Chaudhary Hamdan

1905387

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.**

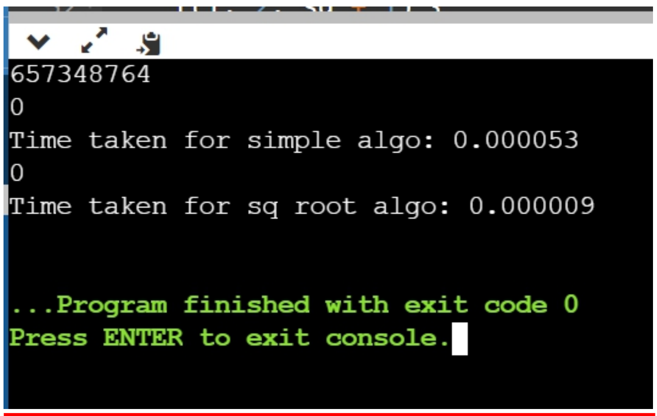
SL No.	Input (n)	Prime Number Testing		
		Algorithm – 1 (Time complexity)	Algorithm – 2 (Time complexity)	Algorithm – 3 (Time complexity)
1	100000	0.000051	0.000007	
2	200000	0.000040	0.000012	
3	300000	0.000049	0.000006	
4	400000	0.000031	0.000011	
5	500000	0.000053	0.000010	
6	600000	0.000024	0.000006	
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

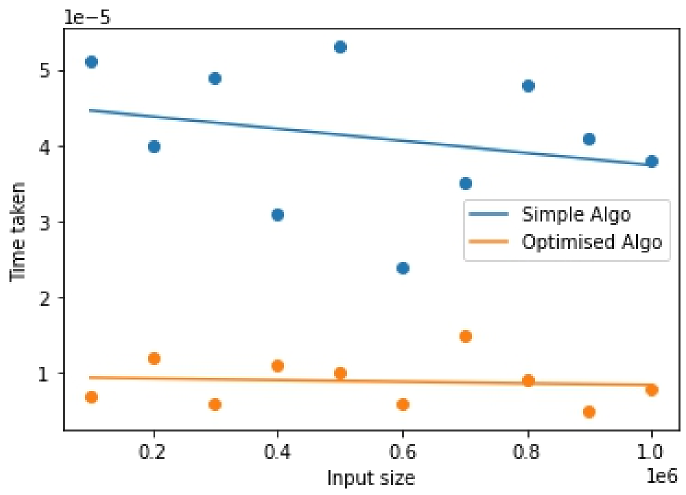
```
#include <stdio.h>

#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)


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



```
657348764
0
Time taken for simple algo: 0.000053
0
Time taken for sq root algo: 0.000009


...Program finished with exit code 0
Press ENTER to exit console.
```

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.**

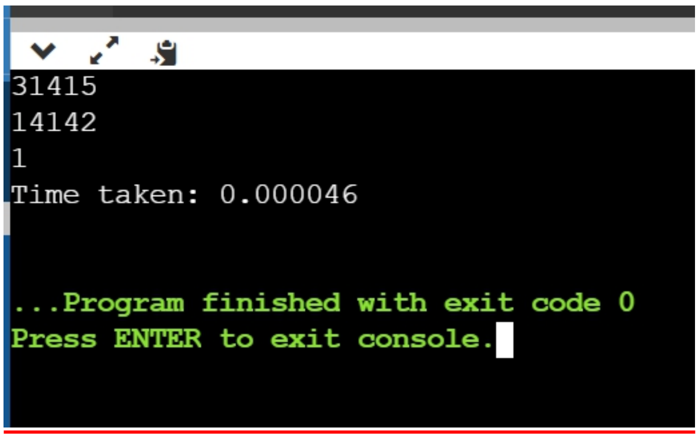
Write the above data in the following format and draw the graph.

Sl. No.	Input GCD (x, y)	GCD Algorithm - Time complexity
		Using Euclid’s Algorithm
1	GCD (31415, 14142)	0.000054
2	GCD (56, 32566)	0.000068
3	GCD (34218, 56)	0.000052
4	GCD (12, 15)	0.000060
5	GCD (31415, 31415)	0.000046
6	GCD (12, 12)	0.000039

Program

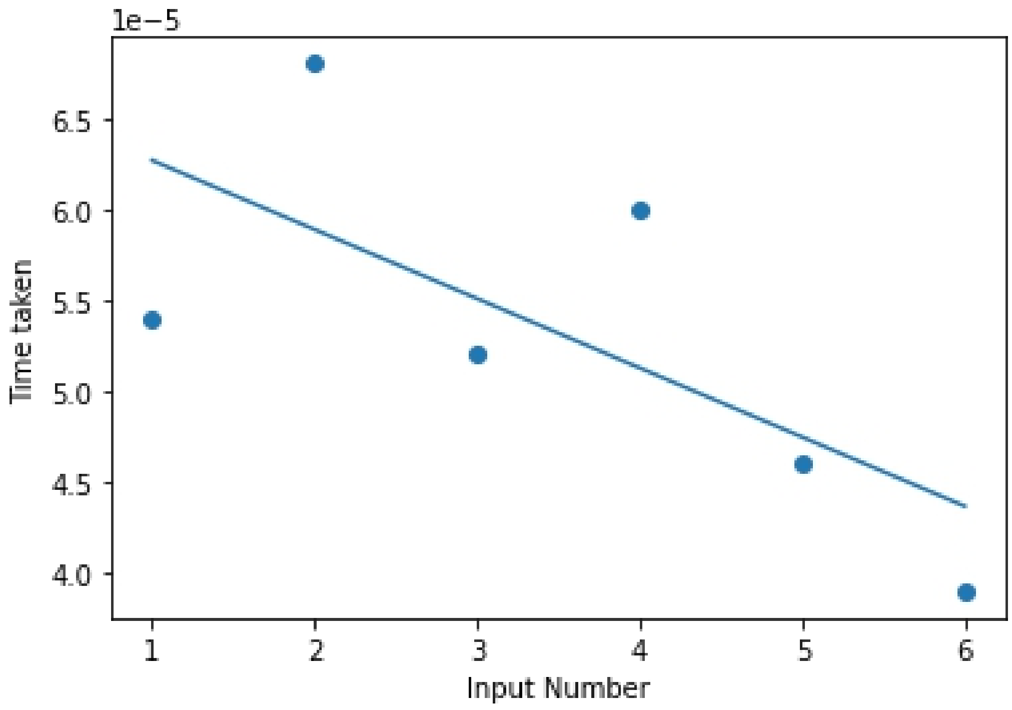
```
#include <stdio.h>

