Assignment #1

Que 1:- Write a program to generate the n unique random numbers by two different algorithms.

Algorithm:-

Using linear congruential generator:

Input:- Integer n,

Output:- N unique random numbers.

Initialize A=5,C=3,rand_MAX=8

RANDM()

    Initialize prev=1

    prev=A*prev+(C%rand_MAX)

RETURN prev

END ALGORITHM

Algorithm analysis:-

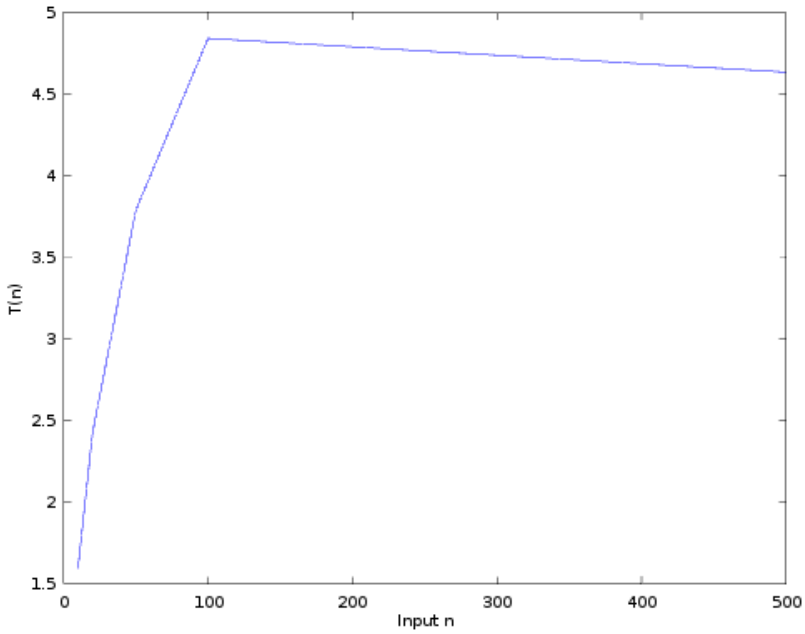
- LCGs are fast and require minimal memory (typically 32 or 64 bits) to retain state. This makes them valuable for simulating multiple independent stream
- T(n)=O(n)

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
#include <time.h>
#include <cmath>
using namespace std;
int A=5,C=3,rand_MAX = 8;
double randm()
{
   static int prev = 1;
   prev = A * prev + (C % rand_MAX);
   return prev;
}

int main()
{
//Approach 1: Using time function
   int n;
   time_t sec;
   sec=time(NULL);
 //   cout<<sec<<endl;
   cout<<"Enter N"<<endl;
   cin>>n;
   cout<<"N Random Numbers are:-"<<endl;
   for(int i=0;i<n;i++)
      {
         sec=sec%3600;
         printf("%ld\n",sec*i);
         sec=time(NULL);
      }

//Approach 2 :Using Linear congruential generator
      cout<<"N Random Numbers are:-"<<endl;
```

```
        for(int i=0; i<n; i++)
          cout <<randm()<< endl;

    return 0;
}
```

Graph:-



Input/Output:-

Input(n)	T(n)
10	1.597
20	2.422
50	3.789
100	4.842
500	4.635

Conclusion/Remark:-

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 2:- There is an array A of integers of size n which cannot directly be accessed. However, you can get true or false response to queries of the form A[i] < A[j]. It is given that A has only one duplicate pair, and rest all the elements are distinct. So it has n-1 distinct elements and 1 element which is same as one of the n-1 elements. Your task is to identify the indices of the two identical elements in A.

Algorithm:-
Input:- Integer n,
Array arr of n integers.
Output:- Index of duplicate element in the array.

Assumptions:- unordered map in which key represents element and value represents frequency of that element.

