|  |  |
| --- | --- |
| **Course Code:** CT2352 | **Course Name:** Lab - Design and Analysis of Algorithms |

|  |  |
| --- | --- |
| **Name of the Student:**  S Akshansh | **Sem/Sec:**  6th/A |
| **Branch:**  Computer Technology | **Roll no.:** 72  **Enrolment no.:** 19010927 |

# Practical 2

**Aim:** Consider the following array X, whose all elements are equal X = [1, 1, 1, 1, 1, 1]

Simulate both the insertion sort and selection sort algorithms on the array X. How does this compare to sorting the arrays U and V in previous practical?

# Theory:

**Insertion Sort:** Insertion sort is a simple sorting algorithm that works like the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

To sort an array of size n in ascending order:

1. Iterate from arr[1] to arr[n] over the array.
2. Compare the current element (key) to its predecessor.
3. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

**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 each 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.

**Time Complexity:** Time complexity of an algorithm signifies the total time required by the program to run till its completion.

The time complexity of algorithms is most commonly expressed using the big O notation. It's an asymptotic notation to represent the time complexity. We will study about it in detail in the next tutorial.

Time Complexity is most commonly estimated by counting the number of elementary steps performed by any algorithm to finish execution. Like in the example above, for the first code the loop will run n number of times, so the time complexity will be n at least and as the value of n will increase the time taken will also increase. While for the second code, time complexity is constant, because it will never be dependent on the value of n, it will always give the result in 1 step.

And since the algorithm's performance may vary with different types of input data, hence for an algorithm we usually use the worst-case Time complexity of an algorithm because that is the maximum time taken for any input size.

# Program (Insertion Sort):

#include<iostream> using namespace std;

int main()

{

cout<<"Name : S Akshansh"<<endl; cout<<"Roll No. : 72"<<endl;

cout<<"Reg. No. : 19010927"<<endl;

int n;

cout<<endl<<"Enter number of elements in array : "; cin>>n;

int a[n];

cout<<endl<<"Enter the elements in array"<<endl; for(int i=0; i<n ; i++)

cin>>a[i];

int inner\_counter=0, outer\_counter=0; for(int i=0; i<n; i++)

{

int key, j; key = a[i]; j = i-1;

while(j>=0 && a[j]>key)

{

a[j+1] = a[j]; j = j-1;

++inner\_counter;

}

a[j+1] = key;

++outer\_counter;

}

cout<<endl<<"Sorted Array : "; for(int i=0; i<6; i++)

cout<<a[i]<<" ";

cout<<endl<<"Inner Counter : "<<inner\_counter; cout<<endl<<"Outer Counter : "<<outer\_counter;

return 0;

}

**Program (Selection Sort):**

#include<iostream> using namespace std;

int main()

{

cout<<"Name : S Akshansh"<<endl; cout<<"Roll No. : 72"<<endl;

cout<<"Reg. No. : 19010927"<<endl;

int n;

cout<<endl<<"Enter number of elements in array : "; cin>>n;

int a[n];

cout<<endl<<"Enter the elements in array"<<endl; for(int i=0; i<n ; i++)

cin>>a[i];

int min, temp;

int inner\_counter=0, outer\_counter=0;

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

{

min=i;

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

{

if(a[j]<a[min])

{

min=j;

++inner\_counter;

}

}

temp = a[i]; a[i] = a[min]; a[min] = temp;

++outer\_counter;

}

cout<<endl<<"Sorted Array : "; for(int i=0; i<6; i++)

cout<<a[i]<<" ";

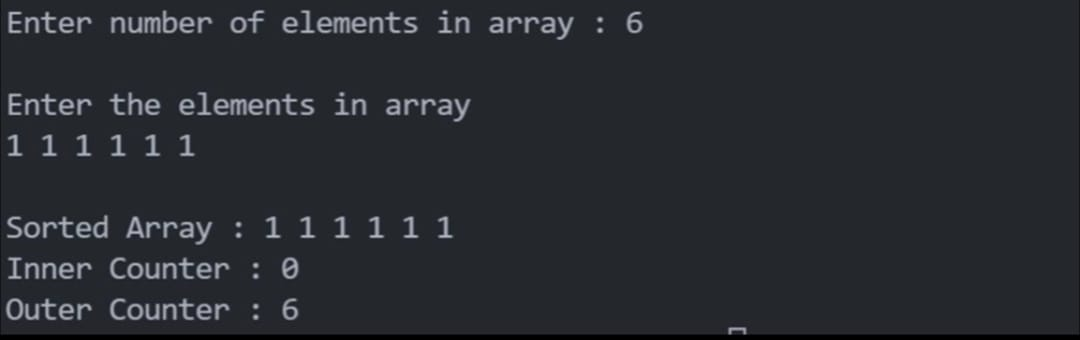
cout<<endl<<"Inner Counter : "<<inner\_counter; cout<<endl<<"Outer Counter : "<<outer\_counter;

