

## **Practical 2**

**Aim: - Implement and analyze algorithms given below (Compare them).**

2.1 Bubble Sort

2.2 Selection Sort

2.3 Insertion Sort

### **2.1 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.

#### **How does Bubble sort work??**

- Repeatedly compare neighbor pairs and swap if necessary.
- Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list.

#### **Algorithm:**

1. Last  $\leftarrow$  n
2. Repeat thru step 5 for pass: 1,2,3...,n-1
3. E  $\leftarrow$  0
4. Repeat for i=1,2,...,last-1
  - If  $k[i] > k[i+1]$
  - then  $k[i] \leftrightarrow k[i+1]$
  - E  $\leftarrow$  E+1
5. If E=0
  - then return
  - else
  - last  $\leftarrow$  last-1
6. Return number of passes

**Program:****Code:**

```
#include<iostream>

using namespace std;

int main()
{
    int *a,n,i,j,temp,c=0,p=0;

    cout<<"Enter the size of array: ";

    c++;

    cin>>n;

    c++;

    a=new int[n];

    c++;

    for(i=0;i<n;++i){

        c++;

        cin>>a[i];

        c++;

    }

    for(i=1;i<n;++i)

    {

        c++;

        for(j=0;j<(n-i);++j){

            c++;

            if(a[j]>a[j+1])

            {
```

```
        p++;
        c++;
        temp=a[j];
        c++;
        a[j]=a[j+1];
        c++;
        a[j+1]=temp;
        c++;
    }
}
if(p==0)
{
    break;
}
p=0;
}
cout<<"Array after bubble sort:";
c++;
for(i=0;i<n;++i){
    c++;
    cout<<" "<<a[i];
}
cout<<endl;
cout<<"and the value of count is"<<c;
```

```

    return 0;
}

```

## Output:-

"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\bubblesort.exe"

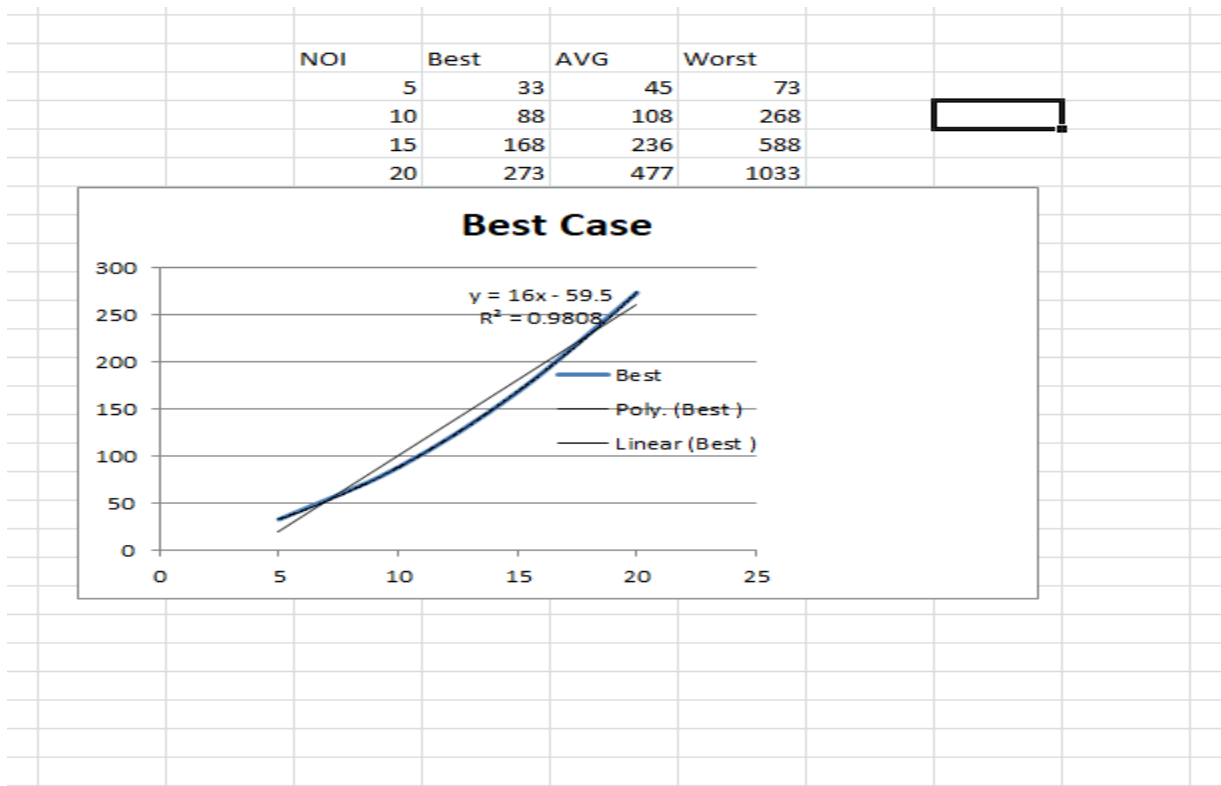
```

Enter the size of array: 5
12
2
45
3
14
Array after bubble sort: 2 3 12 14 45and the value of count is47

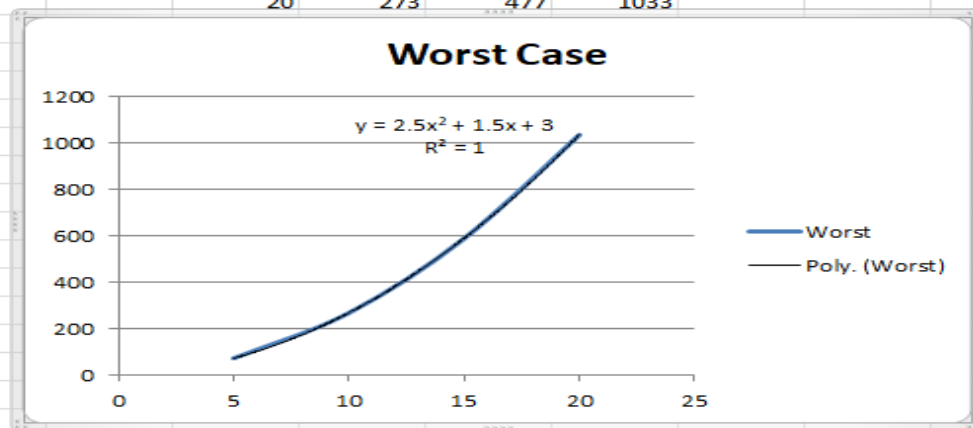
Process returned 0 (0x0)   execution time : 7.262 s
Press any key to continue.