```
unordered_map<int,int>ump
i=0
FOR i=0 to n
        ump[arr[i]]++
END FOR
FOR key in ump
        IF ump->value==2
                index=ump->key
                print index
                break
        END IF
END FOR
END ALGORITHM
```

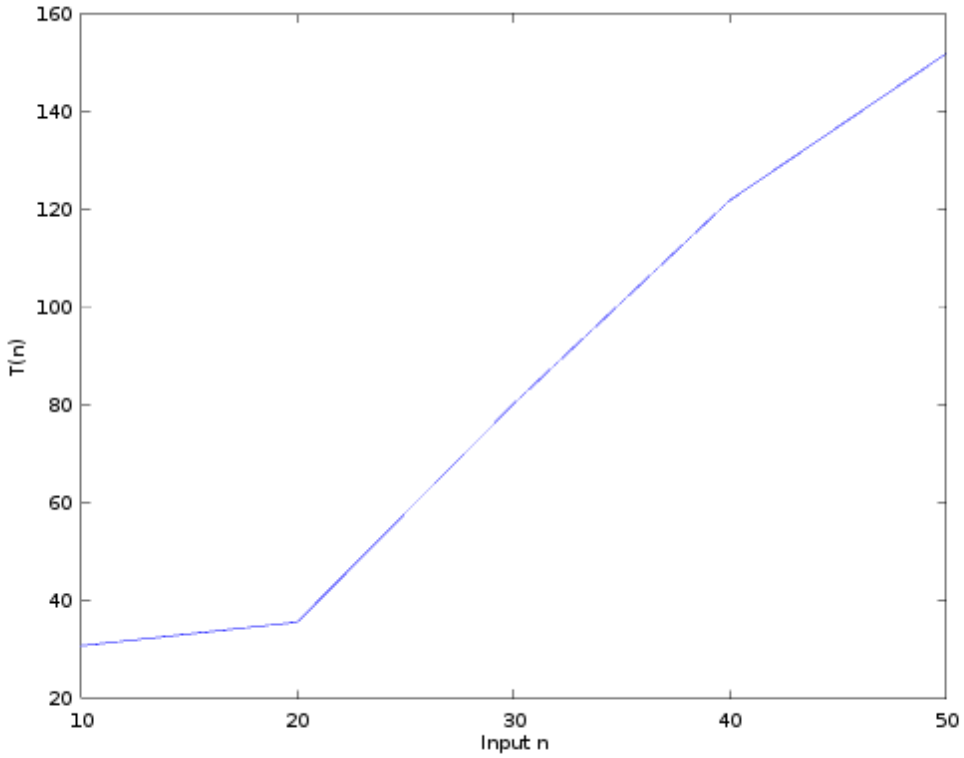
Algorithm analysis:-

- Unordered map has insertion time and searching time O(1) because it's implementation is hash table.
- Therefore complexity of this approach is O(n)
- $T(n)=C_1*n=O(n)$

Code:-

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main(){
        int d,x,n;
        unordered_map <int,int> ump(100);
        cout<<"Enter n"<<endl;
        cin>>n;
        int arr[n];
        unordered_map<int ,int>::iterator it;
        cout<<"Enter n elements"<<endl;
        for(int i=0;i<n;i++){
                cin>>x;
                arr[i]=x;
                ump[x]++;
                it=ump.find(x);
                if(it->second==2)
                        d=x;
        }
        cout<<"Index of element "<<d<<"is :"<<endl;
        for(int i=0;i<n;i++){
//              cout<<arr[i]<<" "<<d<<endl;
                if(arr[i]==d)
                        cout<<i<<" ";
        }
        cout<<endl;
        return 0;
}
```

Graph:-



Input/Output:-

Input(n)	output	T(n)
10	0 2	30.810
20	17 19	35.65
30	26 27	80.29
40	9 20	121.905
50	12 14	151.891

Conclusion/Remark:-

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 3:-Write a function customSort takes a "num" array of integers as an input along with the array of "weights" which contains the weights of the corresponding integers. We wish to sort this vector "num" from the indices start to end based on the fact that numbers with higher weights
appear first and in case the weights are equal, we put the greater number first in the list. We also wish efficient algorithms. We expect an O (n*log(n)) algorithm here instead of a O(n^2) one.