#include <time.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 gcd(int n, int m) {
    while (n > 0) {
            int r = m % n;
            m = n;
            n = r;
        }
        return m;
}

int main() {

        c_();

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



        start = clock();
        pfn(gcd(n, m));
        end = clock();
        double t = end - start;
        printf("Time taken: %f\n", (t / CLOCKS_PER_SEC));

        return 0;
}
```

Output



```
31415
14142
1
Time taken: 0.000046


...Program finished with exit code 0
Press ENTER to exit console.
```

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	0.000024	0.041080	0.021093
2	10000	0.000046	0.162470	0.082199
3	15000	0.000125	0.483918	0.236109
4	20000	0.000090	0.669699	0.326679
5	25000	0.000128	1.067751	0.534011
6	30000	0.000134	1.451958	0.732054
7	35000	0.000159	1.996108	0.997238
8	40000	0.000246	2.591937	1.283326
9	45000	0.000249	3.527495	2.105550
10	50000	0.000255	4.071260	2.037928

Program

```
#include <stdio.h>

#include <stdlib.h>

#include <time.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

}

void insertionSortAsc(int arr[], int n)

{

    int key, j;

    f(i, 1, n) {

        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];
            j = j - 1;
        }

        arr[j + 1] = key;
    }
}

void insertionSortDesc(int arr[], int n)
{
    int key, j;

    f(i, 1, n) {

        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] < key) {

            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
```

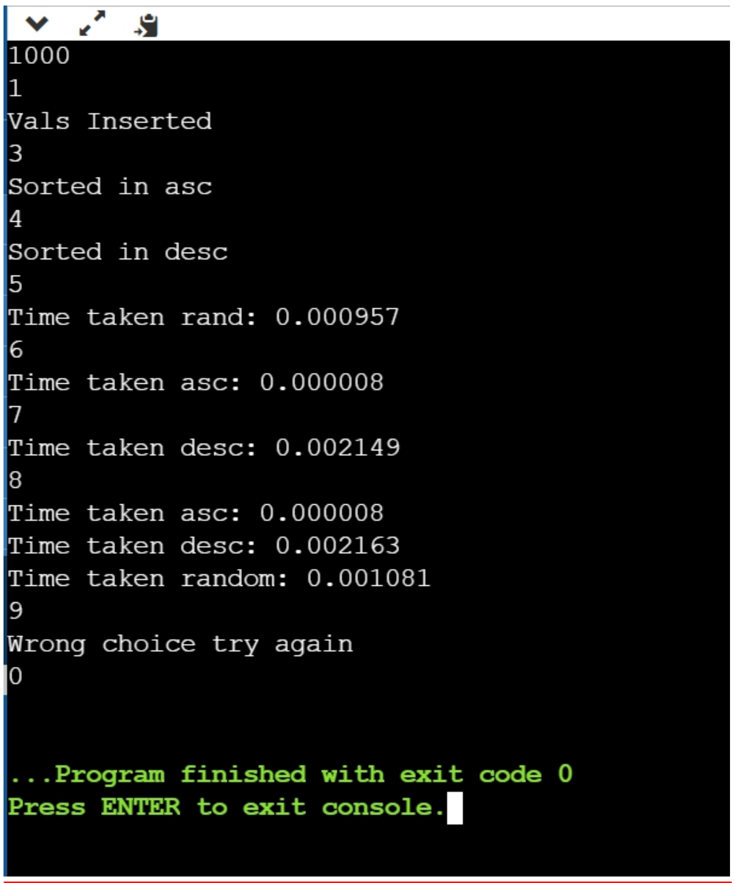
```
}
void printArray(int arr[], int n)
{
    int i;