```

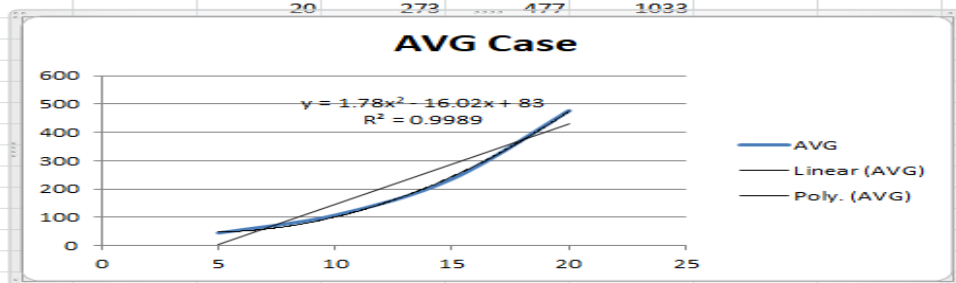
## Graph:-



NOI	Best	AVG	Worst
5	33	45	73
10	88	108	268
15	168	236	588
20	273	477	1033



NOI	Best	AVG	Worst
5	33	45	73
10	88	108	268
15	168	236	588
20	273	477	1033



### Complexity table: -

Algorithm	Best Case	Avg Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$

## 2.2 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.

### How does Insertion sort work??

- Insertion sort maintains a sorted sub-array, and repetitively inserts new elements into it.
- repeatedly add new element to the sorted result.
- 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 considers the elements one at a time, inserting each in its suitable place among those already considered. Insertion sort is an example of an incremental algorithm; it builds the sorted sequence one number at a time.

