EXPERIMENT NO- 1

AIM: Implementation and analysis of Selection Sort.

PROBLEM STATEMENT :

WAP to sort given numbers using Selection Sort algorithm.

Resource Required: Pentium IV, Turbo C, Printer, Printout Stationary

THEORY:

Selection Sort: -

This sorting is called "Selection Sort" because it works by repeatedly element. It works as follows: first find the smallest in the array and exchange it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continue in this way until the entire array is sorted.

Selection sort is among the simplest of sorting techniques and it work very well for small files. Furthermore, despite its evident "naïve approach "Selection sort has a quite important application because each item is actually moved at most once, Section sort is a method of choice for sorting files with very large objects (records) and small keys.

The worst case occurs if the array is already sorted in descending order. Nonetheless, the time require by selection sort algorithm is not very sensitive to the original order of the array to be sorted: the test "if *A*[*j*] < min x" is executed exactly the same number of times in every case. The variation in time is only due to the number of times the "then" part (i.e., min *j* ← *j*; min *x* ← *A*[*j*] of this test are executed.

The Selection sort spends most of its time trying to find the minimum element in the "unsorted" part of the array. Selection sort is quadratic in both the worst and the average case, and requires no extra memory.

Algorithm:

```
Start.
Define an array „A‟ of size „n‟.
Read array elements from the user.Define first element of the
array as „min‟i.e. min=A[0].
For i←0 to n-1
    1) min ←A[i]
    2) loc←I      //loc variable keeps the track of minimum element1
    3) for j← i+1 to n-1
        a. if(A[j]<min) then
        b. min ← A[j]
        c. loc  ← j
    4) if(loc!=i) then
    5) swap(A[i],A[loc])
Display sorted array
Stop.
```

CONCLUSION: The time complexity of selection sort is O(n^2) in Best, Worst and Average cases. The efficiency of selection sort does not depend upon the initial arrangement of the data. Selection sort is an in-place sort; that is, it requires no space other than that required by the input data (that is, the array) and a few variables whose number does not depend on the size of the set being sorted. Thus, its asymptotic space complexity is O(n).

Code:

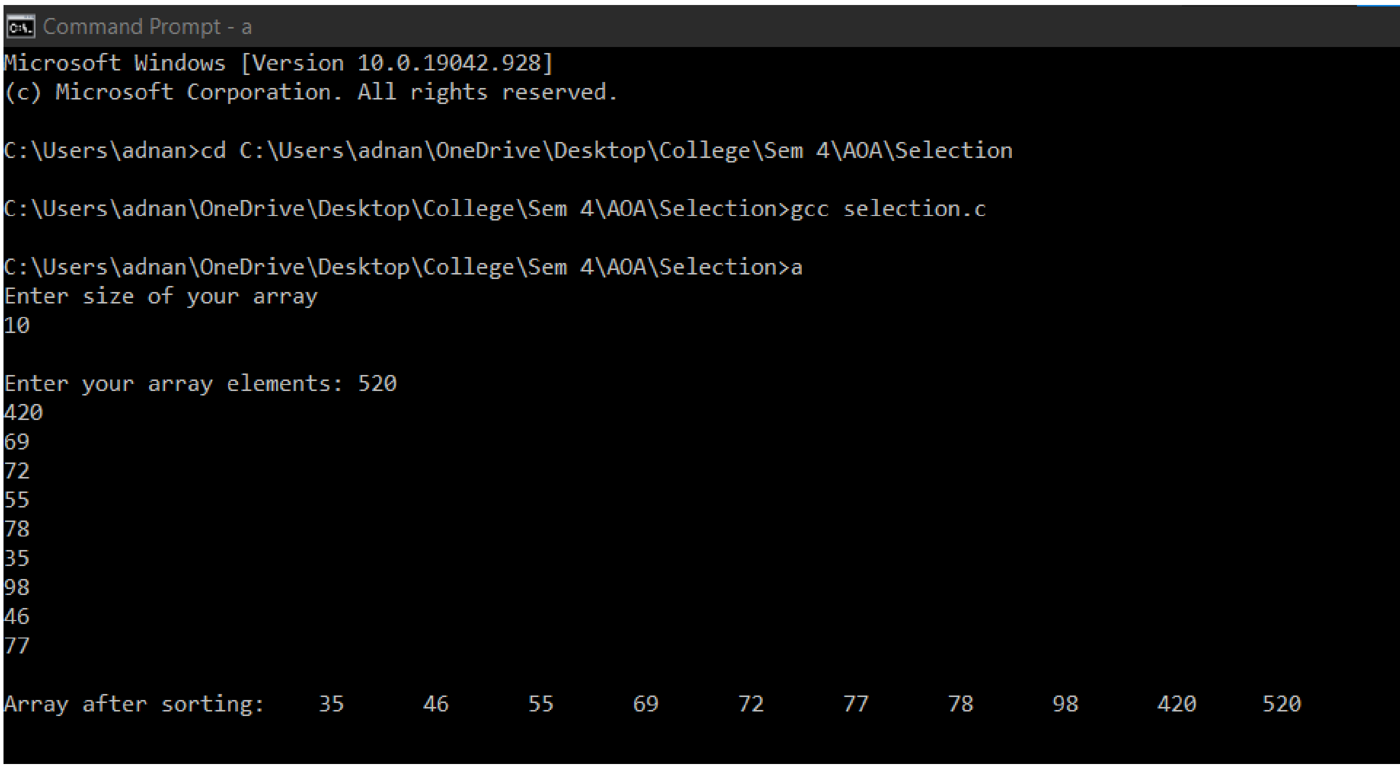
```
#include<stdio.h>
#include<conio.h>

void select(int arr[],int z);

int main()
{
    int z;
    printf("Enter size of your array \n");
    scanf("%d",&z);
    int arr[z];
    printf("\nEnter your array elements: ");
    for (int i = 0;i<z;i++)
    {
        scanf("%d",&arr[i]);
    }
    select(arr,z);
    printf("\nArray after sorting:");
    for (int i = 0;i<z;i++)
    {
        printf("\t%d",arr[i]);
    }
    getch();
    return 0;
}

void select(int arr[],int z)
{
    for(int i=0;i<z;i++)
    {
        int min=i;
        for(int j = i+1;j<z;j++)
        {
            if (arr[j]<arr[min])
                min=j;
        }
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}
```

Output:



Command Prompt - a

```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adnan>cd C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Selection

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Selection>gcc selection.c

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Selection>a
Enter size of your array
10

Enter your array elements: 520
420
69
72
55
78
35
98
46
77

Array after sorting:    35      46      55      69      72      77      78      98      420     520
```