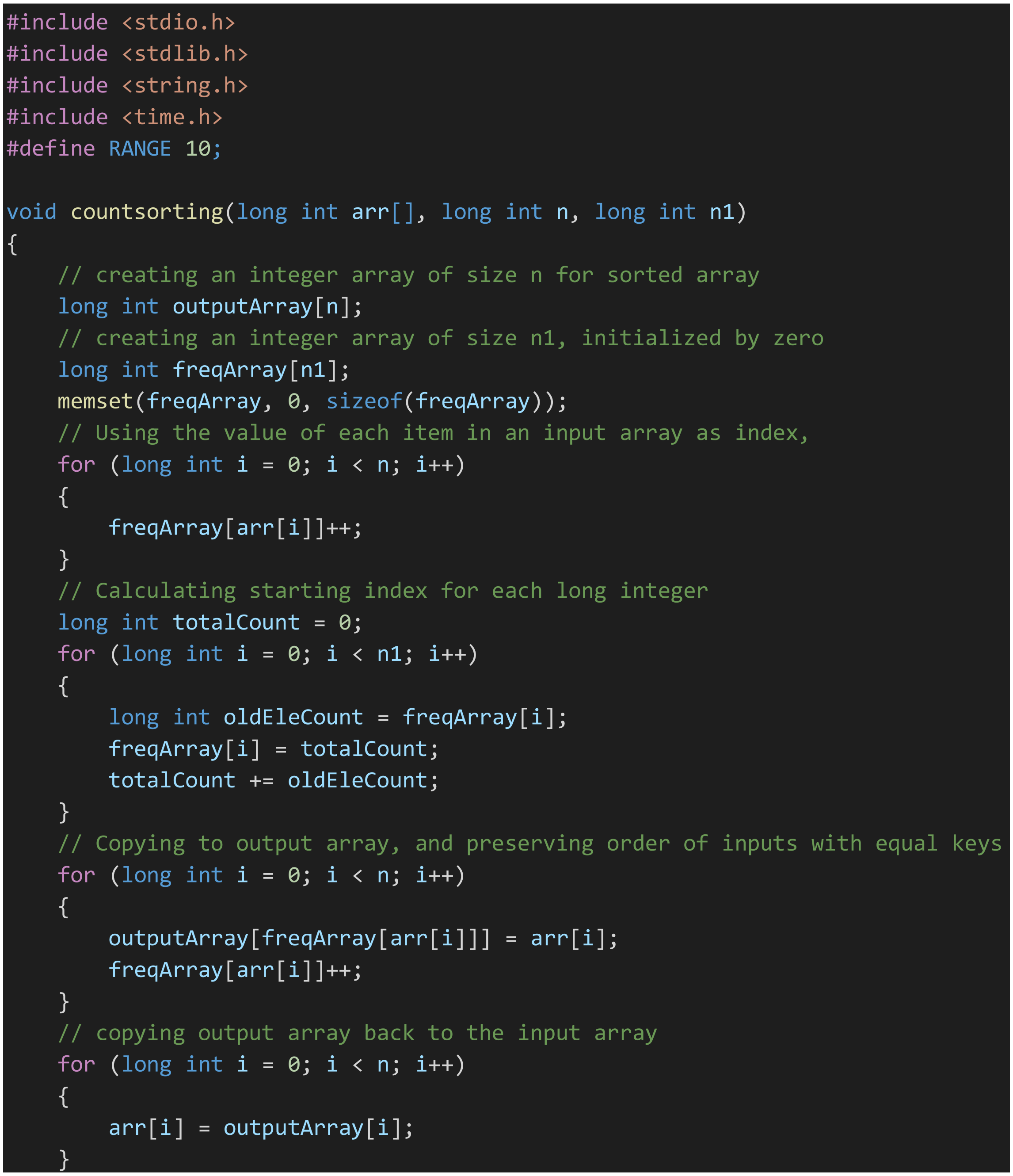
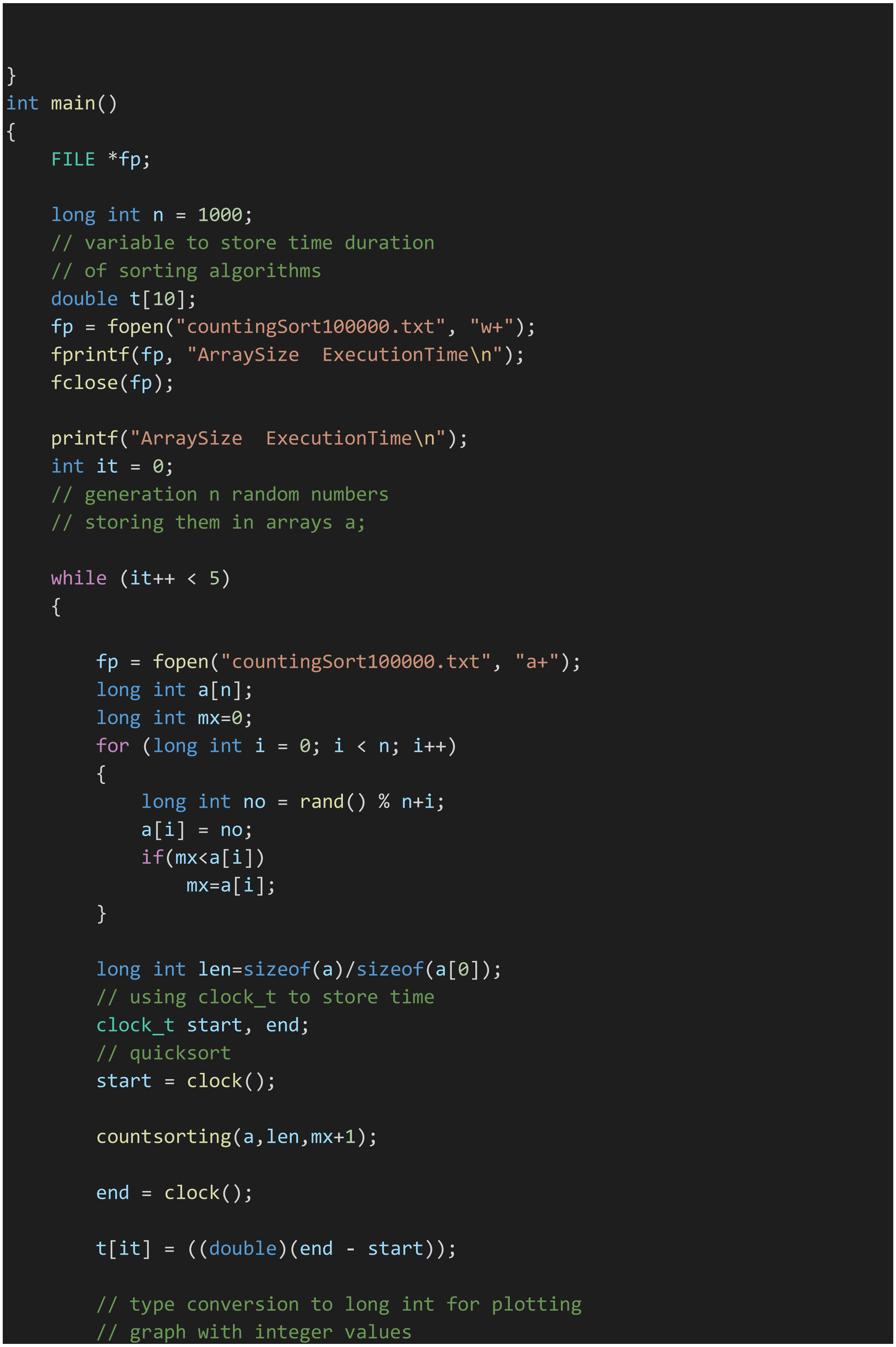
Motilal Nehru National Institute of Technology Allahabad, Prayagraj
Computer Science & Engineering Department
Analysis of Algorithm Lab
Assignment-5
NAME SHISHU
REG 2020CA089