### Algorithm:

1. For  $j \leftarrow 2$  to  $n$   
do  $key \leftarrow A[j]$
2.  $i \leftarrow j-1$
3. while  $i > 0$  &  $A[i] > key$   
do  $A[i+1] \leftarrow A[i]$   
 $i \leftarrow i-1$
4.  $A[i+1] \leftarrow key$

**Program:****Code:**

```
#include<iostream>

using namespace std;

int c=0;

int main()
{
    int i,j,n,temp,a[30];

    c++;

    cout<<"Enter the number of elements:";

    c++;

    cin>>n;

    c++;

    cout<<"\nEnter the elements\n";

    c++;

    for(i=0;i<n;i++)
    {
        c++;

        cin>>a[i];

        c++;
    }

    for(i=1;i<=n-1;i++)
    {
        c++;

        temp=a[i];
```

```
c++;  
j=i-1;  
c++;  
while((temp<a[j])&&(j>=0))  
{  
    c++;  
    a[j+1]=a[j];  
    c++;  
    j=j-1;  
    c++;  
}  
a[j+1]=temp;  
}  
cout<<"\nSorted list is as follows\n";  
c++;  
for(i=0;i<n;i++)  
{  
    c++;  
    cout<<a[i]<<" ";  
    c++;  
}  
cout<<"and the count value is : "<<c;  
    return 0;  
}
```

