NAME:- Sourav Paul
GROUP:- D2
REG. NO:- 20214056
BRANCH:- CSE DEPT.

ASSIGNMENT - 01

1). Write a C program to analyze the time complexity of insertion sort algorithm. Also plot their graph for all cases.

Ans:- code

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

// utility functions
void swap(long int* a, long int* b)
{
    long int temp = *a;
    *a = *b;
    *b = temp;
```

```
}

void reverse(long int arr[], int n)
{
  int i, temp;
  for(i = 0; i < n / 2; i++)
  {
    temp = arr[i];
    arr[i] = arr[n - 1 - i];
    arr[n - 1 - i] = temp;
  }
}


// insertion sort algorithm

void insertionSort(long int arr[], long int n)
{
                long int i,j,key;
           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()
{
        FILE *filep = fopen("insertiondata.txt", "w"); // write only
         // test for files not existing.
         if (filep == NULL)
           {
             printf("Error! Could not open file\n");
             exit(-1);
           }


        long int n = 1000;
        int temp = 0;
```

```
    double insertionAvg[11], insertionBest[11],
insertionWorst[11]; // array to store time duration of insertion sort
algorithm

    printf("Array_Size Insertion_Average Insertion_Best
Insertion_Worst\n");


    while(temp++ < 11)

    {

        long int num[n];

        int i;

        for(i = 0; i < n; i++)

         {

           long int number = rand() % n + 1;

            num[i] = number;

         }

         clock_t start, end;

          // using clock_t to store time  clock_t start, end;

          start = clock();

          insertionSort(num, n);

          end = clock();

          // AVERAGE CASE FOR INSERTION SORT

          insertionAvg[temp] = (double)(end - start) /
CLOCKS_PER_SEC;
```

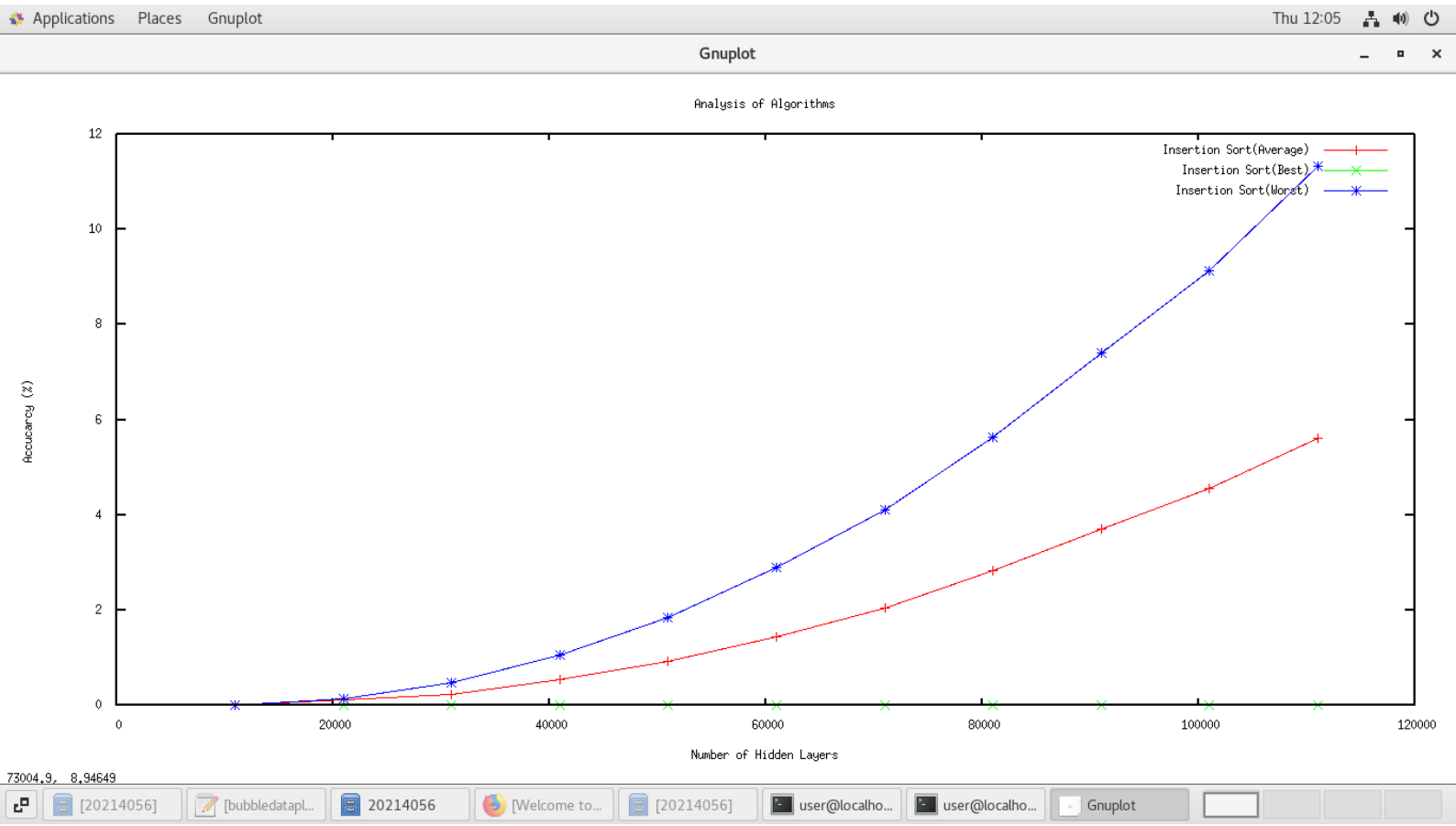
```
            // BEST CASE FOR INSERTION SORT
             start = clock();
              insertionSort(num,n);
              end = clock();