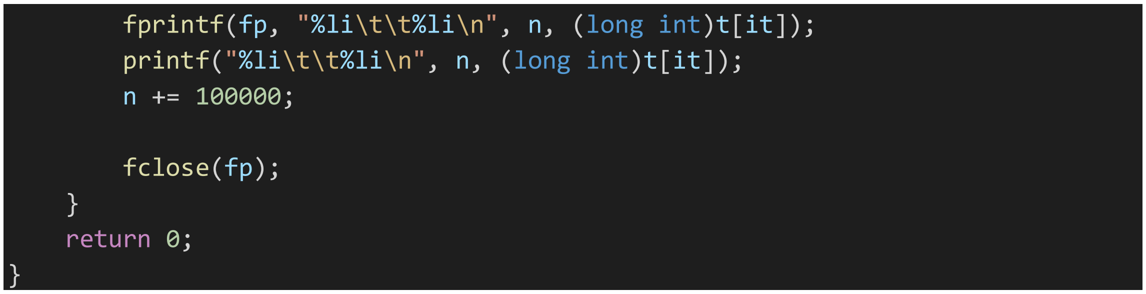
Q-1: Write a C Program to analyse the complexity of Counting Sort Algorithm. Also plot its graph for all cases.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define RANGE 10;

void countsorting(long int arr[], long int n, long int n1)
{
    // creating an integer array of size n for sorted array
    long int outputArray[n];
    // creating an integer array of size n1, initialized by zero
    long int freqArray[n1];
    memset(freqArray, 0, sizeof(freqArray));
    // Using the value of each item in an input array as index,
    for (long int i = 0; i < n; i++)
    {
        freqArray[arr[i]]++;
    }
    // Calculating starting index for each long integer
    long int totalCount = 0;
    for (long int i = 0; i < n1; i++)
    {
        long int oldEleCount = freqArray[i];
        freqArray[i] = totalCount;
        totalCount += oldEleCount;
    }
    // Copying to output array, and preserving order of inputs with equal keys
    for (long int i = 0; i < n; i++)
    {
        outputArray[freqArray[arr[i]]] = arr[i];
        freqArray[arr[i]]++;
    }
    // copying output array back to the input array
    for (long int i = 0; i < n; i++)
    {
        arr[i] = outputArray[i];
    }
```



```c
}
int main()
{
    FILE *fp;

    long int n = 1000;
    // variable to store time duration
    // of sorting algorithms
    double t[10];
    fp = fopen("countingSort100000.txt", "w+");
    fprintf(fp, "ArraySize  ExecutionTime\n");
    fclose(fp);

    printf("ArraySize  ExecutionTime\n");
    int it = 0;
    // generation n random numbers
    // storing them in arrays a;

    while (it++ < 5)
    {

        fp = fopen("countingSort100000.txt", "a+");
        long int a[n];
        long int mx=0;
        for (long int i = 0; i < n; i++)
        {
            long int no = rand() % n+i;
            a[i] = no;
            if(mx<a[i])
                mx=a[i];
        }

        long int len=sizeof(a)/sizeof(a[0]);
        // using clock_t to store time
        clock_t start, end;
        // quicksort
        start = clock();

        countsorting(a,len,mx+1);

        end = clock();

        t[it] = ((double)(end - start));

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



```c
        fprintf(fp, "%li\t\t%li\n", n, (long int)t[it]);
        printf("%li\t\t%li\n", n, (long int)t[it]);
        n += 100000;

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

Countsort1000.txt

