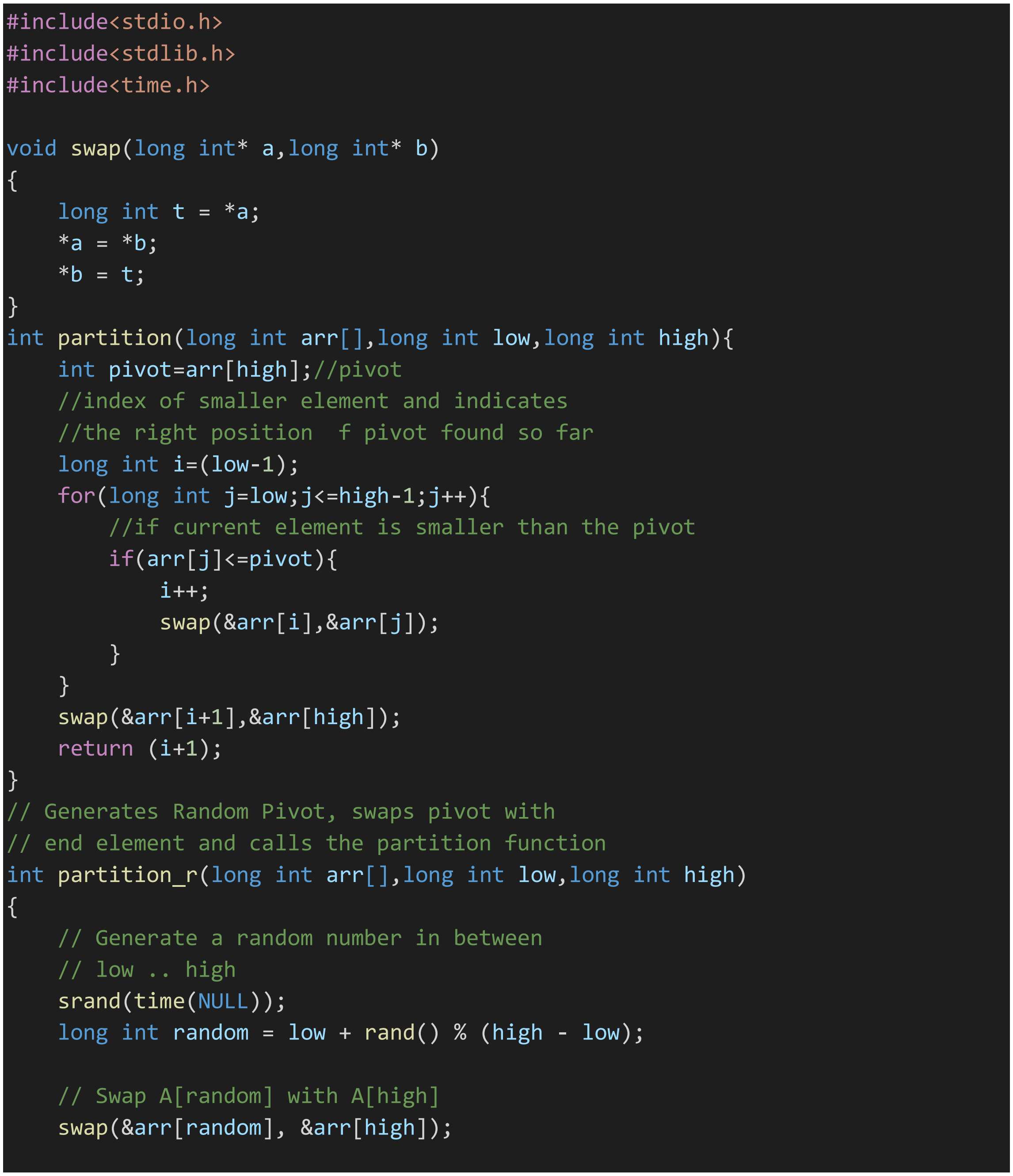
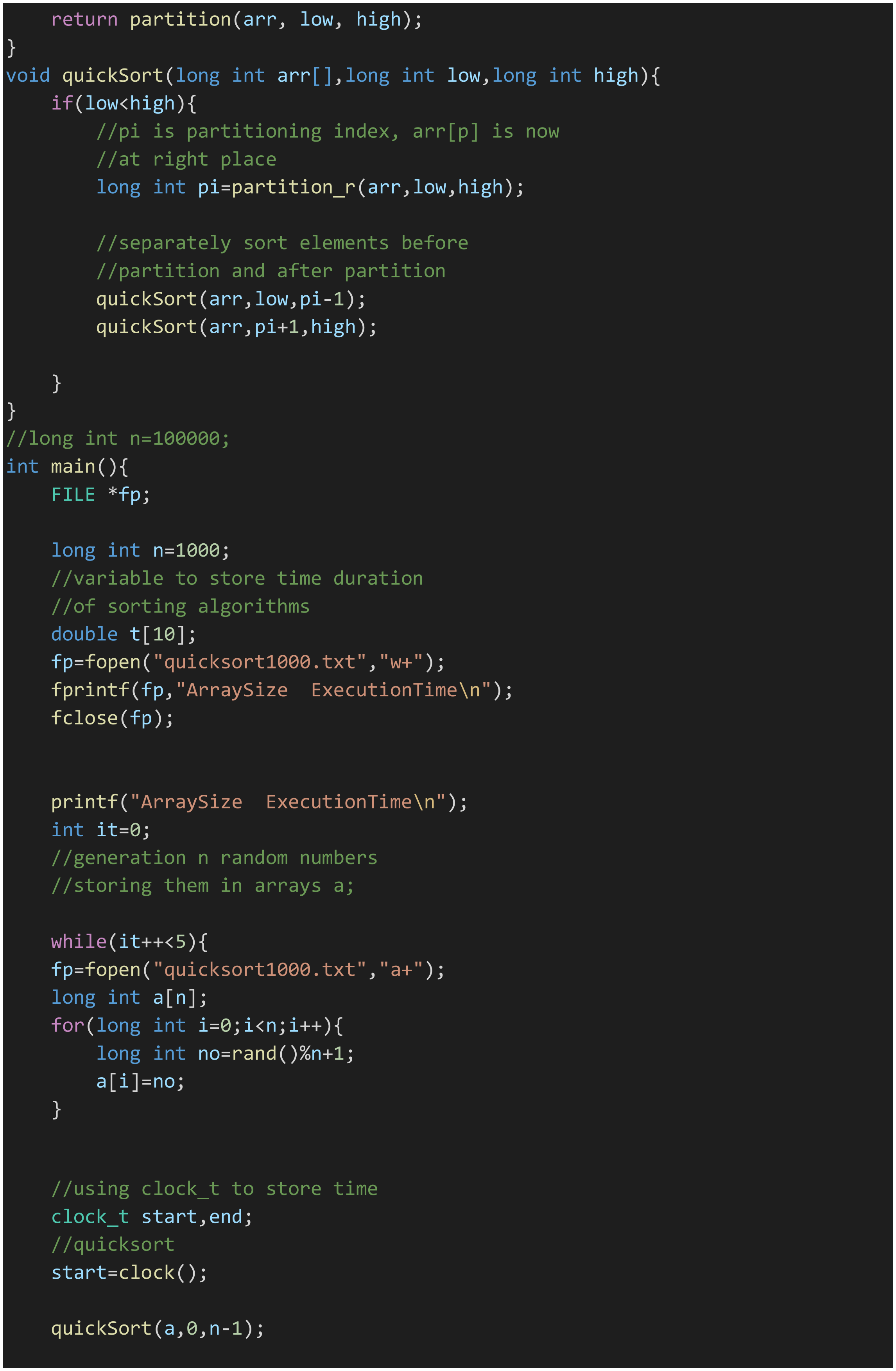
Motilal Nehru National Institute of Technology Allahabad, Prayagraj
Computer Science & Engineering Department
MCA (3rd Semester)
Assignment-3
NAME – SHISHU
REG—2020CA089