              insertionBest[temp] = (double)(end - start) /
CLOCKS_PER_SEC;

            // WORST CASE FOR INSERTION SORT
            reverse(num,n);

             start = clock();
              insertionSort(num, n);
              end = clock();

              insertionWorst[temp] = (double)(end - start) /
CLOCKS_PER_SEC;



              // type conversion to long int
               // for plotting graph with integer values
```

```
        printf("%li %lf %lf %lf\n",
n,insertionAvg[temp],insertionBest[temp],insertionWorst[temp]);


        n += 10000;
   // write to file
          fprintf(filep,"%li %lf %lf %lf\n",
n,insertionAvg[temp],insertionBest[temp],insertionWorst[temp]);


}
        fclose(filep);
        return 0;
}


/*OUTPUT*/
Array_Size Insertion_Average Insertion_Best Insertion_Worst
11000 0.000000 0.000000 0.000000
21000 0.120000 0.000000 0.150000
31000 0.240000 0.000000 0.490000
41000 0.540000 0.000000 1.070000
51000 0.920000 0.000000 1.850000
61000 1.450000 0.000000 2.900000
71000 2.040000 0.010000 4.110000
81000 2.830000 0.000000 5.640000
```

91000 3.700000 0.000000 7.410000

101000 4.550000 0.000000 9.140000

111000 5.620000 0.000000 11.330000



2). Write a C program to analyze the time complexity of selection sort algorithm. Also plot their graph for all cases.

Ans:- code

#include<stdio.h>

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

// utility functions
void swap(long int* a, long int* b)
{
        long int temp = *a;
        *a = *b;
        *b = temp;
}

void reverse(long int arr[], int n)
{
  int i, temp;
  for(i = 0; i < n / 2; i++)
  {
    temp = arr[i];
    arr[i] = arr[n - 1 - i];
    arr[n - 1 - i] = temp;
  }
}


// selection sort algorithm

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

        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
```

```
            min_idx = j;

        // Swap the found minimum element
        // with the first element
        if(min_idx!=i)
            swap(&arr[min_idx], &arr[i]);
    }
}

int main()
{
        FILE *fp = fopen("selectiondata.txt", "w"); // write only
         // test for files not existing.
         if (fp == NULL)
           {
             printf("Error! Could not open file\n");
             exit(-1);
           }


        long int n = 1000;
        int temp = 0;

        double selectionAvg[11], selectionBest[11],
selectionWorst[11]; // array to store time duration of selection sort
algorithm
        printf("Array_Size Selection_Average Selection_Best
Selection_Worst\n");

        while(temp++ < 11)
        {
                long int num[n];
                int i;
                for(i = 0; i < n; i++)
                 {
                   long int number = rand() % n + 1;
```

