1. Student Name:-

2. Student Roll No.:-

Algorithm Lab. Class Assignment-3

CSE Group 1

Date: - 23rd July 2021

1. **Write a program to test whether a number n, entered through the keyboard is prime or not by using different algorithms you know for at least 10 inputs. Note down the time complexity for each algorithm along with each input. Finally make a comparison of time complexities found for different inputs, plot an appropriate graph & decide which algorithm is faster.**

SL No.	Input (n)	Prime Number Testing		
		Algorithm – 1 (Time complexity)	Algorithm – 2 (Time complexity)	Algorithm – 3 (Time complexity)
1	100000	0.000051	0.000007	
2	200000	0.000040	0.000012	
3	300000	0.000049	0.000006	
4	400000	0.000031	0.000011	
5	500000	0.000053	0.000010	
6	600000	0.000024	0.000006	
7	700000	0.000035	0.000015	
8	800000	0.000048	0.000009	
9	900000	0.000041	0.000005	
10	1000000	0.000038	0.000008	

N.B: If you can solve more than three different ways, then add more columns right of Algorithm-3 column.

Program

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


#define sf(x)               scanf("%d", &x)
#define pf                  printf
#define pfs(x)              printf("%d ", x)
#define pfn(x)              printf("%d\n", x)
#define pfc(x)              printf("%d, ", x)
#define f(i,x,y)            for(int i = x; i < y; i++)
#define fi(i,x,y,inc)       for(int i = x; i < y; i += inc)
#define rf(i,x,y)           for(int i = x; i >= y; i--)
```

```
void c_() {

#ifndef ONLINE_JUDGE
    freopen("C:\\Users\\KIIT\\input", "r", stdin);
    freopen("C:\\Users\\KIIT\\output", "w", stdout);
#endif
}
int checkPrimeSimple(int n) {

    f(i, 2, n) {
        if (n % i == 0) {
            return 0;
        }
    }
    return 1;

}
int checkPrimeSqRoot(int n) {

    int sq = sqrt(n);

    f(i, 2, sq + 1) {
        if (n % i == 0) {
            return 0;
        }
    }

    return 1;

}
int main() {

    c_();

    time_t start, end;
    int n;
    sf(n);


    start = clock();
    pfn(checkPrimeSimple(n));
    end = clock();
    double t = end - start;
    printf("Time taken for simple algo: %f\n", (t / CLOCKS_PER_SEC));


    start = clock();
    pfn(checkPrimeSqRoot(n));
    end = clock();
    t = end - start;
    printf("Time taken for sq root algo: %f\n", (t / CLOCKS_PER_SEC))
;
    return 0;
}
```

Output

```
1
Time taken for simple algo: 0.000015
1
Time taken for sq root algo: 0.000001
```

Graph

2. **Write a program to find out GCD (greatest common divisor) using Euclid's algorithm and calculate the time complexity.**

   I. **Find GCD of two numbers when both are very large i.e. GCD (31415, 14142) by applying the above algorithm.**

   II. **Find GCD of two numbers when one of them can be very large i.e. GCD (56, 32566) or GCD (34218, 56) by applying the above algorithm.**

   III. **Find GCD of two numbers when both are very small i.e. GCD (12, 15) by applying the above algorithm.**

   IV. **Find GCD of two numbers when both are same i.e. GCD (31415, 31415) or GCD (12, 12) by applying the above algorithm.**

Write the above data in the following format and draw the graph.

Sl. No.	Input GCD (x, y)	

		GCD Algorithm - Time complexity
		Using Euclid's Algorithm
1	GCD (31415, 14142)	0.000000
2	GCD (56, 32566)	0.000000
3	GCD (34218, 56)	0.000000
4	GCD (12, 15)	0.000000
5	GCD (31415, 31415)	0.000000
6	GCD (12, 12)	-1.000000

Program

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

long long gcd(long long m, long long n)
{
    if(n>m){
        long long temp = m;
        m=n;
        n=temp;
    }
    while (n>0){
        long long r= m%n;
        m=n;
        n=r;
    }
    return m;
}

double getTimeForExection(long long m, long long n){
    double start= clock();
    gcd(m,n);
    double end = clock()-start;
    return (end-start);
}

int main(){
    printf("Time for GCD (31415,14142) : %f\n", getTimeForExection( 31415, 14142));
    printf("Time for GCD (56,32566) : %f\n", getTimeForExection( 56, 32566));
```

