Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Name	Akshat Biniwale
UID No.	2021300014
Subject	Design and Analysis of Algorithms
Class	Comps A
Experiment No.	2
AIM	Experiment on finding the running time of an algorithm.

THEORY:

1. Insertion Sort: It 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.

2. Quick sort: It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. 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.

ALGORITHM:

1. Insertion Sort:

    a. Start with the second element of the array.
    b. Compare this element with the previous one. If the previous one is larger, swap them.

c. Continue moving backwards through the array, comparing each element with the previous one and swapping them if necessary, until the element reaches its correct position.
d. Move to the next element and repeat the process until the end of the array is reached.
e. The final array will be sorted in ascending or descending order.

2. Selection Sort:

a. Find the minimum element in the unsorted portion of the array.
b. Swap the minimum element with the first element of the unsorted portion.
c. Move the boundary of the unsorted portion one step to the right.
d. Repeat the process until the entire array is sorted.

PROGRAM:

1. Insertion Sort:

```
void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}
```

2. Selection Sort:

```
void selectionSort(int arr[], int len){
   int minIndex, temp;
   for(int i=0; i<len; i++){
      minIndex = i;
      for(int j=i+1; j<len; j++){
         if(arr[j] < arr[minIndex]){
            minIndex = j;
         }
      }
      temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
   }
}
```

ANALYTICAL RESULT:

Insertion sort has a time complexity of O(n^2), meaning its performance degrades quickly as the size of the input grows. It performs well on small inputs and partially sorted data, but can become slow on large inputs. Selection sort also has a time complexity of O(n^2), similar to insertion sort.