Algorithm:-
Input:- Integer n,
n integers along with its weight w.

Output:- list of number with weights in decreasing order.
Assumptions:- Let multiset of pair of number with its weight.

```
multiset<pair<int,int>> A
i=0
FOR i=0 to n
        A.insert(weight[i],num[i]) //insert in increasing order
END FOR
FOR every index in A  // print from the end
        print A->num and A->weight
END FOR
END ALGORITHM
```

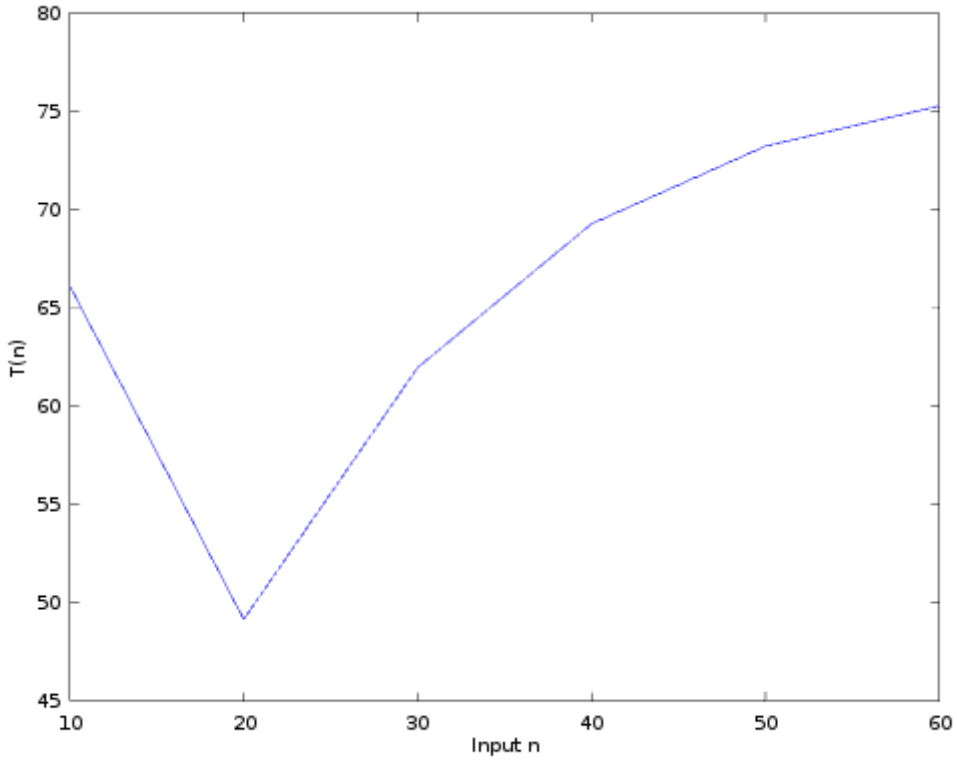
Algorithm analysis:-

- Multisets are implemented as binary search tree. Therefore, insertion cost for one element is O(logn).
  For n elements, total cost is O(n*logn).
- Linear in time if the elements are already sorted.