```
ArraySize  ExecutionTime
1000    0
2000    0
3000    1
4000    1
5000    0
```

Countsort10000.txt

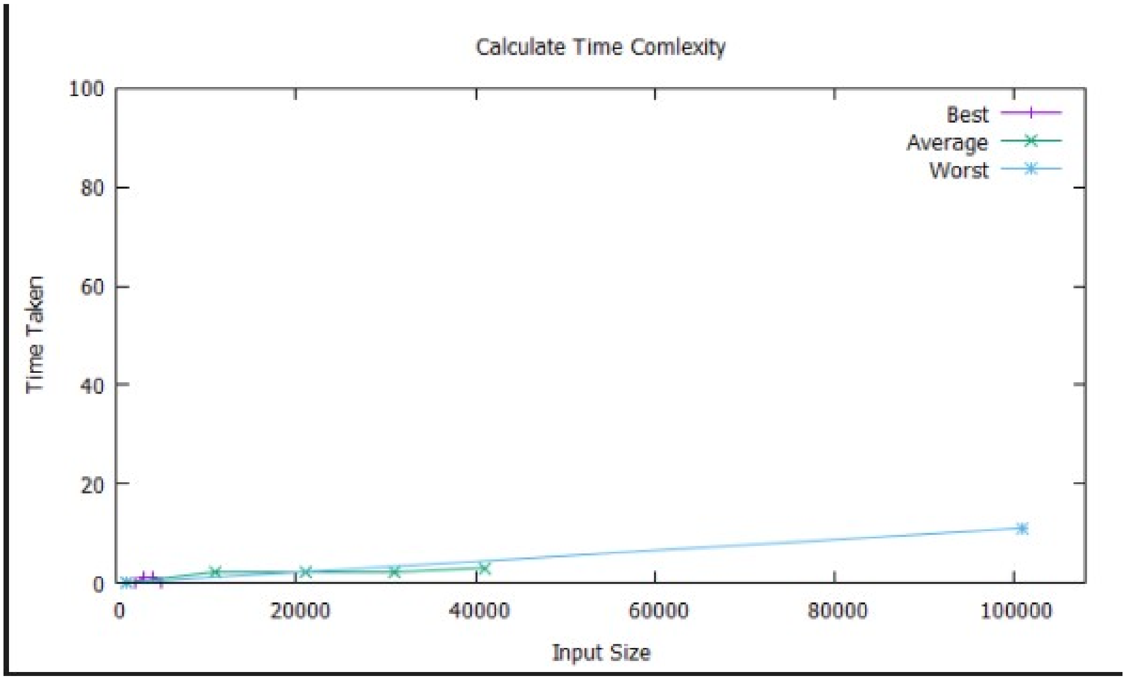
```
ArraySize  ExecutionTime
1000    0
11000           2
21000           2
31000           2
41000           3
```

Countsort100000.txt

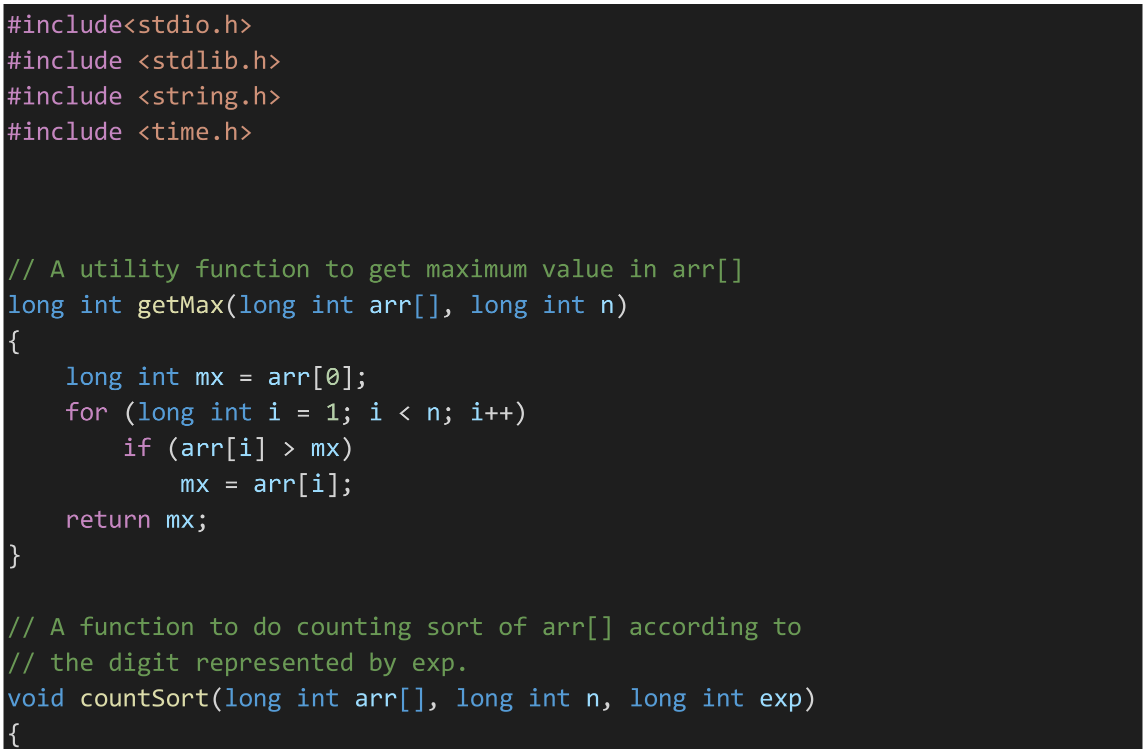
```
ArraySize  ExecutionTime
1000                    0
101000                  11
```

Dataplot.p

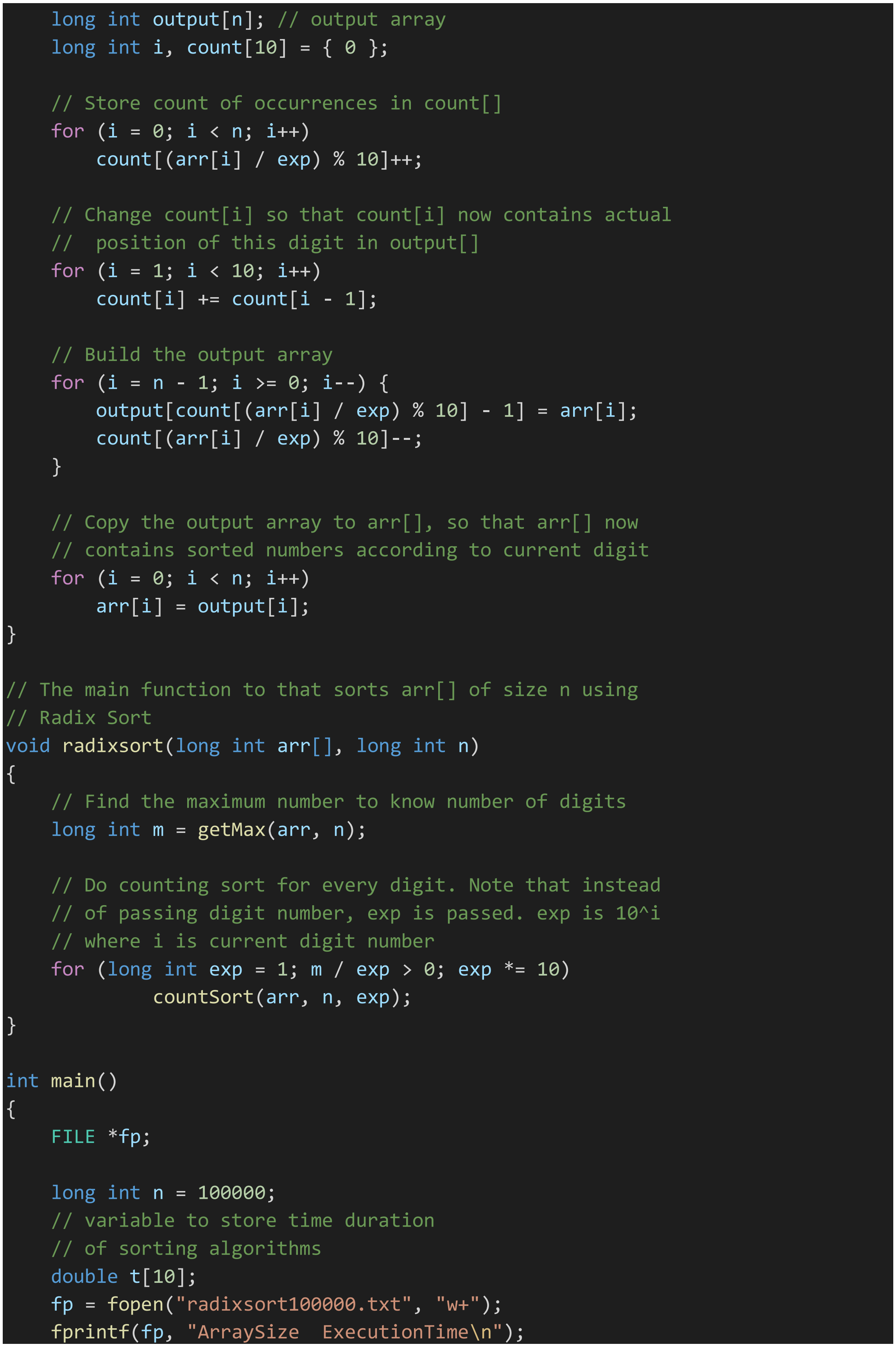
```
  set autoscale                        # scale axes automatically
    unset log                            # remove any log-scaling
    unset label                          # remove any previous labels
    set xtic auto                        # set xtics automatically
    set ytic auto                        # set ytics automatically
        set tics font "Helvetica,10"
    set title "Calculate Time Comlexity"
    set xlabel "Input Size"
    set ylabel "Time Taken"
    #set key 0.01,100
    #set label "Yield Point" at 0.003,260
    #set arrow from 0.0028,250 to 0.003,280
    set xr [0:108000]
    set yr [0.00000:100]
     plot  "countingSort1000.txt" using 1:2 title 'Best' with linespoints, \
