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	**Naman Badlani**
UID No.	**2021300008**
Subject	**Design And Analysis of Algorithm**
Class	**Comps A**
Experiment No.	**2**
AIM	Experiment on finding the running time of an algorithm

Theory –

1. **Insertion sort–** It works like 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. **Selection 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 –

- Iterate from arr[1] to arr[N] over the array.
- Compare the current element (key) to its predecessor.

- If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

2. Selection Sort –

- Initialize minimum value(min_idx) to location 0.
- Traverse the array to find the minimum element in the array.
- While traversing if any element smaller than min_idx is found then swap both the values.
- Then, increment min_idx to point to the next element.
- Repeat until the array is sorted.

Programs –

1. Random Number Generator Code:

```c
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    FILE *fptr;
    fptr = fopen("Random.txt", "w");
    srand(time(NULL));
    int random;
    for(int i=1; i<=100000; i++){
        random = rand()%100000 + 1;
        fprintf(fptr, "%d ", random);
    }
    fclose(fptr);
    return 0;
}
```

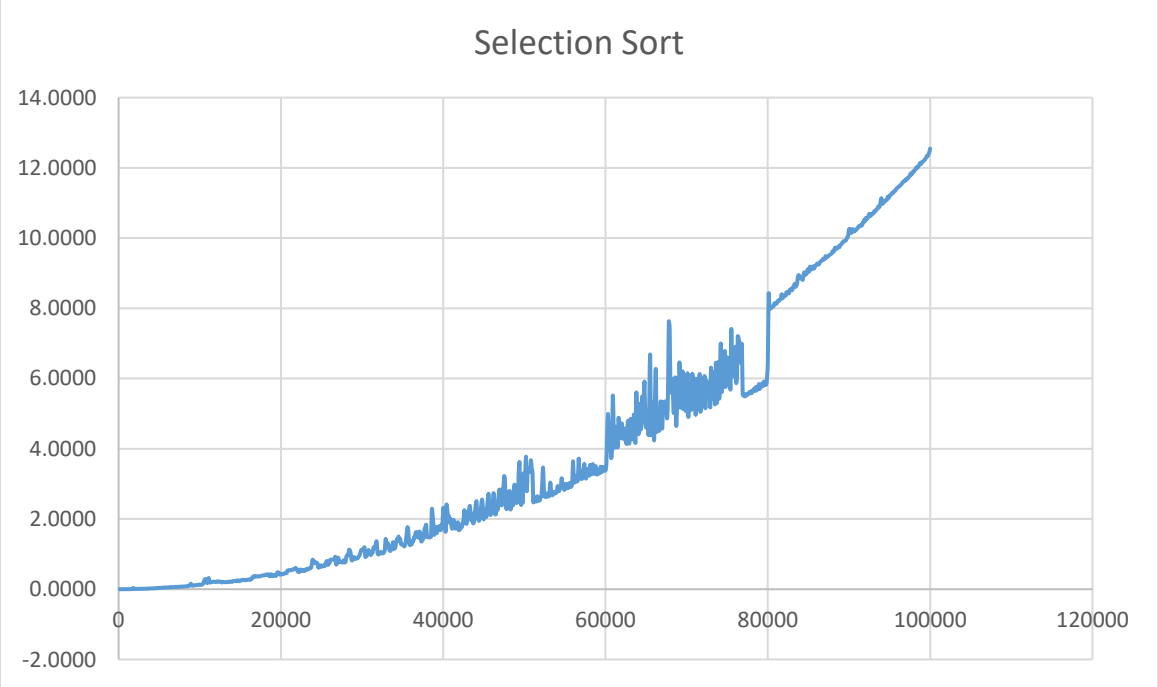
2. Selection & Insertion Sort Code:

```c
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
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;
    }
}
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;
    }
}
int main(){
    FILE *fptr, *sPtr;
    int index=99;
    int arrNums[100000];
    clock_t t;
    fptr = fopen("Random.txt", "r");
    sPtr = fopen("iTimes.txt", "w");
    for(int i=0; i<=999; i++){
```

```
        for(int j=0; j<=index; j++){
            fscanf(fptr, "%d", &arrNums[j]);
        }
        t = clock();
        insertionSort(arrNums, index+1);
        t = clock() - t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;
        fprintf(sPtr, "%lf\n", time_taken);
        printf("%d\t%lf\n", (i+1), time_taken);
        index = index + 100;
        fseek(fptr, 0, SEEK_SET);
    }
    fclose(sPtr);
    fclose(fptr);
    return 0;
}
```

