# Practical No 1

**Aim:** Sort a given set of elements using different sorting methods and determine the time required to sort the elements.

## 1. Insertion Sort

**Theory:**

The array is split virtually in the two parts in the insertion sort - An unsorted part and sorted part. The sorted part contains the first element of the array and other unsorted subpart contains the rest of the array. The first element in the unsorted array is compared to the sortedarray so that we can place it into a proper sub-array. It focuses on inserting the elements by moving all elements if the right-side value is smaller than the left side. It will repeatedly happenuntil the all element is inserted at correct place.

1. Iterate over the input elements by growing the sorted array at each iteration.

2. Compare the current element with the largest value available in the sorted array.

3. If the current element is greater, then it leaves the element in its place and moves on to the next element else it finds its correct position in the sorted array and moves it to that position inthe array.

4. This is achieved by shifting all the elements towards the right, which are larger than thecurrent element, in the sorted array to one position ahead.

**CODE :**

```
from time import *
def insertionSort(lists):
  for index in range(1,len(lists)):
    current_element = lists[index]
    position = index

    while current_element < lists[position-1] and position>0 :
      lists[position],lists[position-1]=lists[position-1],lists[position]
      position = position-1
    lists[position] = current_element

lists = [5,2,80,45,1]
print("Unsorted: ",lists)
start_time = time()
insertionSort(lists)
print("Sort : ",lists)
end_time = time()
execution_time = end_time - start_time
print(execution_time)
```

**OUTPUT :**

## 2. Bubble Sort

**THEORY :**

The bubble sort uses a straightforward logic that works by repeating swapping the adjacent elements if they are not in the right order. It compares one pair at a time and swaps if the firstelement is greater than the second element; otherwise, move further to the next pair of elements for comparison.

1. Starting with the first element(index = 0), compare the current element with the nextelement of the array.

2. If the current element is greater than the next element of the array, swap them.

3. If the current element is less than the next element, move to the next element.

**CODE :**

```
from time import *
lists = [10,25,2,3,1]
print("Unsorted List: ",lists)
start_t = time()
for j in range(len(lists)-1):
  for i in range(len(lists)-1):
    if lists[i]>lists[i+1]:
      lists[i],lists[i+1]=lists[i+1],lists[i]
print("Sorted list:",lists)
end_t = time()
execution_t = end_t - start_t
print(execution_t)
```

**OUTPUT :**

## 3. Selection Sort

**THEORY :**

In the selection sort algorithm, an array is sorted by recursively finding the minimum element from the unsorted part and inserting it at the beginning. Two subarrays are formed during theexecution of Selection sort on a given array.

1. The subarray, which is already sorted.

2. The subarray, which is unsorted.

During every iteration of selection sort, the minimum element from the unsorted subarray ispopped and inserted into the sorted subarray.

**CODE :**

```
from time import *
lists = [35,40,2,1,80,5]
print(lists)
start_t = time()
for i in range(len(lists)):
  min_value = min(lists[i:])
  min_index = lists.index(min_value)
  lists[i],lists[min_index] = lists[min_index],lists[i]
print(lists)
end_t = time()
execution_t = end_t - start_t
print(execution_t)
```

**OUTPUT :**

**4. Quick Sort**

**THEORY :**

1. An array is divided into subarrays by selecting a pivot element (element selected from the array). While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on theright side of the pivot.

2. The left and right subarrays are also divided using the same approach. This process continuesuntil each subarray contains a single element.

3. At this point, elements are already sorted. Finally, elements are combined to form a sortedarray.

**CODE :**

```
from time import *
def partition(array, low, high):
    pivot = array[high]
    i = low - 1

    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1

            (array[i], array[j]) = (array[j], array[i])

    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1
def quickSort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)


start_time = time()
data = [8, 14, 9, 6,1]
print("Original: ")
print(data)

size = len(data)
quickSort(data, 0, size - 1)
print(" Result: ")
print(data)

end_time = time()
execute_time = end_time - start_time
print(" Execution time is ", execute_time , " Seconds. ")
```

**OUTPUT :**

**Conclusion:**

| Sr. no. | Sorting Method | Array | Execution time |
|---|---|---|---|
| 1. | | | |
| 2. | | | |
| 3. | | | |
| 4. | | | |