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
using namespace std;

int main(){
        int n,num,weight;
        multiset< pair<int, int> >a;
        cout<<"Enter n"<<endl;
        cin>>n;
        cout<<"Enter n elements along with its weight"<<endl;

        for(int i=0;i<n;i++){
                cin>>num>>weight;
                a.insert(pair<int,int>(weight,num));
        }

        multiset<pair<int,int>>::reverse_iterator it;
        cout<<"Number\tWeights"<<endl;
        for(it=a.rbegin();it!=a.rend();it++){
                cout<<it->second<<"\t"<<it->first<<endl;
        }
}
```

Graph:-



Input/Output:-

Input(n)	T(n)
10	66.116
20	49.157
30	61.968
40	69.282
50	73.239
60	75.282

Que:-4 Write a program to find the value of a b , where ‘a’ and ‘b’ are the real numbers using 3 different algorithms.

Algorithm:-
Input:- real numbers a and b.
Output:- a to the power b (a^b).

By Newton rapson method:-

```
NewtonRapson(a,b):
        initialize pre=1E-10
        initialize c=1/b,x0=1,x1
        WHILE TRUE:
                x1=x0-x0*b*(1-a*(x0^(-c))
                IF absolute(x1-x0) < pre
                        BREAK
                ELSE
```

```
                x0=x1
            END IF
        END WHILE
RETURN x1
END ALGORITHM
```

By Regula Falsi Method:-

```
RegulaFalsi(a,b):
        initialize pre=1E-10
        initialize c=1/b,x0=1,x1=2,x2
        WHILE TRUE:
            x2=x0-(x0^c-a)/(x1^c-x0^c)*(x1-x0)
            IF absolute(x2-x0) < pre
                BREAK
            ELSE
                x0=x2
            END IF
        END WHILE
RETURN x2
END ALGORITHM
```

By Successive Bisection Method:-