"countingSort10000.txt" using 1:2 title 'Average'  with linespoints,\
"countingSort100000.txt" using 1:2 title 'Worst'  with linespoints
```



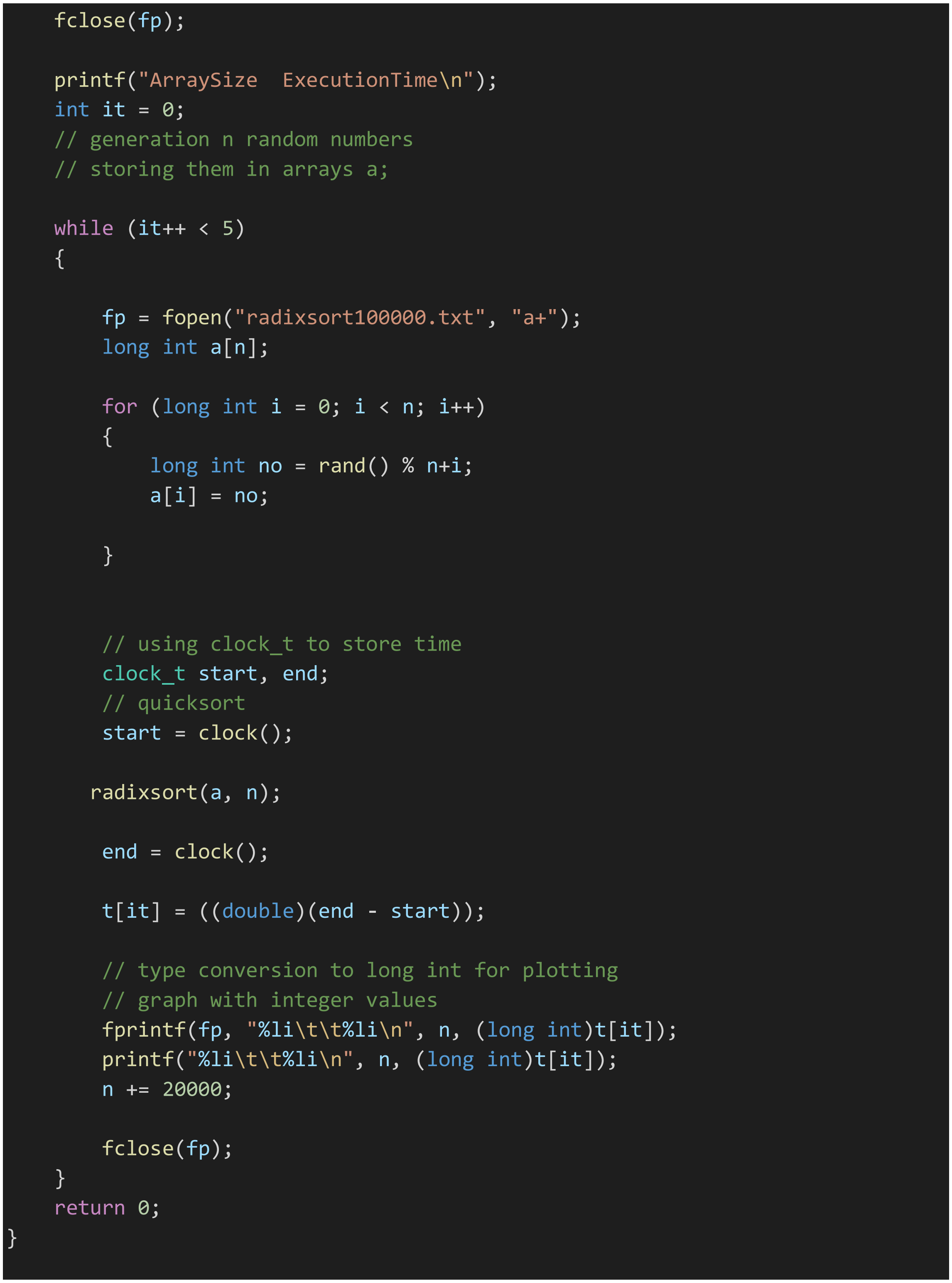
Q-2: Write a C Program to analyse the complexity of Radix Sort Algorithm. Also plot its graph for all cases.



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



// A utility function to get maximum value in arr[]
long int getMax(long int arr[], long int n)
{
    long int mx = arr[0];
    for (long int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(long int arr[], long int n, long int exp)
{
```



```
    long int output[n]; // output array
    long int i, count[10] = { 0 };

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    // Change count[i] so that count[i] now contains actual
    //  position of this digit in output[]
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(long int arr[], long int n)
{
    // Find the maximum number to know number of digits
    long int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (long int exp = 1; m / exp > 0; exp *= 10)
             countSort(arr, n, exp);
}

int main()
{
    FILE *fp;

    long int n = 100000;
    // variable to store time duration
    // of sorting algorithms
    double t[10];
    fp = fopen("radixsort100000.txt", "w+");
    fprintf(fp, "ArraySize  ExecutionTime\n");
```



```
    fclose(fp);

    printf("ArraySize  ExecutionTime\n");
    int it = 0;
    // generation n random numbers
    // storing them in arrays a;

    while (it++ < 5)
    {

        fp = fopen("radixsort100000.txt", "a+");
        long int a[n];

        for (long int i = 0; i < n; i++)
        {
            long int no = rand() % n+i;
            a[i] = no;

        }


        // using clock_t to store time
        clock_t start, end;
        // quicksort
        start = clock();

       radixsort(a, n);

        end = clock();

        t[it] = ((double)(end - start));

        // type conversion to long int for plotting
        // graph with integer values
        fprintf(fp, "%li\t\t%li\n", n, (long int)t[it]);
        printf("%li\t\t%li\n", n, (long int)t[it]);
        n += 20000;

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

Radixsort1000.txt

ArraySize ExecutionTime

