Experiment No. 1

Aim-: To study and implement Insertion and Selection sorting algorithms

Theory

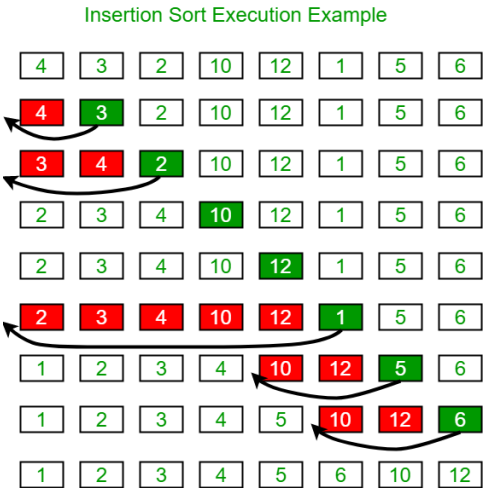
a) Insertion Sort

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 'insert'ed 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).

For example:

Suppose we have an unsorted elements like 4,3,2,10,12,1,5 & 6. The process for sorting all the elements in ascending order is given below



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.

This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where n is the number of items.

b) Selection Sort

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list. 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.

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.

Consider the following depicted array as an example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in a sorted manner.

The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process −

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
```

This algorithm is not suitable for large data sets as its average and worst case complexities are of O(n2), where n is the number of items

Conclusion :Thus we have successfully studied and implemented Insertion and selection Sorting algorithms