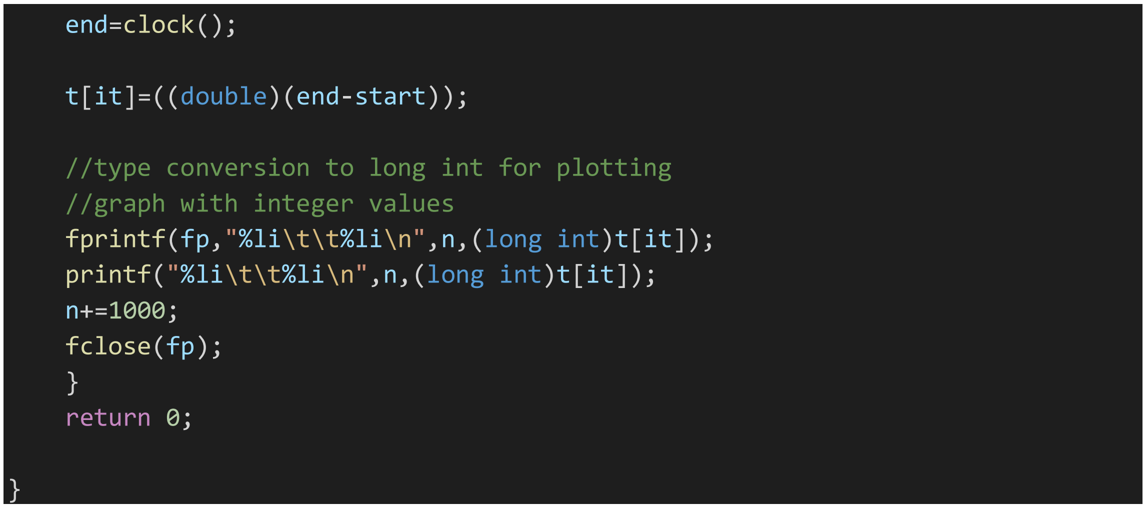
Q-1: Write a C program to analyze the time complexity of Quick Sort algorithm. Also plot their graph for all cases.



