

AOA EXPERIMENT 1

Name: Mohammed Arhaan Ali

Div: S11

Roll No: 02

AIM: To study and implement Selection sort and Insertion sort Algorithms

THEORY:

A) SELECTION SORT:

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. It is also the simplest algorithm. It is an in-place comparison sorting algorithm. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right.

In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

The average and worst-case complexity of selection sort is $O(n^2)$, where n is the number of items. Due to this, it is not suitable for large data sets.

Selection sort is generally used when -

- A small array is to be sorted
- Swapping cost doesn't matter
- It is compulsory to check all elements

Now, let's see the algorithm of selection sort.

Algorithm:

SELECTION SORT(arr, n)

Step 1: Repeat Steps 2 **and** 3 **for** $i = 0$ to $n-1$

Step 2: CALL SMALLEST(arr, i, n, pos)

Step 3: SWAP arr[i] with arr[pos]

[END OF LOOP]

Step 4: EXIT

SMALLEST (arr, i, n, pos)

Step 1: [INITIALIZE] SET SMALL = arr[i]

Step 2: [INITIALIZE] SET pos = i

Step 3: Repeat **for** $j = i+1$ to n

if (SMALL > arr[j])

 SET SMALL = arr[j]

SET pos = j

[END OF **if**]

[END OF LOOP]

Step 4: RETURN pos

1. Time Complexity

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity** - It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of selection sort is $O(n^2)$.
- **Average Case Complexity** - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of selection sort is $O(n^2)$.
- **Worst Case Complexity** - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of selection sort is $O(n^2)$.

2. Space Complexity

Space Complexity	$O(1)$
Stable	YES

- The space complexity of selection sort is $O(1)$. It is because, in selection sort, an extra variable is required for swapping.

Code:

```
#include<stdio.h>
#include<conio.h>
#define MAX 100

void swap(int *xp,int *yp){
    int temp=*xp;
    *xp=*yp;
    *yp=temp;
}

void selection_sort(int arr[],int n){
    int i,j,min;
    for(i=0;i<n;i++){
        min=i;
        for(j=i+1;j<n;j++){
            if(arr[j]<arr[min]){
                min=j;
            }
            if(arr[min]!=arr[i])
                swap(&arr[min],&arr[i]);
        }
    }
}

void print_array(int arr[],int n ){
    int i;
    for(i=0;i<n;i++){
        printf(" %d ",arr[i]);

    }
}

int main(){
    int i,n;
    int arr[MAX];
    printf("Enter the number of elements : ");
    scanf("%d",&n);
```

```

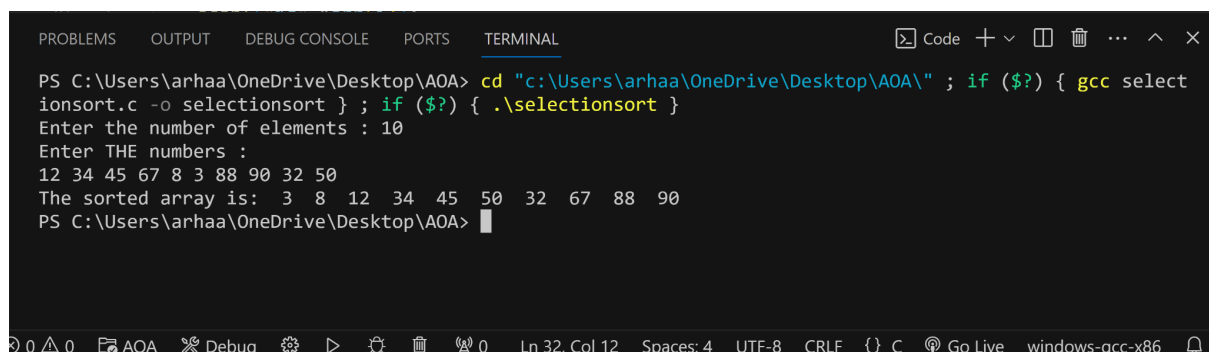
printf("Enter THE numbers :\n");
for(i=0;i<n;i++){
    scanf("%d",&arr[i]);
}

selection_sort(arr,n);
printf("The sorted array is: ");
print_array(arr,n);
return 0;

}