```
bisection(a,b):
        initialize pre=1E-10,c=1/b,x0=1,x1=2
        f0,f1
        WHILE TRUE:
            f0=(x0^c-a)
            f1=(x1^c-a)
            x2=(x1+x0)/2
            IF (f0*f1)<0
                x1=x2
            ELSE
                x0=x2
            END IF
            IF absolute(x2-x0) < pre
                BREAK
            END IF
        END WHILE
RETURN x2
END ALGORITHM
```

Algorithm analysis:-

- Recurrance Relation for Newton Rapson is:
  $x(k+1)=x(k) - f(k)/f'(k)$ where k=0,1,2,3 ....

- Recurrance Relation for Regula Falsi is:
  $x(k+1)=x(k) - (x(k) - x(k-1))/(f(k)-f(k-1))*f(k)$

- Recurrance Relation for Succesive bisection is:
  $x(k+1)=x(k) – f(k)/f'(k)$ where k=0,1,2,3 ....
  $x(k+1)=x(k) – (x(k) – x(k-1))/(f(k)-f(k-1))*f(k)$

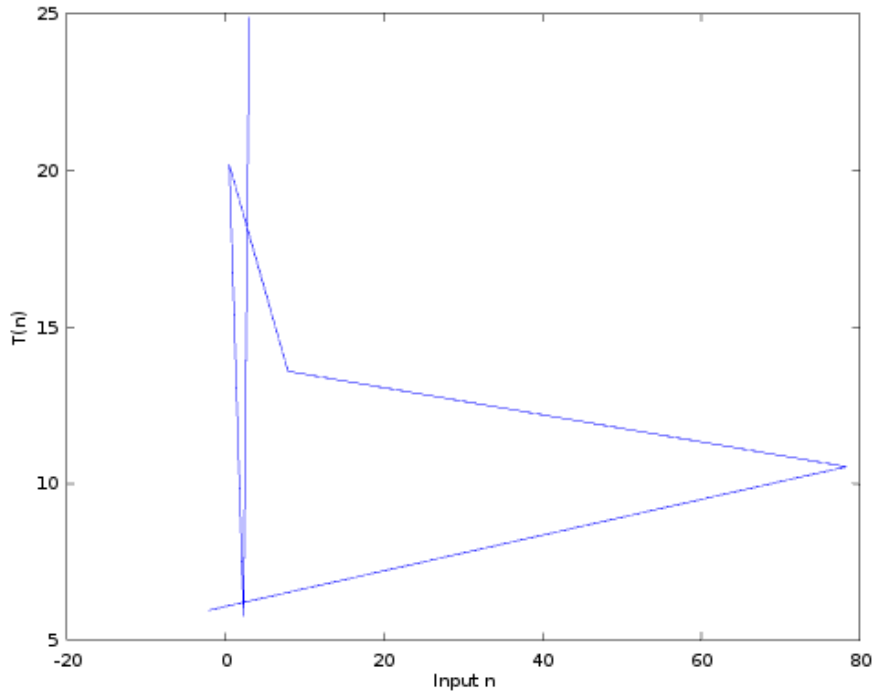
- Using Newton's method as described above, the time complexity with n-digit precision is O((logn)F(n)) where F(n) is the cost of calculating f(x)/f'(x) with n-digit precision.

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
#define MAX_SIZE 1000000
using namespace std;
int digits(int i)
{
   return i > 0 ? (int) log10((double) i) + 1 : 1;
}

double bisection(double a,double b){
        double EPS=1E-15 ,c=1/b,f0,f1,x2;
        double x0=1,x1=2;
        while(1){

                f0=pow(x0,c)-a;
                f1=pow(x1,c)-a;
                x2=(x1+x0)/2;
                if(f0*f1<0)
                        x1=x2;
                else
                        x0=x2;
                if(abs(x1-x0)<EPS)
                         break ;
                }
        return x2;
}

double regulaFalsi(double a,double b){
        double EPS = 1E-15 ,c=1/b;
        double x0=1,x1=2,x2;
        while(1){
                x2=x0-(pow(x0,c)-a)/(pow(x1,c)-pow(x0,c))*(x1-x0);
                if(abs(x2-x0)<EPS)
                         break ;
                x0=x2;
                }
        return x2;
}

double newtonRapson(double a,double b){
        double EPS = 1E-15 ,c=1/b;
        double x0=1,x1;
        while(1){
                x1=x0-x0*b*(1-a*pow(x0,-(int)c));
                if(abs(x1-x0)<EPS)
                         break ;
```

```
                x0=x1 ;
                }
        return x1;
}
int main()
{
        double n,x,a,b,ans=1,flag=0;
        cin>>a>>b;
        if(b<0.0){
                flag=1;
                b=abs(b);
        }
        int     d=digits((int)b);
        d=pow(10,d);

// Approach 1: By Newton Rapson Method
        if(b<1.0)
                ans=newtonRapson(a,b);
        else{
                if(b==(b-(int)b/d))
                        ans=pow(a,b);
                else
                        ans=ans*newtonRapson(a,b-b/d);
        }
        if(flag==1)
                ans=1/ans;
        printf("By Newton Rapson Method : a to the power b is %.15lf\n",ans);

// Approach 2: By Regula Falsi Method
        ans=1;
        if(b<1.0)
                ans=regulaFalsi(a,b);
        else{
                if(b==(b-(int)b/d))
                        ans=pow(a,b);
                else
                        ans=ans*regulaFalsi(a,b-b/d);
        }
        if(flag==1)
                ans=1/ans;
        printf("By Regula Falsi Method : a to the power b is %.15lf\n",ans);

// By Iterative Bisection
        ans=1;
        if(b<1.0)
                ans=bisection(a,b);
        else{
                if(b==(b-(int)b/d))
                        ans=pow(a,b);
                else
                        ans=ans*bisection(a,b-b/d);
        }
        if(flag==1)
                ans=1/ans;
        printf("By Iterative bisection Method : a to the power b is %.15lf\n",ans);
        return 0;
}
```

Graph:-



Input/Output:-

Input(a,b)	output(a^b)	T(n)
12, 3	1728	24.894
15.22, 2.3	524.2684	5.770
30.12, 0.5	5.4881	20.172
4.5 , 7.9	144670.1257	13.585
150 ,78.3		10.541
6,-2.2	0.0194	5.956

