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 an algorithm(selection sort and insertion sort).

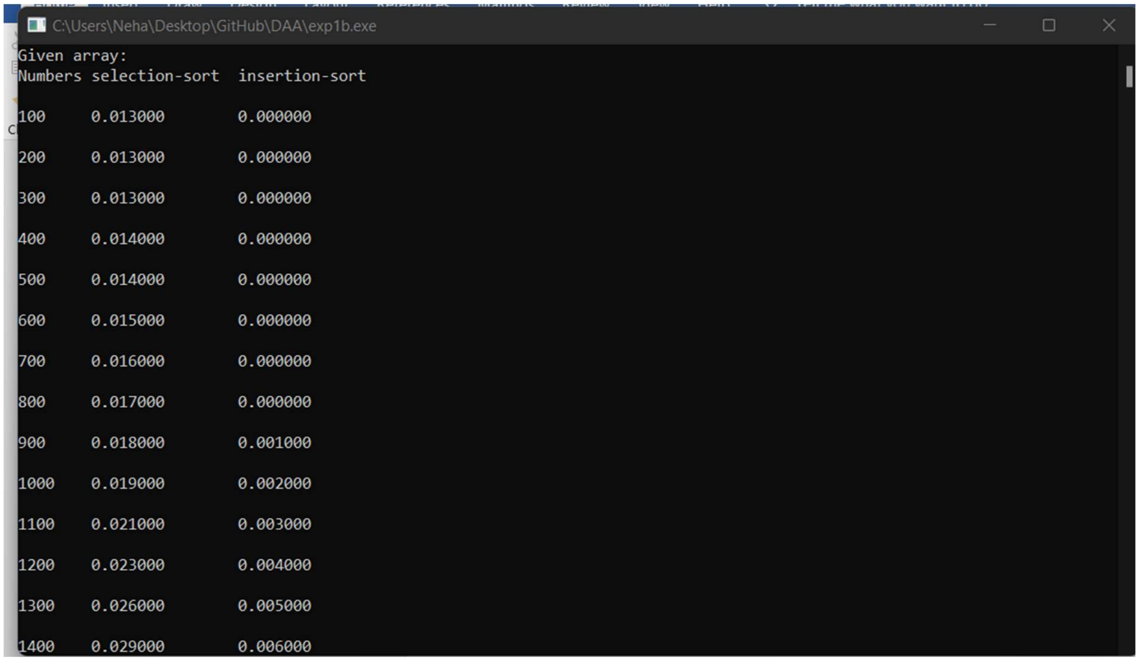
Code:

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

void selectSort(int arr[], int n)
{
   int min, temp;
   int index;
   for (int i = 0; i < n - 1; i++)
   {
      min = arr[i];
      index = i;
      for (int j = i + 1; j < n; j++)
      {
         if (arr[j] < min)
         {
            min = arr[j];
            index = j;
         }
      }
      temp = arr[i];
      arr[i] = min;
      arr[index] = temp;
   }
}