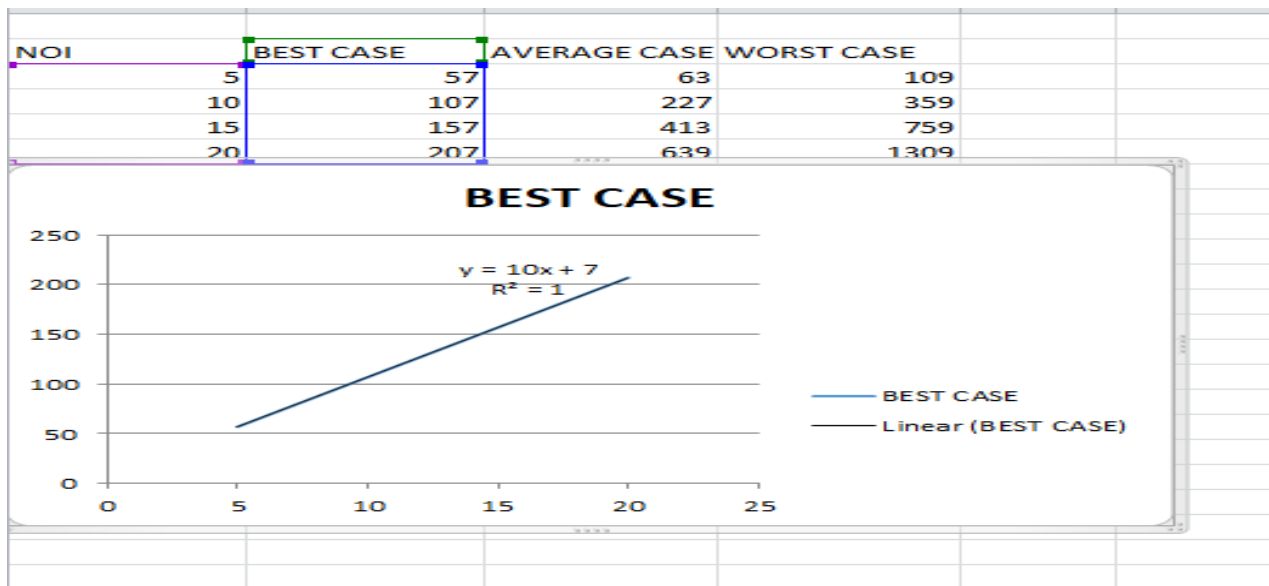


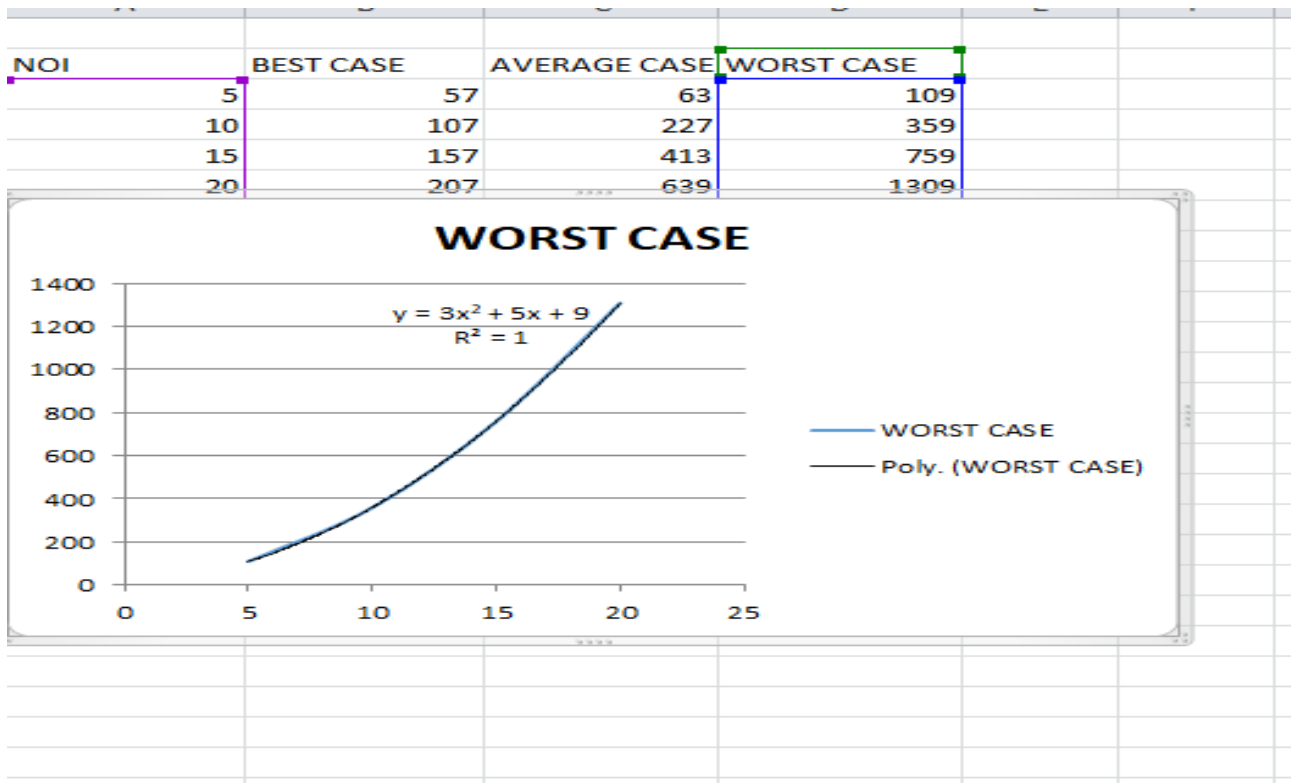
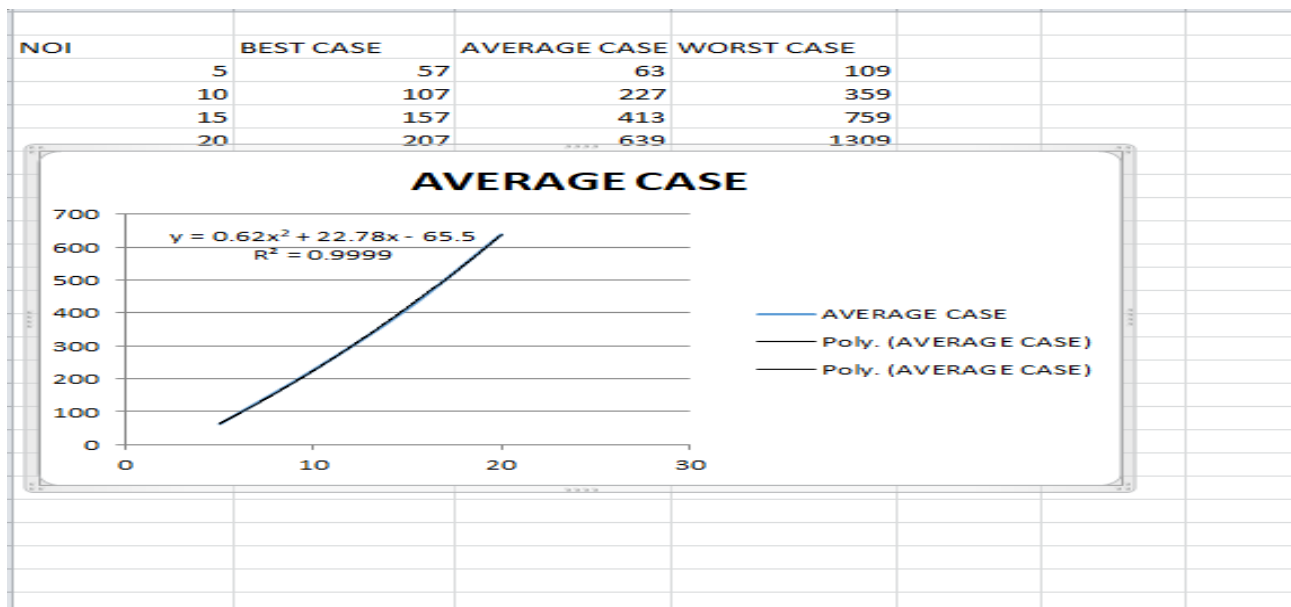
## Output:-