    f(i, 0, n) {
            pfs(arr[i]);
    }
    pf("\n");
}

int main() {

    c_();

    time_t start, end;
    double t;
    int n;
    sf(n);
    int arr[n];

    insertionSortAsc(arr, n);

    int ch = 1;

    while (ch) {

            sf(ch);
```

```
switch (ch) {
case 0:
        break;
case 1:
        f(i, 0, n) {
                arr[i] = 1 + rand() % 100;
        }
        pf("Vals Inserted\n");
        break;
case 2:
        printArray(arr, n);
        break;
case 3:
        insertionSortAsc(arr, n);
        pf("Sorted in asc\n");
        break;
case 4:
        insertionSortDesc(arr, n);
        pf("Sorted in desc\n");
        break;
case 5:
        f(i, 0, n) {
                arr[i] = 1 + rand() % 100;
        }
        start = clock();
        insertionSortAsc(arr, n);
```

```
            end = clock();


            t = end - start;
            printf("Time taken rand: %f\n", (t / CLOCKS_PER_SEC));


            break;
        case 6:
            f(i, 0, n) {
                arr[i] = 1 + rand() % 100;
            }
            insertionSortAsc(arr, n);


            start = clock();
            insertionSortAsc(arr, n);


            end = clock();


            t = end - start;
            printf("Time taken asc: %f\n", (t / CLOCKS_PER_SEC));


            break;
        case 7:
            f(i, 0, n) {
                arr[i] = 1 + rand() % 100;
            }
            insertionSortDesc(arr, n);


            start = clock();
        insertionSortAsc(arr, n);

        end = clock();

        t = end - start;
        printf("Time taken desc: %f\n", (t / CLOCKS_PER_SEC));

        break;

    case 8:

        f(i, 0, n) {
            arr[i] = 1 + rand() % 100;
        }
        insertionSortAsc(arr, n);

        start = clock();
        insertionSortAsc(arr, n);

        end = clock();

        t = end - start;
        printf("Time taken asc: %f\n", (t / CLOCKS_PER_SEC));



        f(i, 0, n) {
            arr[i] = 1 + rand() % 100;
        }
```

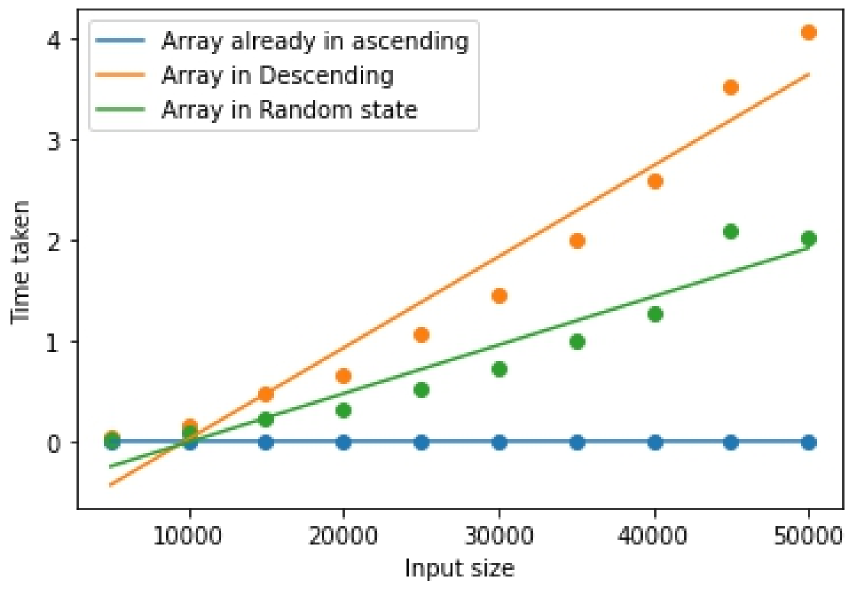
```
                insertionSortDesc(arr, n);

                start = clock();
                insertionSortAsc(arr, n);
                end = clock();
                t = end - start;
                printf("Time taken desc: %f\n", (t / CLOCKS_PER_SEC));



                f(i, 0, n) {
                        arr[i] = 1 + rand() % 100;
                }
                start = clock();
                insertionSortAsc(arr, n);

                end = clock();

                t = end - start;
                printf("Time taken random: %f\n", (t / CLOCKS_PER_SEC));
                break;

        default:
                pf("Wrong choice try again\n");
        }
    }

    return 0;
}
```

Output



```
1000
1
Vals Inserted
3
Sorted in asc
4
Sorted in desc
5
Time taken rand: 0.000957
6
Time taken asc: 0.000008
7
Time taken desc: 0.002149
8
Time taken asc: 0.000008
Time taken desc: 0.002163
Time taken random: 0.001081
9
Wrong choice try again
0


...Program finished with exit code 0
Press ENTER to exit console.
```

Graph


Array already in ascending
Array in Descending
Array in Random state
Time taken
Input size