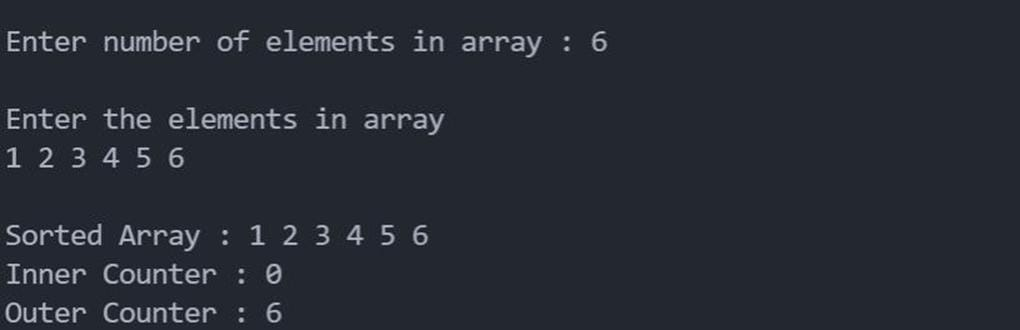
return 0;

}

# Output (Insertion Sort):

* **For array x[ ] = {1, 1, 1, 1, 1, 1}**





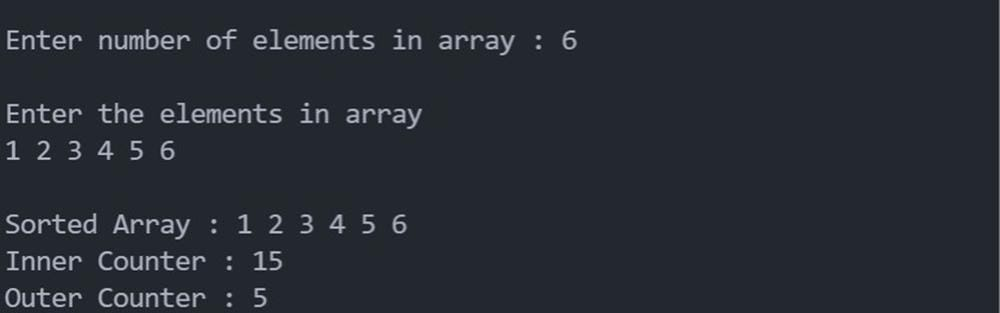
# For array v[ ] = {6, 5, 4, 3, 2, 1}

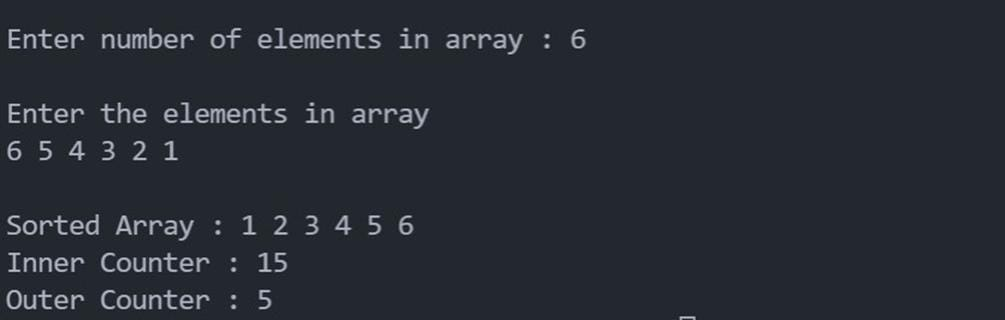
# 

**Output (Selection Sort):**

# For array x[ ] = {1, 1, 1, 1, 1, 1}

# 



* **For array v[ ] = {6, 5, 4, 3, 2, 1}**
* ****

# Answers:

1. Insertion sort runs faster than selection sort on array X which is already sorted
2. Insertion sort runs in same time for both arrays X and U as both are sorted already
3. Insertion sort runs faster on array X than array V as X is sorted already
4. Selection sort runs in same time for both arrays X and U as selection sort takes same time irrespective whether the array is sorted or not.
5. Selection sort runs in same time for both arrays X and V as selection sort takes same time irrespective whether the array is sorted or not.

Conclusion: – Hence, all programs of different problem statements were executed, and all questions are answered successfully.