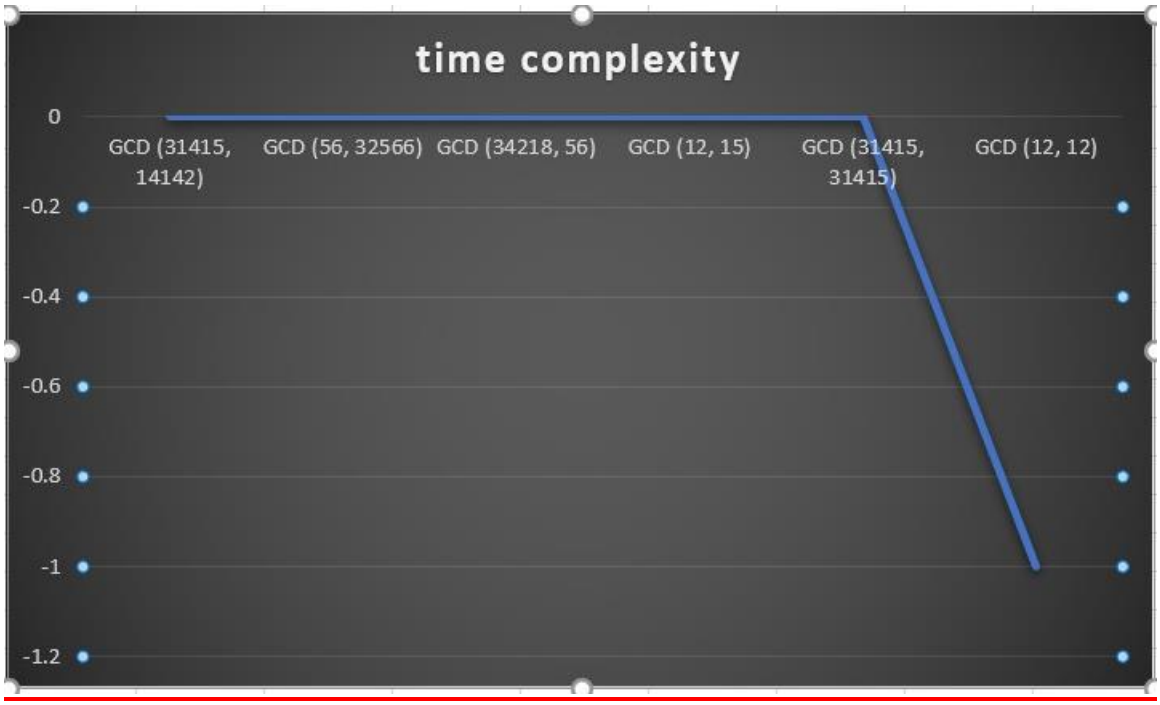
```
    printf("Time for GCD (34218,56) : %f\n", getTimeForExection( 3421
8,56));
    printf("Time for GCD (12,15) : %f\n", getTimeForExection( 12, 15)
);
    printf("Time for GCD (31415,31415) : %f\n", getTimeForExection( 3
1415, 31415));
    printf("Time for GCD (12,12) : %f\n", getTimeForExection( 12, 12)
);
}
```

Output

```
C:\Users\hp\Documents\DAA LAB>cd "c:\Users\hp\Documents\DAA LAB\LAB-3\" && gcc que2.c -o que2 && "c:\Users\hp\Documents\DAA LAB\LAB-3\"que2
Time for GCD (31415,14142) : 0.000000
Time for GCD (56,32566) : 0.000000
Time for GCD (34218,56) : 0.000000
Time for GCD (12,15) : 0.000000
Time for GCD (31415,31415) : 0.000000
Time for GCD (12,12) : -1.000000

c:\Users\hp\Documents\DAA LAB\LAB-3>
```

Graph



3. **Write a menu driven program as given below, to sort an array of n integers in ascending order by insertion sort algorithm and determine the time required to sort the elements.**

Repeat the experiment for different values of n and different nature of data (i.e. apply insertion sort algorithm on the data of array that are already sorted, reversely sorted and random data). Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator.

NSERTION SORT MENU

0. **Quit**
1. **n Random numbers ➔ Array**
2. **Display the Array**
3. **Sort the Array in Ascending Order by using Insertion Sort Algorithm**
4. **Sort the Array in Descending Order by using any sorting Algorithm**
5. **Time Complexity to sort ascending of random data**
6. **Time Complexity to sort ascending of data already sorted in ascending order**
7. **Time Complexity to sort ascending of data already sorted in descending order**
8. **Time Complexity to sort ascending of data for all Cases (Data Ascending, Data in descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000**

Enter your choice:

If the choice is option 8, then it will display the tabular form as follows:

Sl No.	Value of n	Time Complexity (Data in Ascending)	Time Complexity (Data in descending)	Time Complexity (Random Data)
1	5000	0.029000	0.000000	0.047000
2	10000	0.111000	0.000000	0.157000
3	15000	0.207000	0.005000	0.333000
4	20000	0.279000	0.000000	0.537000
5	25000	0.413000	0.000000	0.791000
6	30000	0.601000	0.012000	1.169000
7	35000	0.778000	0.000000	1.548000
8	40000	1.011000	0.004000	2.016000
9	45000	1.255000	0.000000	2.514000
10	50000	1.702000	0.000000	3.215000