```
1000    0
2000    1
3000    1
4000    1
5000    1
```

Radixsort10000.txt

```
ArraySize  ExecutionTime
10000           5
11000           4
12000           3
13000           3
14000           2
```

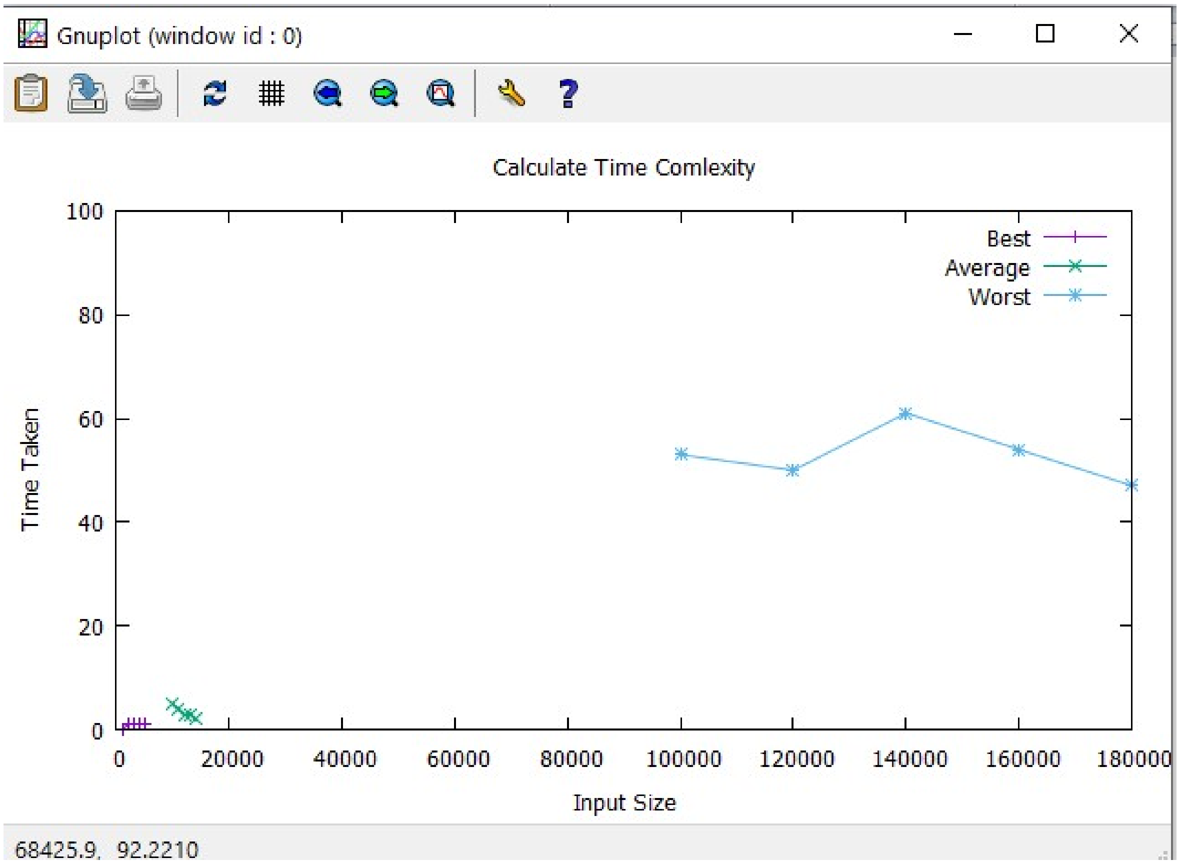
Radixsort100000.txt

```
ArraySize  ExecutionTime
100000                  53
120000                  50
140000                  61
160000                  54
180000                  47
```

Dataplot.p

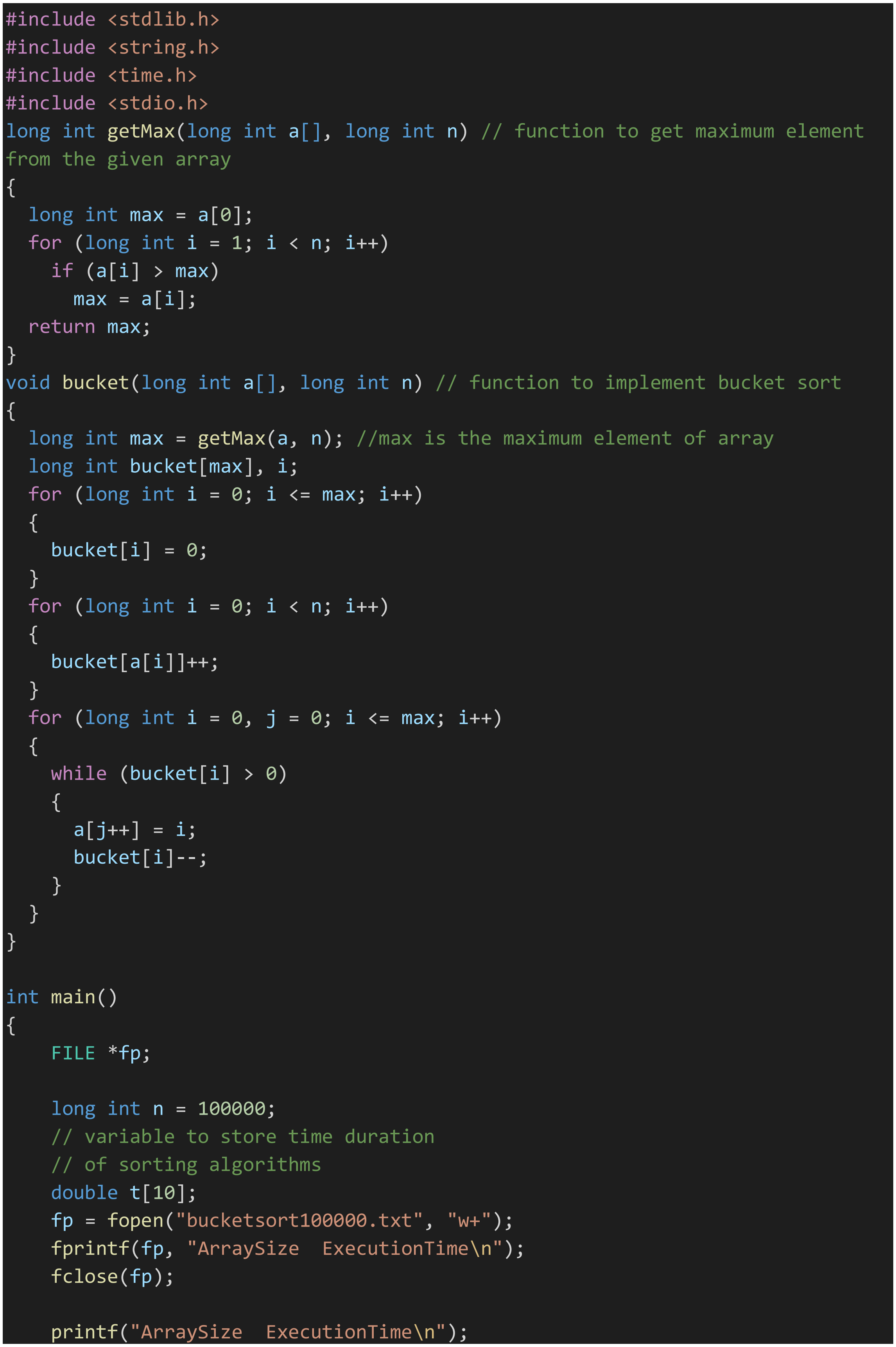
```
  set autoscale                      # scale axes automatically
    unset log                          # remove any log-scaling
    unset label                        # remove any previous labels
    set xtic auto                      # set xtics automatically
    set ytic auto                      # set ytics automatically
        set tics font "Helvetica,10"
    set title "Calculate Time Comlexity"
```

