NAME:	NEHA NILESH GODE
EXPERIMENT:	1b
BATCH:	A3
DATE OF EXPERIMENT:	06/02/2023
DATE OF SUBMISSION:	12/02/2023

Aim: Experiment on finding the running time of the algorithms selection sort and insertion sort.

Theory:

Selection Sort:

Selection sort is a simple sorting algorithm. 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.

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

Algorithm for Selection Sort
Step 1 – Set MIN to location 0
Step 2 – Search the minimum element in the list
Step 3 – Swap with value at location MIN
Step 4 – Increment MIN to point to next element
Step 5 – Repeat until list is sorted

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 inserted 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). Its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

Algorithm for Insertion Sort

Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>


void dataInput() {
   //generate 100000 random numbers
   for (int i=0;i<100000; i++)
   {
      int temp = rand();
      FILE *fptr;
      fptr = fopen("Numgenerated.txt", "a");
      fprintf(fptr, "%d\n", temp);
      fclose(fptr);
  }
}

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

//selection sort algorithm
void selectionSort(long arr[], int n) {
   int i, j, min_idx;
   for (i = 0; i < n-1; i++) {
```

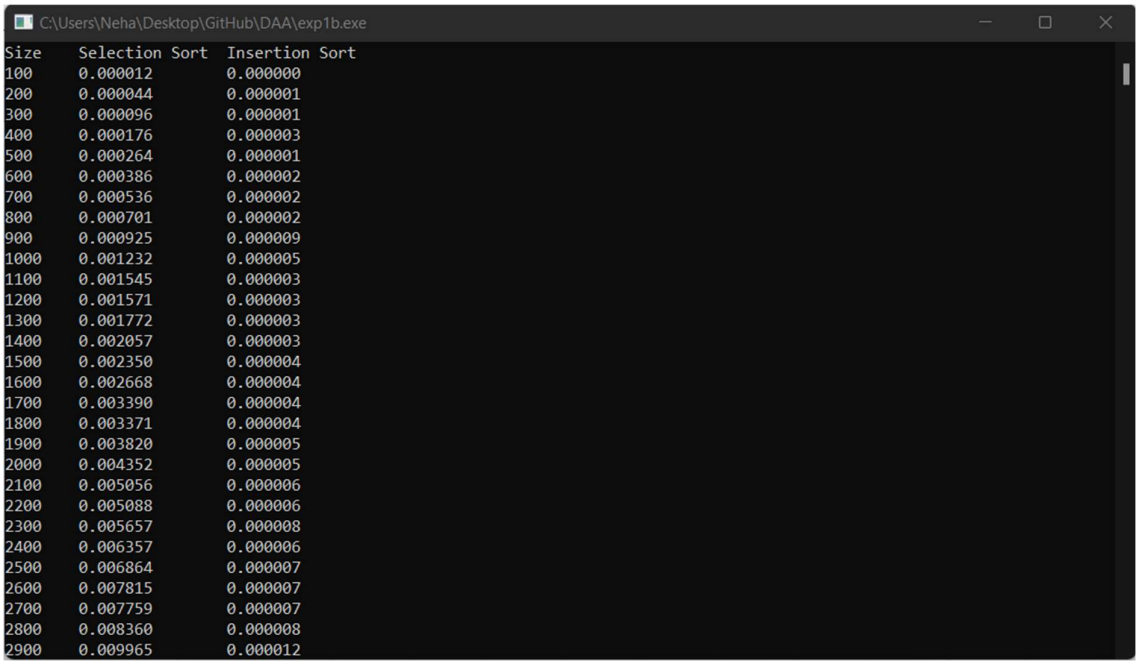
```
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        swap(&arr[min_idx], &arr[i]);
    }
}

//insertion sort algorithm
void insertionSort(long arr[], int n) {
    int i, key, j;
    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;
    }
}

int main() {
    dataInput();
    FILE *fptr;
    fptr = fopen("Numgenerated.txt", "r");
    long arr[100000], arr1[100000], arr2[100000];
    for (int i = 0; i < 100000; i++)
    {
        fscanf(fptr, "%8ld", &arr[i]);
    }
    fclose(fptr);
    int s = 100;
    printf("Size\tSelection Sort\tInsertion Sort\n");
    for(int i=0;i<=1000;i++) {
        for (int j = 0; j < 100000; j++) {
            arr1[j] = arr[j];
            arr2[j] = arr[j];
```