Conclusion /Remark**:-**

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que:-5 Write a program to find the factorial of the given number
(0 < n < 10,000,000,000)

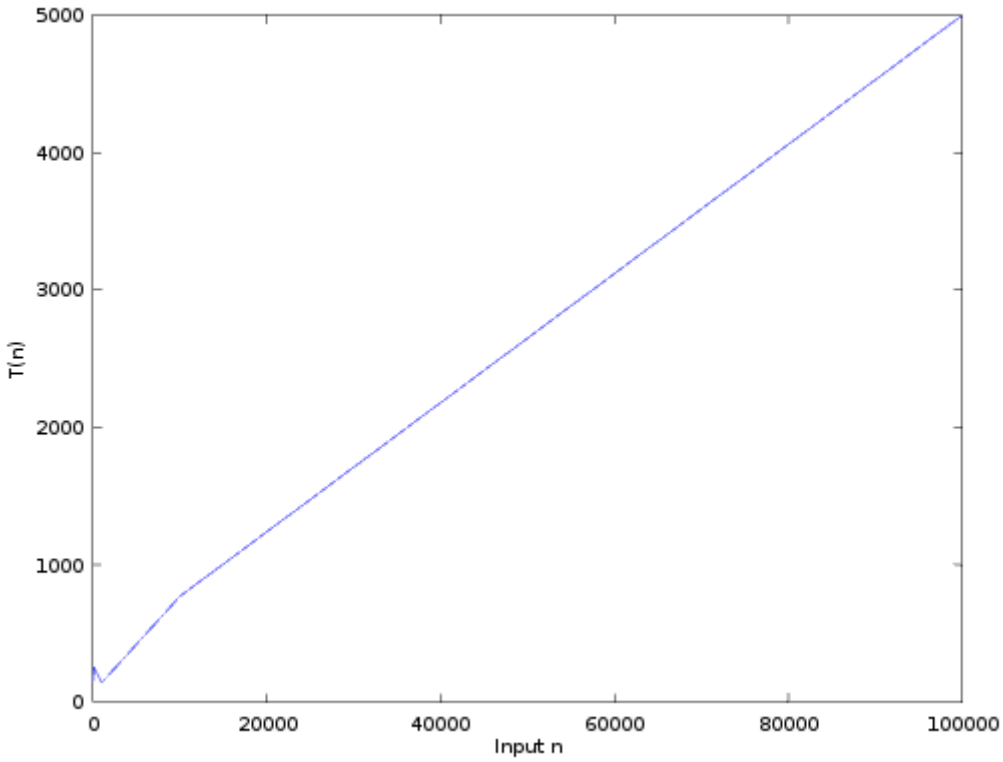
Algorithm:

```
FACTORIAL(n)
    Input: integer n
    Output: factorial of n
    Initialize fact=1
    FOR i=2 to n
        fact=fact*i
    END FOR
RETURN fact
END ALGORITHM
```

Code:-

```
import java.util.*;
import java.lang.*;
import java.io.*;
import java.math.*;
class Facorial
{
	public static void main(String args[])
	{
		BigInteger fact= BigInteger.ONE;
		System.out.println("Enter any number");
		int n = new Scanner(System.in).nextInt();

		for (int i = 2; i <= n; i++)
		  {
		    fact = fact.multiply(new BigInteger(String.valueOf(i)));
		  }

	    System.out.println("The factorial of " + n +" is: " + fact);
	 }
}
```

Graph:-



Algorithm Analysis:-

- Considering that multiplication is $O(1)$,i.e. $C_1=1$ then
  $T(n)=C_1n=O(n)$

Input/Output:-

Input(n)	T(n)
10	150.04
50	150.02
100	260.172
1000	140.585
10000	770
100000	5000

Conclusion /Remark**:-**

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 6:- Write a program to find the n th Fibonacci number with two different approach and compare their time complexity.