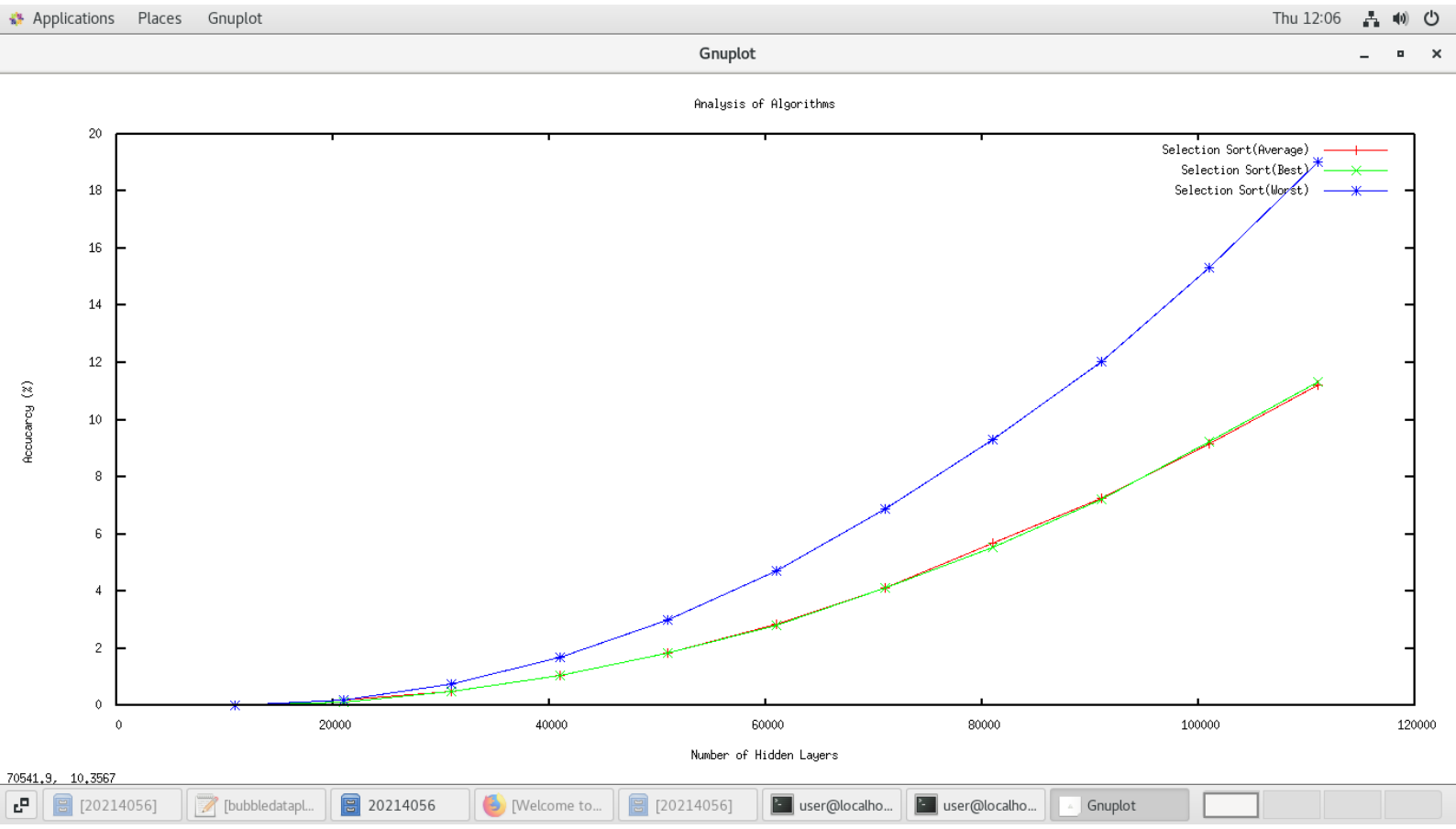
```
            num[i] = number;
          }
          clock_t start, end;
           // using clock_t to store time  clock_t start, end;
           start = clock();
           selectionSort(num, n);
           end = clock();
           // AVERAGE CASE FOR SELECTION SORT
           selectionAvg[temp] = (double)(end - start) /
CLOCKS_PER_SEC;


          // BEST CASE FOR SELECTION SORT
           start = clock();
            selectionSort(num,n);
            end = clock();

            selectionBest[temp] = (double)(end - start) /
CLOCKS_PER_SEC;

          // WORST CASE FOR SELECTION SORT
          reverse(num,n);

           start = clock();
            selectionSort(num, n);
            end = clock();

            selectionWorst[temp] = (double)(end - start) /
CLOCKS_PER_SEC;



            // type conversion to long int
             // for plotting graph with integer values
      printf("%li %lf %lf %lf\n",
n,selectionAvg[temp],selectionBest[temp],selectionWorst[temp]);
```

```
        n += 10000;
    // write to file
          fprintf(fp,"%li %lf %lf %lf\n",
n,selectionAvg[temp],selectionBest[temp],selectionWorst[temp]);

}
        fclose(fp);
        return 0;
}
```