void insertSort(int arr[], int n)
{
   int temp;
   for (int i = 0; i < n - 1; i++)
```

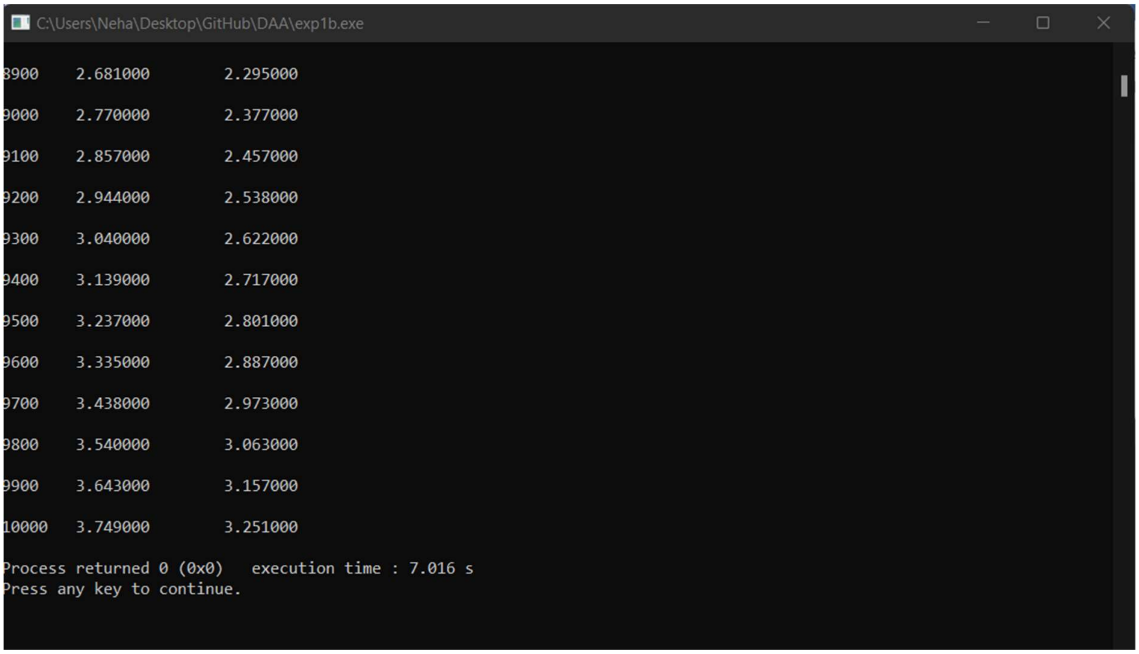
```
    {
        for (int j = i + 1; j < n - 1 - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main()
{
    clock_t t;
    t = clock();
    double time_taken1=0.0,time_taken2=0.0;

    FILE *fp;
    fp = fopen("Numbergenerated.txt", "w");
    int arr[10000];
    for (int i = 0; i < 100000; i++)
    {
        fprintf(fp,"%d ", rand()%10);
    }
    fclose(fp);
    FILE *fptr;
    fptr = fopen("Numbergenerated.txt", "r");
    printf("Given array: ");
    for (int i = 0; i < 10000; i++)
    {
        arr[i]=getw(fptr);
    }
    printf("\nNumbers\tselection-sort\tinsertion-sort\n");

    for(int i=100;i<=10000;i=i+100)
    {
    printf("\n");
    selectSort(arr, i);
```