"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\Insertionsort.exe"

```
Enter the number of elements:5
Enter the elements
14
4
2
52
3
Sorted list is as follows
2 3 4 14 52 and the count value is :55
Process returned 0 (0x0)   execution time : 9.846 s
Press any key to continue.
```

## Graph: -





### Complexity table: -

Algorithm	Best Case	Avg Case	Worst Case
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$

## 2.3 Selection Sort: -

### Theory: -

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

### How does Insertion sort work??

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

### Algorithm:

1. Repeat thru step 4 for Pass=1,2,...,n-1
2. Min\_index  $\leftarrow$  Pass
3. Repeat for i=Pass+1, Pass+2... n  
    if  $k[i] < k[\text{Min\_index}]$  then Min\_index  $\leftarrow$  i
4. if Min\_index  $\neq$  Pass  
    then  $k[\text{Pass}] \leftrightarrow k[\text{Min\_index}]$
5. Return

**Program:****Code:**

```
#include<iostream>

using namespace std;

int main()
{
    int i,j,n,loc,temp,min,a[50],c=0;
    cout<<"Enter the number of elements:";

    c++;

    cin>>n;

    c++;

    cout<<"\nEnter the elements\n";
    c++;

    for(i=0;i<n;i++)
    {
        c++;

        cin>>a[i];

        c++;
    }

    for(i=0;i<n-1;i++)
    {
        c++;

        min=a[i];
```

```
    c++;
    loc=i;
    c++;
    for(j=i+1;j<n;j++)
    {
        c++;
        if(min>a[j])
        {
            c++;
            min=a[j];
        }
        c++;
        loc=j;
        c++;
    }
    temp=a[i];
    c++;
    a[i]=a[loc];
    c++;
    a[loc]=temp;
    c++;
}

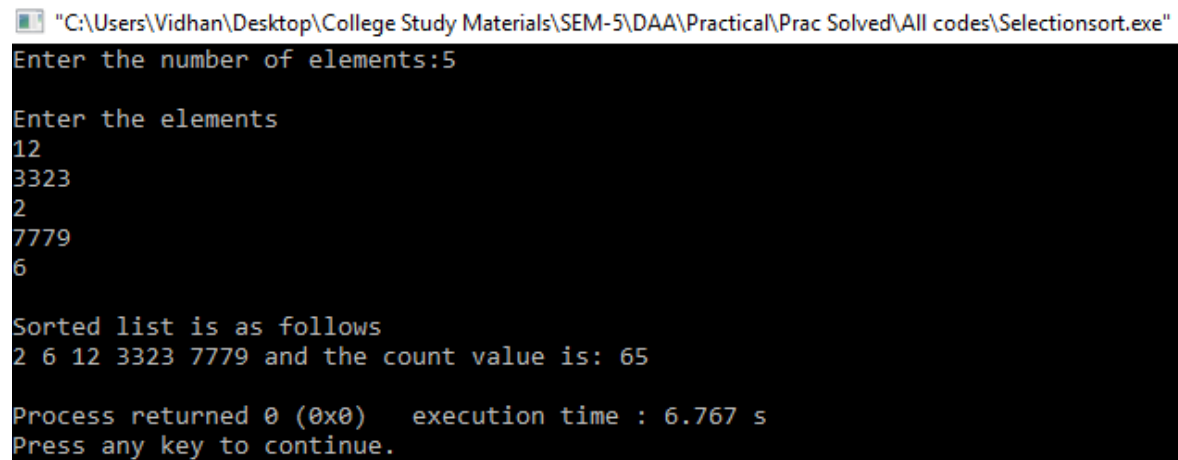
cout<<"\nSorted list is as follows\n";

c++;

for(i=0;i<n;i++)
```