```

Output:



```

PS C:\Users\arhaa\OneDrive\Desktop\AOA> cd "c:\Users\arhaa\OneDrive\Desktop\AOA\" ; if ($?) { gcc select
ionsort.c -o selectionsort } ; if ($?) { .\selectionsort }
Enter the number of elements : 10
Enter THE numbers :
12 34 45 67 8 3 88 90 32 50
The sorted array is:  3  8 12 34 45 50 32 67 88 90
PS C:\Users\arhaa\OneDrive\Desktop\AOA>

```

B) INSERTION SORT:

THEORY:

Insertion sort works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is $O(n^2)$, where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

Insertion sort has various advantages such as -

- Simple implementation
- Efficient for small data sets
- Adaptive, i.e., it is appropriate for data sets that are already substantially sorted.

Now, let's see the algorithm of insertion sort.

Algorithm

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a **key**.

Step3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted

Insertion sort complexity:

Now, let's see the time complexity of insertion sort in best case, average case, and in worst case. We will also see the space complexity of insertion sort.

1. Time Complexity

Case	Time Complexity
Best Case	$O(n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity** - It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of insertion sort is **$O(n)$** .
- **Average Case Complexity** - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of insertion sort is **$O(n^2)$** .

- **Worst Case Complexity** - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of insertion sort is **$O(n^2)$** .

2. Space Complexity

Space Complexity $O(1)$

Stable YES

- The space complexity of insertion sort is $O(1)$. It is because, in insertion sort, an extra variable is required for swapping.

Code:

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

```
insertion_sort(int n, int arr[])
{
    int i, j, key;

    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
```



```

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }

    print_arr(n, arr);
}
print_arr(int n, int arr[])
{
    printf("Sorted array:");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
}

void main()
{
    int n;
    printf("Enter number of elements:");
    scanf("%d", &n);

    int arr[n];

    printf("Enter elements in array:");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    insertion_sort(n, arr);

    int a;
    printf("\nEnter new element to be insereted:");
    scanf("%d", &a);

```

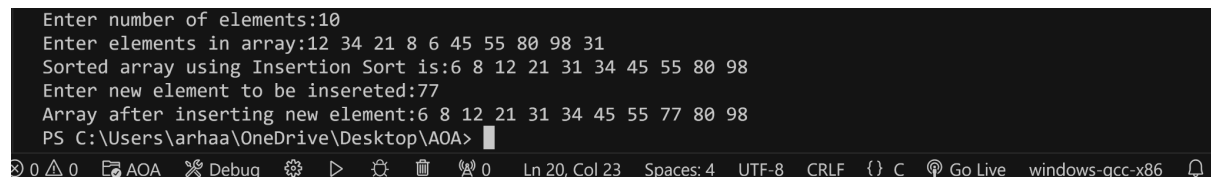
```

int index = n;
while (index >= 0 && arr[index-1] > a)
{
    arr[index] = arr[index-1];
    index--;
}
arr[index] = a;

printf("Array after inserting new element:");
for (int i = 0; i < n+1; i++)
{
    printf("%d ", arr[i]);
}
}

```

Output:



```

Enter number of elements:10
Enter elements in array:12 34 21 8 6 45 55 80 98 31
Sorted array using Insertion Sort is:6 8 12 21 31 34 45 55 80 98
Enter new element to be insereted:77
Array after inserting new element:6 8 12 21 31 34 45 55 77 80 98
PS C:\Users\arhaa\OneDrive\Desktop\AOA>

```

Conclusion:

Thus we have successfully implemented Selection Sort and Insertion Sort algorithms.