```
   t = clock() - t;
   time_taken1 = ((double)t)/CLOCKS_PER_SEC; // in seconds
   insertSort(arr, i);
   t = clock() - t;
   time_taken2 = ((double)t)/CLOCKS_PER_SEC; // in seconds
   printf("%d\t%f\t%f \n",i,time_taken1, time_taken2);
   }
}
```

Output:

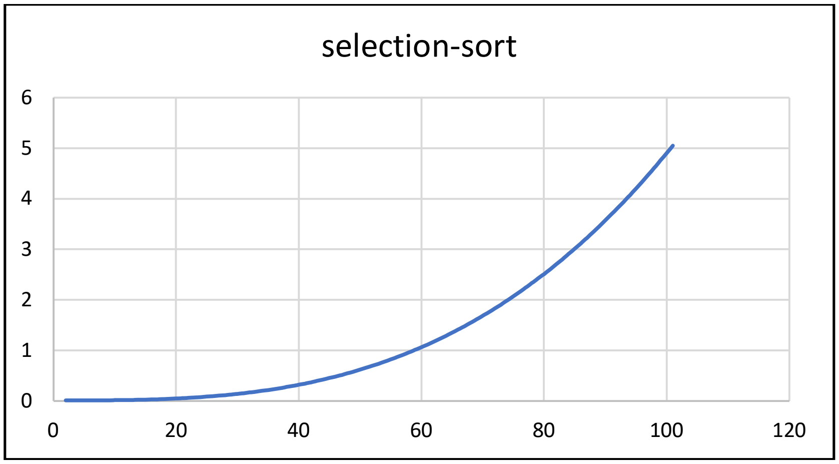


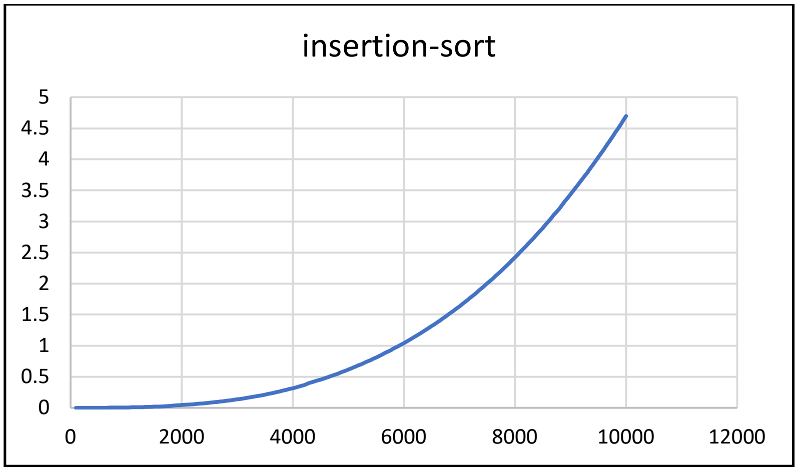
```
Given array:
Numbers selection-sort  insertion-sort

100     0.013000        0.000000

200     0.013000        0.000000

300     0.013000        0.000000

400     0.014000        0.000000

500     0.014000        0.000000

600     0.015000        0.000000

700     0.016000        0.000000

800     0.017000        0.000000

900     0.018000        0.001000

1000    0.019000        0.002000

1100    0.021000        0.003000

1200    0.023000        0.004000

1300    0.026000        0.005000

1400    0.029000        0.006000
```

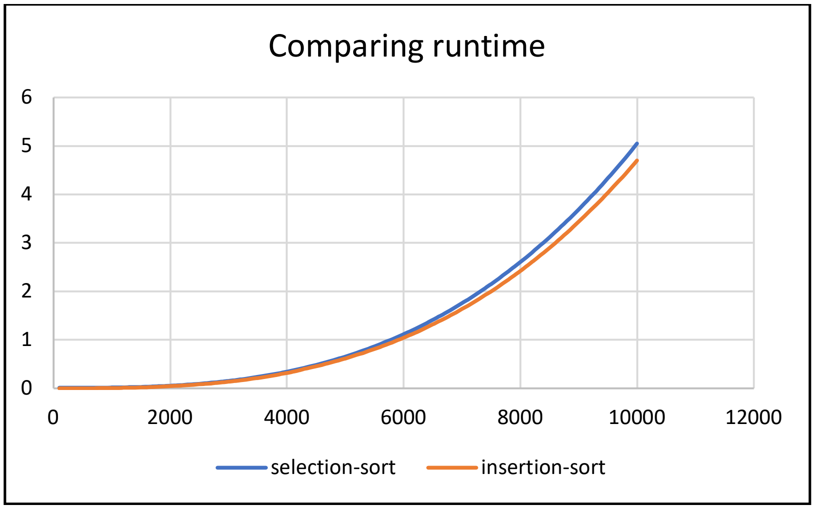


```
8900    2.681000        2.295000

9000    2.770000        2.377000

9100    2.857000        2.457000

9200    2.944000        2.538000

9300    3.040000        2.622000

9400    3.139000        2.717000

9500    3.237000        2.801000

9600    3.335000        2.887000

9700    3.438000        2.973000

9800    3.540000        3.063000

9900    3.643000        3.157000

10000   3.749000        3.251000

Process returned 0 (0x0)   execution time : 7.016 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.