```
        }
        double diff1, diff2;
        struct timespec start, end;
        int i;
        clock_gettime(CLOCK_MONOTONIC, &start);
        selectionSort(arr1, s);
        clock_gettime(CLOCK_MONOTONIC, &end);
        diff1 = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec);
        clock_gettime(CLOCK_MONOTONIC, &start);
        insertionSort(arr2, s);
        clock_gettime(CLOCK_MONOTONIC, &end);
        diff2 = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec);
        printf("%d\t%f\t%f\n", s, diff1, diff2);
        s+=100;
    }
    return 0;
}
```

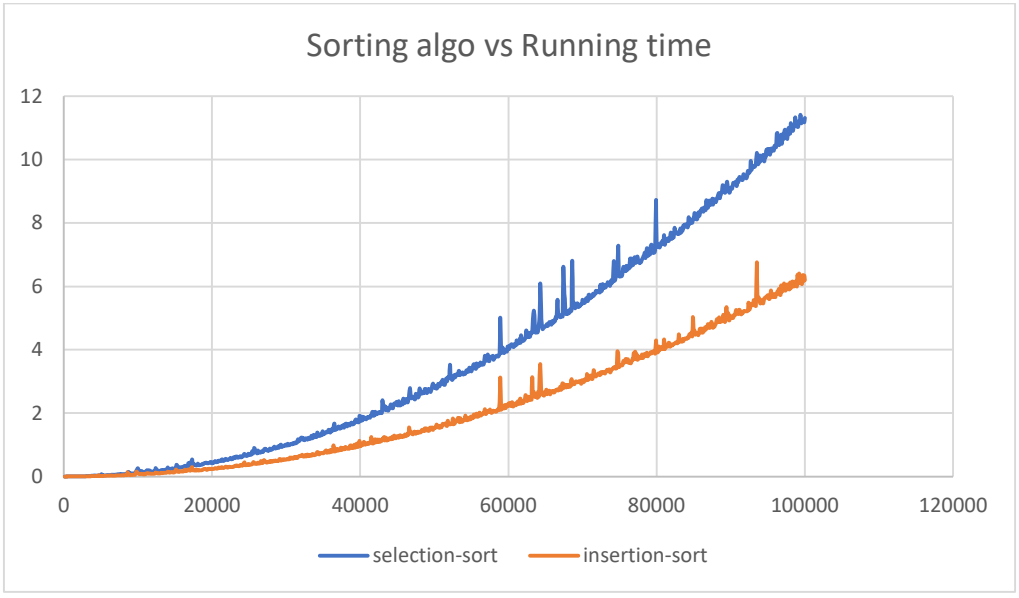
Output:



```
C:\Users\Neha\Desktop\GitHub\DAA\exp1b.exe
97600   15.901196       0.000369
97700   15.493696       0.000310
97800   15.290939       0.002102
97900   16.759025       0.003475
98000   16.445788       0.006344
98100   15.713695       0.004379
98200   17.077493       0.007205
98300   15.125437       0.007981
98400   15.618689       0.014344
98500   15.795432       0.009198
98600   17.063972       0.014483
98700   15.947403       0.009842
98800   16.351042       0.012385
98900   17.133259       0.023129
99000   16.166605       0.016086
99100   16.392617       0.016007
99200   17.152228       0.022485
99300   15.927627       0.018982
99400   16.306407       0.036748
99500   15.638367       0.027780
99600   16.695250       0.043523
99700   15.842777       0.026087
99800   15.514762       0.055853
99900   25.728001       0.041044
100000  16.277016       0.032407
100100  17.222078       0.088298

Process returned 0 (0x0)   execution time : 11885.034 s
Press any key to continue.
```

Graphs:



Observation:

For the initial lower input numbers, both selection sort and insertion sort requires equal amount of running time. After a particular value of the number of inputs, insertion sort has a lesser running time than selection sort.

Conclusion:
Insertion sort is more feasible for higher number of inputs as it gives output with less running time.