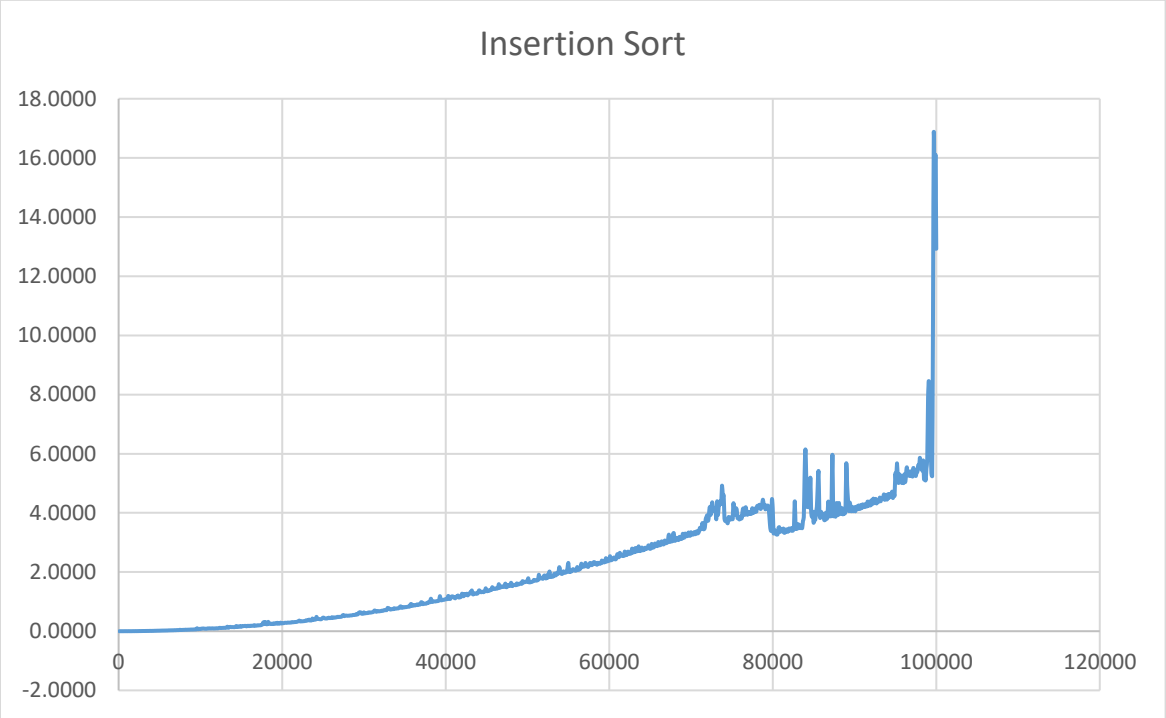
Result Analysis:

1. Selection Sort –



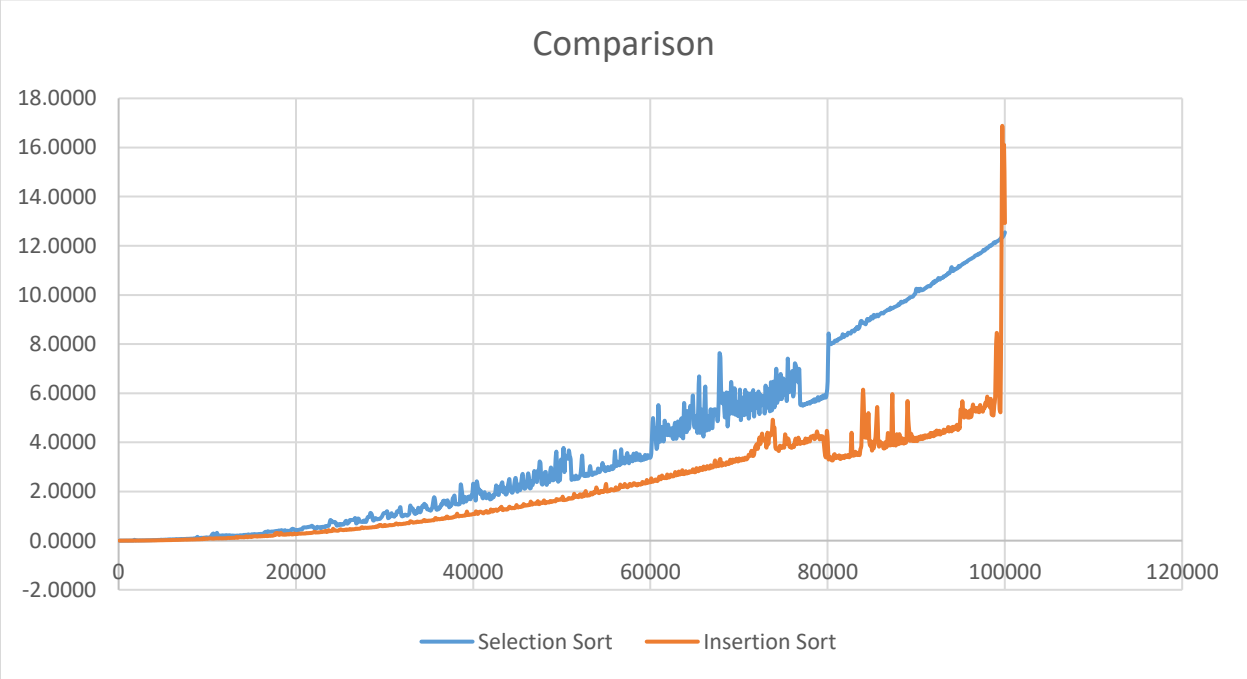
- The graph seen here is of selection sort done on a sample of 100000 numbers. We see that the time required for the iterations to take place has been increasing at great jumps.
- After the 80000 marks, the time increased exponentially, and it took 14secs to execute the last iteration.

2. Insertion Sort –



- The graph seen here is of Insertion Sort done on a sample of 100000 numbers. We see that time required for the iterations to take place has been taking small jumps at various timestamps.
- In the last 5 values, we see the spike to be very steep.

3. Comparison between the two sorting algorithms –



- Time complexity of Selection as well as Insertion Sort is O(n^2) but on observing the graph, we see that insertion sort works well and faster.
- The above scenario is because in Insertion Sort, the algorithm being complex is yet faster and efficient. While Selection Sort is simple but inefficient algorithm.