/*OUTPUT*/

Array_Size Selection_Average Selection_Best Selection_Worst

11000 0.000000 0.000000 0.000000
21000 0.200000 0.130000 0.200000
31000 0.480000 0.480000 0.760000
41000 1.060000 1.040000 1.690000
51000 1.860000 1.840000 3.000000
61000 2.840000 2.830000 4.700000
71000 4.110000 4.120000 6.900000
81000 5.690000 5.540000 9.330000
91000 7.260000 7.210000 12.020000
101000 9.180000 9.250000 15.320000
111000 11.210000 11.330000 19.010000



3.) Write a C program to analyze the time complexity of bubble sort algorithm. Also plot their graph for all cases.

Ans:- code

```
#include<stdio.h>

#include<stdlib.h>

#include<time.h>
```

```
// utility functions
void swap(long int* a, long int* b)
{
        long int temp = *a;
        *a = *b;
        *b = temp;
}

void reverse(long int arr[], int n)
{
  int i, temp;
  for(i = 0; i < n / 2; i++)
  {
    temp = arr[i];
    arr[i] = arr[n - 1 - i];
    arr[n - 1 - i] = temp;
  }
}


// bubble sort algorithm

void bubbleSort(long int arr[], long int n)
```

```
{
        long int i,j;
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n - 1 - i; j++)
        {
            if(arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);

        }
    }

}

int main()
{
    FILE *out_file = fopen("bubbledata.txt", "w"); // write only
     // test for files not existing.
     if (out_file == NULL)
       {
         printf("Error! Could not open file\n");
         exit(-1);
       }
```

```
long int n = 1000;
int temp = 0;

double bubbleAvg[11], bubbleBest[11], bubbleWorst[11]; // array to store time duration of bubble sort algorithm
printf("Array_Size Bubble_Worst Bubble_Best Bubble_Average\n");

while(temp++ < 11)
{
    long int num[n];
    int i;
    for(i = 0; i < n; i++)
    {
  long int number = rand() % n + 1;
   num[i] = number;
    }
    clock_t start, end;
     // using clock_t to store time  clock_t start, end;
     start = clock();
     bubbleSort(num, n);
```

```
            end = clock();
            // AVERAGE CASE FOR BUBBLE SORT
            bubbleAvg[temp] = (double)(end - start) /
CLOCKS_PER_SEC;


           // BEST CASE FOR BUBBLE SORT
            start = clock();
            bubbleSort(num, n);
            end = clock();

            bubbleBest[temp] = (double)(end - start) /
CLOCKS_PER_SEC;

           // WORST CASE FOR BUBBLE SORT
           reverse(num,n);

            start = clock();
            bubbleSort(num, n);
            end = clock();

            bubbleWorst[temp] = (double)(end - start) /
CLOCKS_PER_SEC;
```

```
                // type conversion to long int
                 // for plotting graph with integer values
        printf("%li %lf %lf %lf\n",
n,bubbleAvg[temp],bubbleBest[temp],bubbleWorst[temp]);


        n += 10000;
   // write to file
          fprintf(out_file,"%li %lf %lf %lf\n",
n,bubbleAvg[temp],bubbleBest[temp],bubbleWorst[temp]);


}
        fclose(out_file);
        return 0;
}
```

```
/*OUTPUT*/
   Array_Size  Bubble_Worst  Bubble_Best  Bubble_Average
11000 0.000000 0.000000 0.010000
21000 0.390000 0.120000 0.330000
31000 1.220000 0.470000 1.180000
```

41000 2.710000 1.010000 2.580000

51000 4.790000 1.770000 4.480000

61000 7.490000 2.730000 6.990000

71000 10.840000 3.950000 10.070000

81000 14.720000 5.330000 13.590000

91000 19.080000 6.890000 17.560000

101000 24.250000 8.780000 22.330000

111000 30.430000 10.720000 27.560000