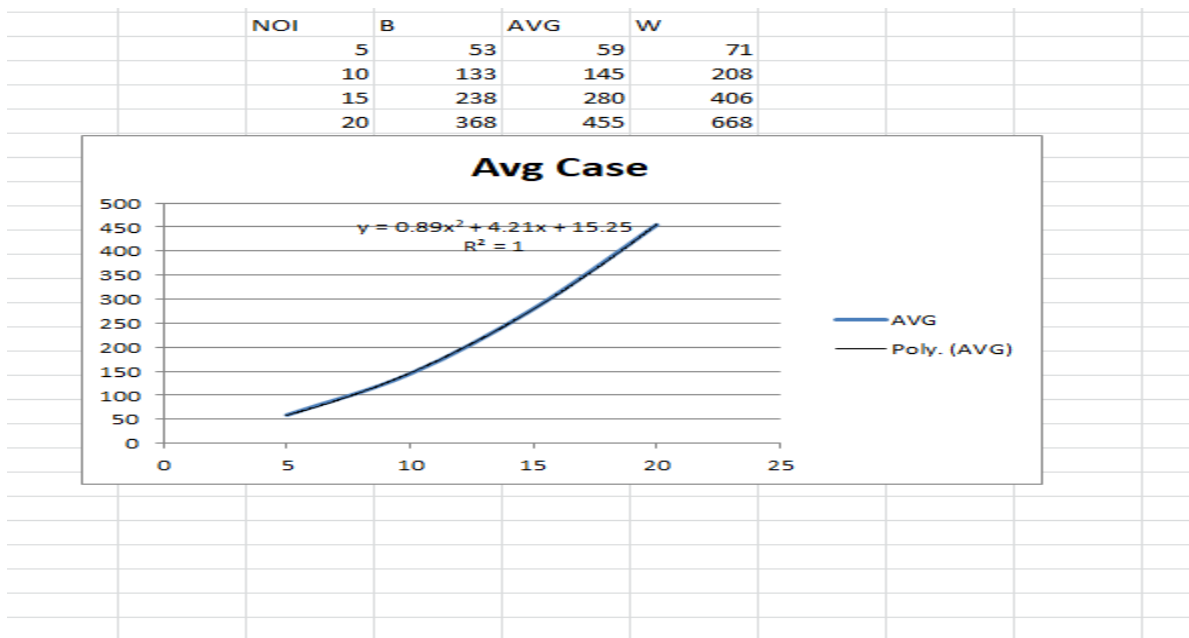
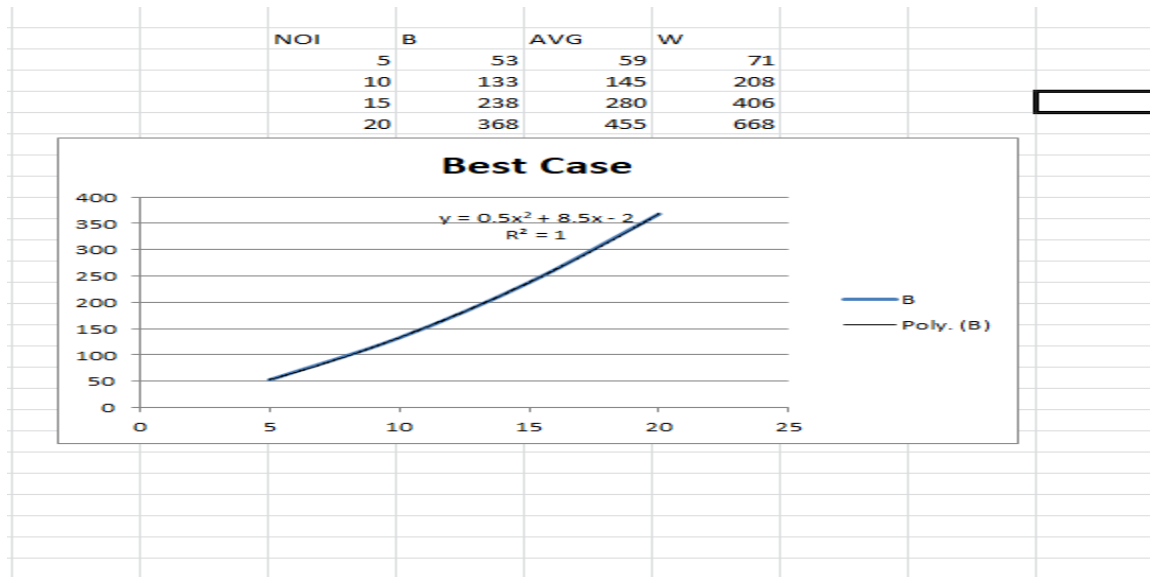
```
{  
    c++;  
    cout<<a[i]<<" ";  
}  
cout<<" "<<c;  
cout<<endl;  
return 0;  
}
```