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

void swap(long int* a,long int* b)
{
    long int t = *a;
    *a = *b;
    *b = t;
}
int partition(long int arr[],long int low,long int high){
    int pivot=arr[high];//pivot
    //index of smaller element and indicates
    //the right position  f pivot found so far
    long int i=(low-1);
    for(long int j=low;j<=high-1;j++){
        //if current element is smaller than the pivot
        if(arr[j]<=pivot){
            i++;
            swap(&arr[i],&arr[j]);
        }
    }
    swap(&arr[i+1],&arr[high]);
    return (i+1);
}
// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
int partition_r(long int arr[],long int low,long int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    long int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(&arr[random], &arr[high]);
```



```
    return partition(arr, low, high);
}
void quickSort(long int arr[],long int low,long int high){
    if(low<high){
        //pi is partitioning index, arr[p] is now
        //at right place
        long int pi=partition_r(arr,low,high);

        //separately sort elements before
        //partition and after partition
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);

    }
}
//long int n=100000;
int main(){
    FILE *fp;

    long int n=1000;
    //variable to store time duration
    //of sorting algorithms
    double t[10];
    fp=fopen("quicksort1000.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("quicksort1000.txt","a+");
    long int a[n];
    for(long int i=0;i<n;i++){
        long int no=rand()%n+1;
        a[i]=no;
    }


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

    quickSort(a,0,n-1);
```



```
    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+=1000;
    fclose(fp);
    }
    return 0;

}
```

Bestcase file is →quicksort1000.txt

ArraySize	ExecutionTime
1000	1
2000	1
3000	2
4000	2
5000	2

Averagecase file is →quicksort10000.txt

ArraySize	ExecutionTime
10000	8
11000	10
12000	6
13000	3
14000	4

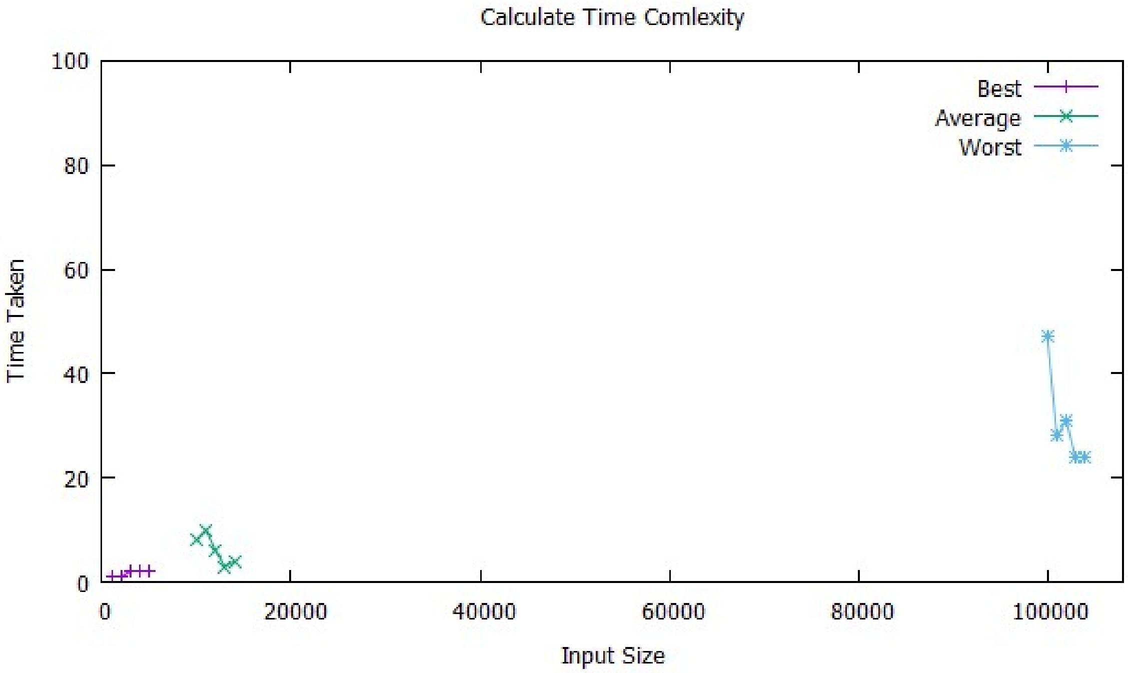
Worstcase file is → quicksort100000.txt

ArraySize ExecutionTime

```
100000                47
101000                28
102000                31
103000                24
104000                24
```

DataPloat File is → dataploat.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  "quicksort1000.txt" using 1:2 title 'Best' with linespoints, \
         "quicksort10000.txt" using 1:2 title 'Average'  with linespoints,\
         "quicksort100000.txt" using 1:2 title 'Worst'  with linespoints
```


Calculate Time Comlexity
Best
Average
Worst
Time Taken
0
20
40
60
80
100
0
20000
40000
60000
80000
100000
Input Size