```
set xlabel "Input Size"

set ylabel "Time Taken"

#set key 0.01,100

#set label "Yield Point" at 0.003,260

#set arrow from 0.0028,250 to 0.003,280

set xr [0:180000]

set yr [0.00000:100]

 plot  "radixsort1000.txt" using 1:2 title 'Best' with linespoints, \

    "radixsort10000.txt" using 1:2 title 'Average'  with linespoints,\

    "radixsort100000.txt" using 1:2 title 'Worst'  with linespoints
```

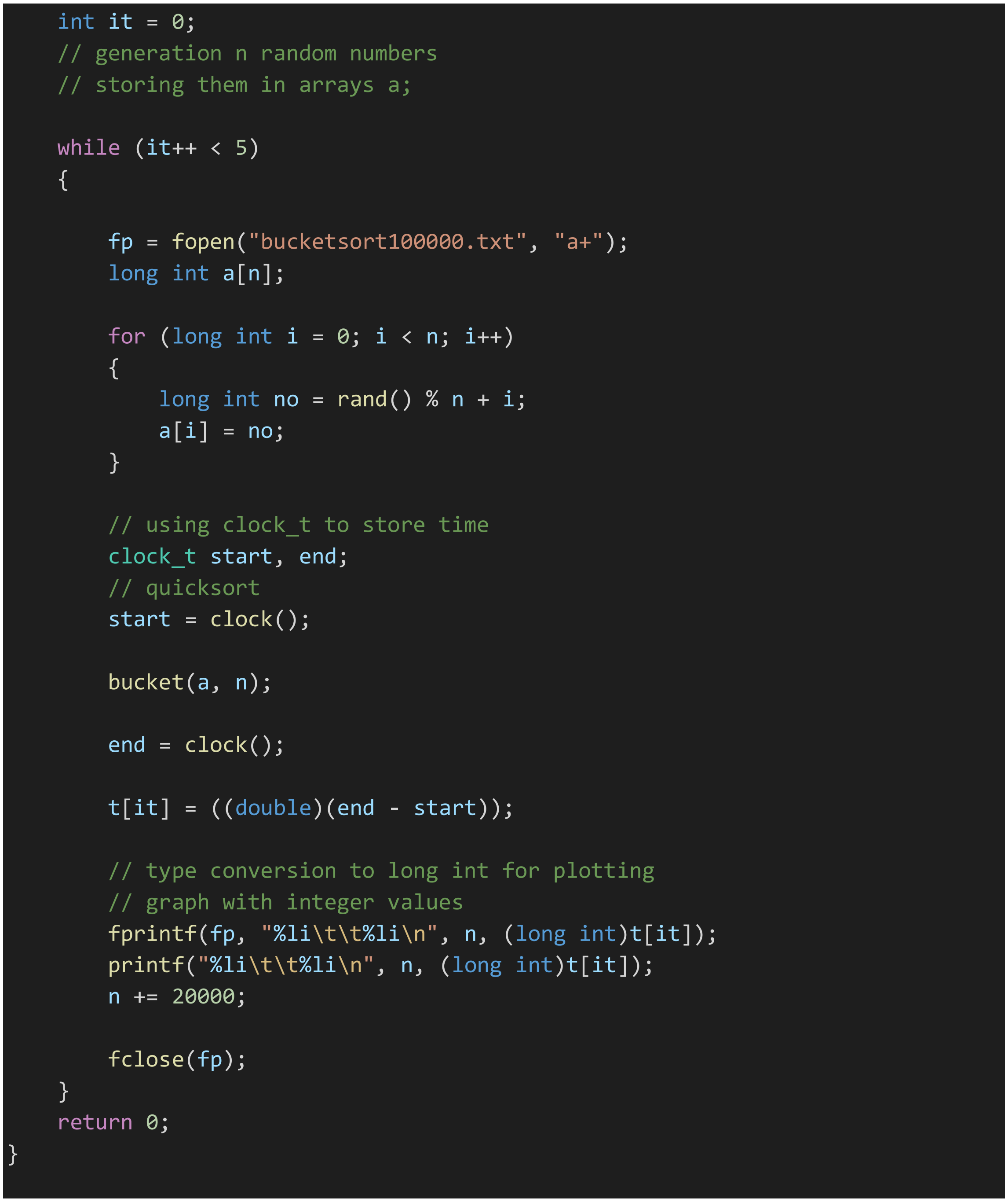


Q-3: Write a C Program to analyse the complexity of Bucket Sort Algorithm. Also plot its graph for all cases.

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



```
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>
long int getMax(long int a[], long int n) // function to get maximum element
from the given array
{
  long int max = a[0];
  for (long int i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
void bucket(long int a[], long int n) // function to implement bucket sort
{
  long int max = getMax(a, n); //max is the maximum element of array
  long int bucket[max], i;
  for (long int i = 0; i <= max; i++)
  {
    bucket[i] = 0;
  }
  for (long int i = 0; i < n; i++)
  {
    bucket[a[i]]++;
  }
  for (long int i = 0, j = 0; i <= max; i++)
  {
    while (bucket[i] > 0)
    {
      a[j++] = i;
      bucket[i]--;
    }
  }
}

int main()
{
    FILE *fp;

    long int n = 100000;
    // variable to store time duration
    // of sorting algorithms
    double t[10];
    fp = fopen("bucketsort100000.txt", "w+");
    fprintf(fp, "ArraySize  ExecutionTime\n");
    fclose(fp);

    printf("ArraySize  ExecutionTime\n");
```



```
    int it = 0;
    // generation n random numbers
    // storing them in arrays a;

    while (it++ < 5)
    {

        fp = fopen("bucketsort100000.txt", "a+");
        long int a[n];

        for (long int i = 0; i < n; i++)
        {
            long int no = rand() % n + i;
            a[i] = no;
        }