Program

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

```c
#include<stdlib.h>
#include<time.h>
int a[1000000];
clock_t start,end;
double tot;
void func1(int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        a[i]=rand()%n;
    }
}
void func2(int n)
{
    int i;
    printf("\n Current Array");
    for(i=0;i<n;i++)
    {
        printf("\n%d",a[i]);
    }
}
double func3(int n,int o)
{
    srand(time(NULL));
    start=clock();
    int i,key,j;
    for(i=1;i<n;i++)
    {
        key=a[i];
        j=i-1;
        while(j>=0&&a[j]>key)
        {
            a[j+1]=a[j];
            j=j-1;
        }
        a[j+1]=key;
    }
    end=clock();
    tot=((double)end-start)/((double)CLOCKS_PER_SEC);
    if(o==0) return 0;
    else return tot;
}
double func4(int n, int o)
{
    srand(time(NULL));
    start=clock();
    int i,j,t;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(a[i]<a[j])
            {
```

```
                t=a[j];
                a[j]=a[i];
                a[i]=t;
            }
        }
    }
    end=clock();
    tot=((double)(end-start))/((double)(CLOCKS_PER_SEC));
    if(o==0) return 0;
    else return tot;
}
void func5()
{
    int i,n;
    double ti;
    printf("\nValue of n\t\tRandom Data\t\tAscending\t\tDescending");
    for(i=5000;i<=50000;i+=5000)
    {
        printf("\n%d",i);
        func1(i);
        ti=func3(i,1);
        printf("\t\t%f",ti);
        ti=func3(i,1);
        printf("\t\t%f",ti);
        func4(i,0);
        ti=func4(i,1);
        printf("\t\t%f",ti);
    }
}
int main()
{
    int n,i;
    double ti;
    printf("\n Enter n :");
    scanf("%d",&n);
    while(1)
    {
       printf("\n0. To Exit");
       printf("\n1. n Random numbers -> Array");
       printf("\n2. Display the Array");
       printf("\n3. Sort the Array in Ascending Order by using Insert
ion Sort Algorithm");
       printf("\n4. Sort the Array in Descending Order by using any s
orting Algorithm");
       printf("\n5. Time Complexity to sort ascending of random data"
);
       printf("\n6. Time Complexity to sort ascending of data already
 sorted in ascending order");
       printf("\n7. Time Complexity to sort ascending of data already
 sorted in descending order");
       printf("\n8. Time Complexity to sort ascending of data for all
 Cases (Data Ascending, Data in descending & Random Data) in Tabular
form for values n=5000 to 50000, step=5000");
       printf("\n Choose");
```

```
        scanf("%d",&i);
        switch(i)
        {
              case 0: exit(0);
                      break;
              case 1: func1(n);
                      break;
              case 2: func2(n);
                      break;
              case 3: func3(n,0);
                      break;
              case 4: func4(n,0);
                      break;
              case 5: ti=func3(n,1);
                      printf("\n Time Taken =%f",ti);
                      break;
              case 6: func3(n,0);
                      func2(n);
                      ti=func3(n,1);
                      printf("\n Time Taken for sorted=%f",ti);
                      break;
              case 7: func4(n,0);
                      func2(n);
                      ti=func3(n,1);
                      printf("\n Time taken for sorted=%f",ti);
                      break;
              case 8: func5();
                      break;
              default: printf("\n Wrong Entry");
        }

    }
    return 0;
}
```

Output

```
8. Time Complexity to sort ascending of data for all Cases (Data Ascending, Data in descending & Random Data) in Tabular form for values n=5000 to 5000
0, step=5000
 Choose1

0. To Exit
1. n Random numbers -> Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting Algorithm
5. Time Complexity to sort ascending of random data
6. Time Complexity to sort ascending of data already sorted in ascending order
7. Time Complexity to sort ascending of data already sorted in descending order
8. Time Complexity to sort ascending of data for all Cases (Data Ascending, Data in descending & Random Data) in Tabular form for values n=5000 to 5000
0, step=5000
 Choose2

 Current Array
5
5
4
4
5
4
0. To Exit
1. n Random numbers -> Array
2. Display the Array
3. Sort the Array in Ascending Order by using Insertion Sort Algorithm
4. Sort the Array in Descending Order by using any sorting Algorithm
```

Graph


Chart Title
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
10000
20000
30000
40000
50000
60000
Time Complexity (data in asc)
Time Complexity(data in desc)
Time Complexity (random data)