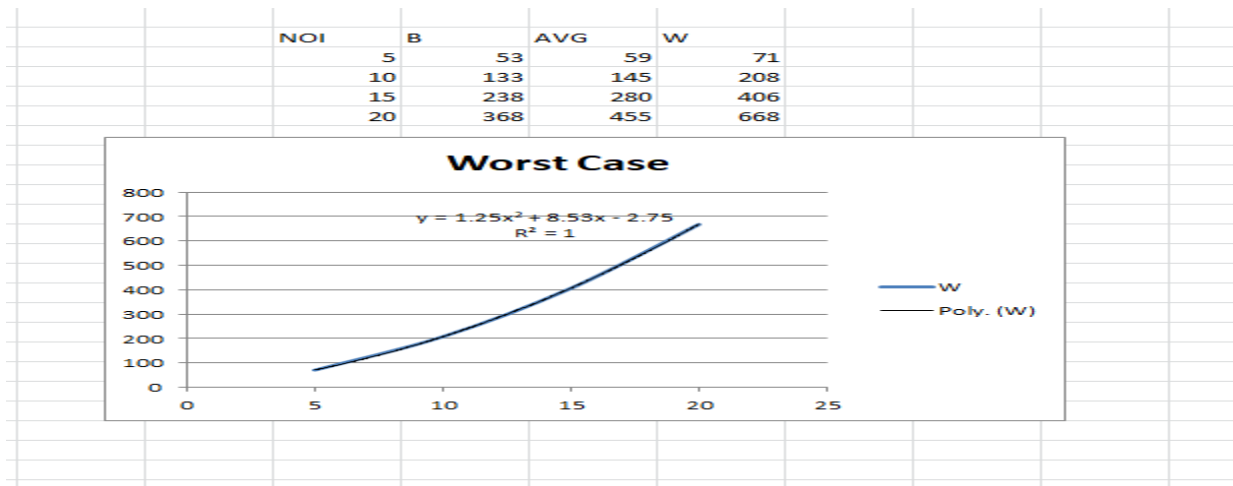
### Output:-



```
"C:\Users\Vidhan\Desktop\College Study Materials\SEM-5\DAA\Practical\Prac Solved\All codes\Selectionsort.exe"  
Enter the number of elements:5  
  
Enter the elements  
12  
3323  
2  
7779  
6  
  
Sorted list is as follows  
2 6 12 3323 7779 and the count value is: 65  
  
Process returned 0 (0x0)   execution time : 6.767 s  
Press any key to continue.
```

**Graph: -**





### Complexity table: -

Algorithm	Best Case	Avg Case	Worst Case
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

### Comparative Study of all 3 algorithms: - (Complexity Table)

Algorithm	Best Case	Avg Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$

### Conclusion:-

Bubble sort has time complexity of  $O(n^2)$  in its worst case. In case of an almost sorted small array, it's recommended to go for Insertion Sort rather than Bubble Sort. Best used when the data is small i.e. the list of elements to be sorted is low.

Insertion sort has time complexity of  $O(n^2)$  in its worst case. It is useful for adding new data to a presorted list. Also this is fast when the number of elements is small.

Selection sort has time complexity of  $O(n^2)$  in its worst case. Selection sort has number of swaps as  $O(n)$  whereas Insertion sort has  $O(n^2)$ .