        // using clock_t to store time
        clock_t start, end;
        // quicksort
        start = clock();

        bucket(a, n);

        end = clock();

        t[it] = ((double)(end - start));

        // type conversion to long int for plotting
        // graph with integer values
        fprintf(fp, "%li\t\t%li\n", n, (long int)t[it]);
        printf("%li\t\t%li\n", n, (long int)t[it]);
        n += 20000;

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

Bucketsort1000.txt

```
ArraySize  ExecutionTime

1000    0

3000    0

5000    1
```

```
7000    0

9000    1
```

Bucketsort10000.txt

```
ArraySize  ExecutionTime

10000           1

30000           6

50000           2

70000           4

90000           4
```

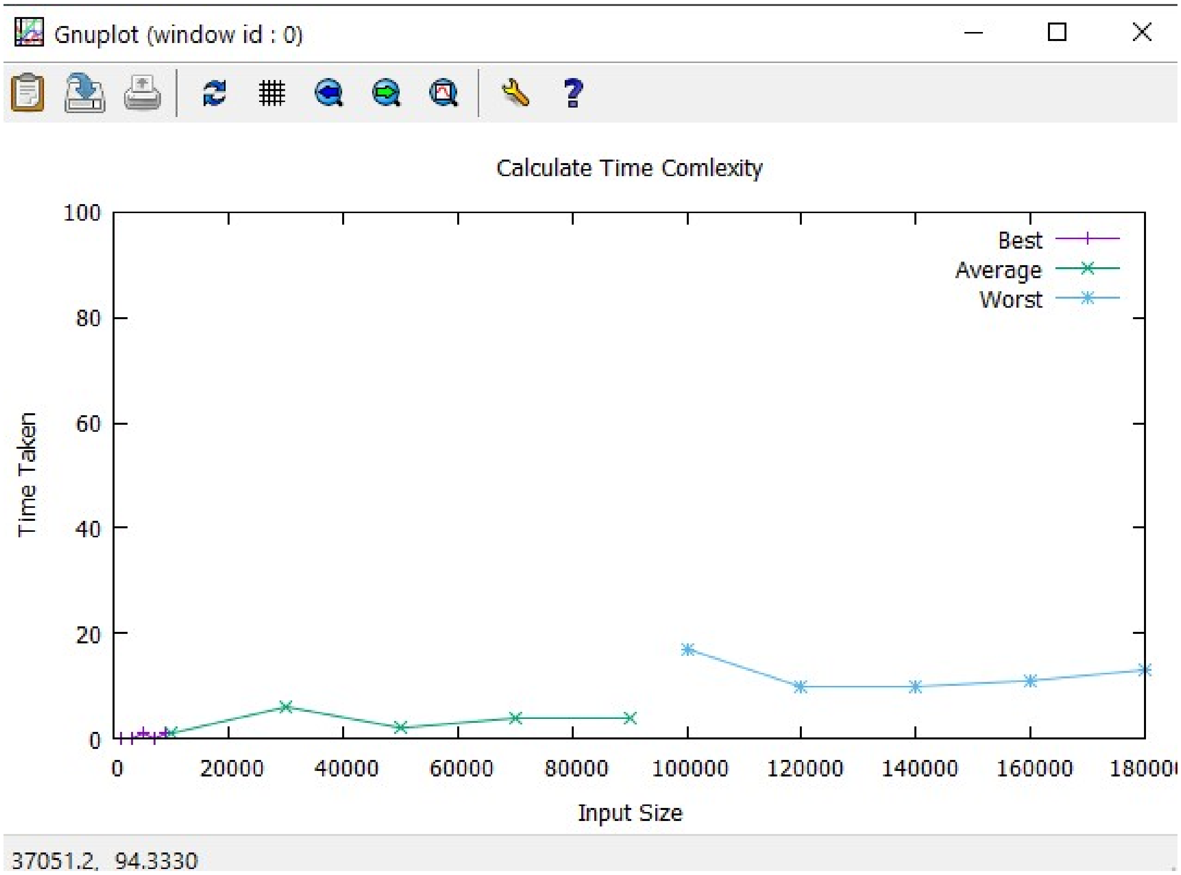
Bucketsort100000.txt

```
ArraySize  ExecutionTime

100000                17

120000                10

140000                10

160000                11

180000                13
```

Dataplot.p

```
 set autoscale                       # scale axes automatically

   unset log                           # remove any log-scaling

   unset label                         # remove any previous labels

   set xtic auto                       # set xtics automatically

   set ytic auto                       # set ytics automatically

       set tics font "Helvetica,10"

   set title "Calculate Time Comlexity"

   set xlabel "Input Size"

   set ylabel "Time Taken"

   #set key 0.01,100
```

```
#set label "Yield Point" at 0.003,260

#set arrow from 0.0028,250 to 0.003,280

set xr [0:180000]

set yr [0.00000:100]

 plot  "bucketsort1000.txt" using 1:2 title 'Best' with linespoints, \

     "bucketsort10000.txt" using 1:2 title 'Average'  with linespoints,\

     "bucketsort100000.txt" using 1:2 title 'Worst'  with linespoints
```


Gnuplot (window id : 0)
Calculate Time Comlexity
Best
Average
Worst
Time Taken
0
20
40
60
80
100
Input Size
0
20000
40000
60000
80000
100000
120000
140000
160000
18000
37051.2, 94.3330