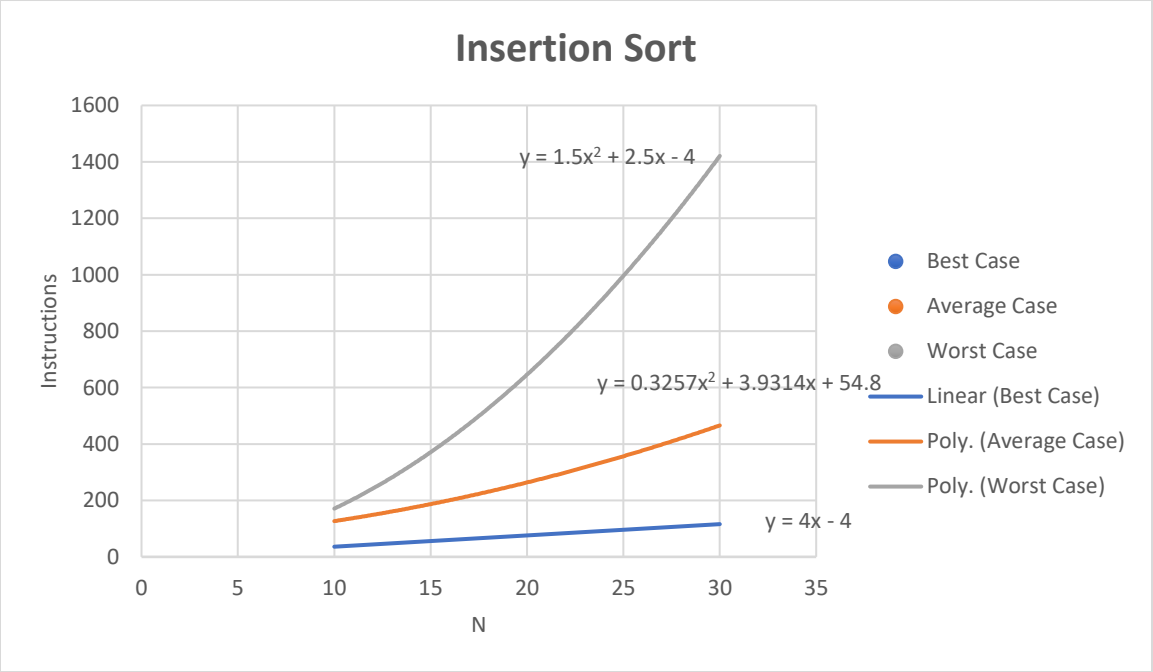
```
Enter the number of elements:5
Enter the elements:5 4 3 2 1
5 4 3 2 1
---->Options<----

1)Bubble Sort
2)Selection Sort
3)Insertion Sort
4)Exit
Enter your choice:3

Sorted array:
1 2 3 4 5
Number of Instructions are:46
```

ANALYSIS OF ALGORITHM:

N	Best Case	Average Case	Worst Case
10	36	117	171
15	56	206	371
20	76	265	646
25	96	336	996
30	116	476	1421



CONCLUSION

Performed bubble sort operation for different cases and found out the time complexities for each case. If all the elements are already sorted then it becomes the best case and if all elements are sorted in reverse order, then it becomes the worst case for this sorting algorithm.

i. Best Case: O(n)
ii. Average Case: $O(n^2)$
iii. Worst Case: $O(n^2)$

Final Conclusion:

Practical 1: Implement and analyse algorithms					
Sr. No	**Problem Definition**	**I/P Quality**	**Practical Time Complexity with Equation**		**Theoretical Complexity**
2.1	Bubble Sort	Best Case	$y = x + 1$ `	O(n) (Improvised Bubble Sort)	O(n) (Improvised Bubble Sort)
		Avg. Case	$y = 0.2571x^2 + 27.314x - 51$	$O(n^2)$	$O(n^2)$
		Worst Case	$y = 3.5x^2 - 2.5x + 1$	$O(n^2)$	$O(n^2)$
2.2	Selection Sort	Best Case	$y = 0.5x^2 + 0.5x - 1$ `	$O(n^2)$	$O(n^2)$
		Avg. Case	$y = 0.42x^2 + 10.82x - 41.4$	$O(n^2)$	$O(n^2)$
		Worst Case	$y = 1.0143x^2 + 1.9286x + 3$	$O(n^2)$	$O(n^2)$
2.3	Insertion Sort	Best Case	$y = 4x - 4$ `	O(n)	O(n)
		Avg. Case	$y = 0.3257x^2 + 3.9314x + 54.8$	$O(n^2)$	$O(n^2)$
		Worst Case	$y = 1.5x^2 + 2.5x - 4$	$O(n^2)$	$O(n^2)$