Algorithm:-

Using Dynamic programing:-

input:- Integer n

output:- nth fibonocci number

```
Initialize dp[1]=0,dp[2]=1
FOR i=3 to MAX_SIZE
        dp[i]=dp[i-1]+dp[i-2]
END FOR
print dp[n]
```

Using Recursion:-

```
FIBONOCCI(n)
        Input: integer n
        Output: n th Fibonacci number
        Assumption: First two Fibonacci Numbers are 0 and 1.
        IF n < 2 THEN
                RETURN n // Fibonacci(0) = 0 and Fibonacci(1) = 1
        ELSE
                RETURN Fibonacci(n-1) + Fibonacci(n-2)
        END IF
END ALGORITHM
```

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
#define MAX_SIZE 1000000
using namespace std;
```

```
long long int dp[MAX_SIZE];
int main(){
	int n;
	cout<<"Enter any number"<<endl;
	cin>>n;
	//Approach 1: Using Dynamic programming
	dp[1]=0;
	dp[2]=1;
	for(int i=3;i<MAX_SIZE;i++){
		dp[i]=dp[i-1]+dp[i-2];
	}
	cout<<"Nth Fibonocci number is : "<<dp[n]<<endl;

	//Approach 2: Simple iterative
	long long int f1=0,f2=1,f3;
	for(int i=1;i<n-1;i++){
		f3=f1+f2;
		f1=f2;
		f2=f3;
	}
	cout<<"Nth Fibonocci number is : "<<f3<<endl;
}
```

Algorithm analysis:-

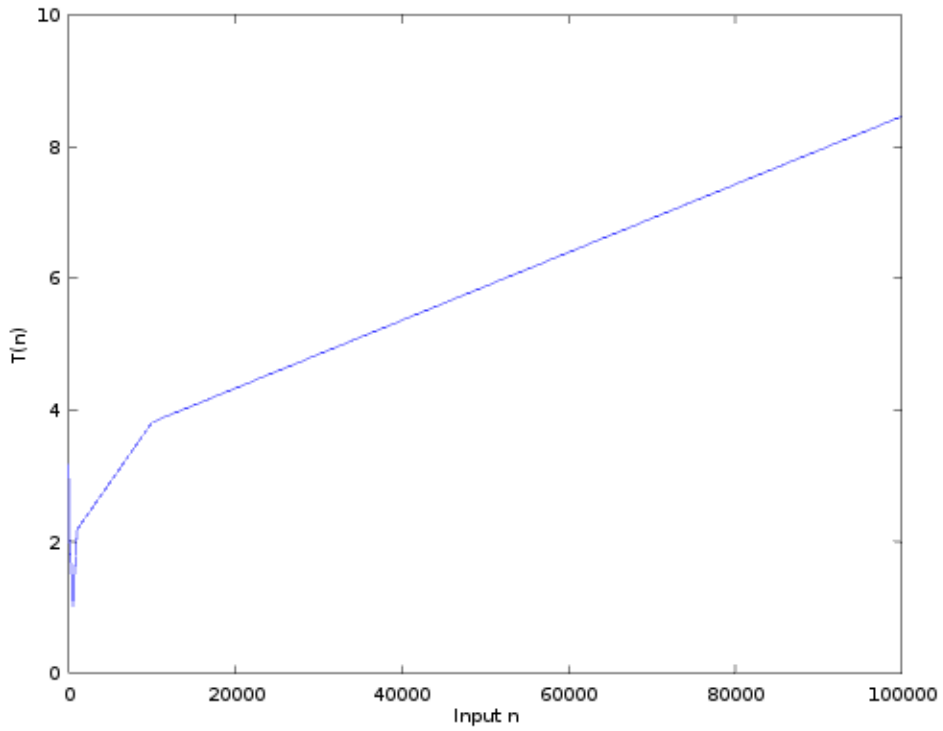
- For Recursive method

  $T(n) = T(n-1) + T(n-2) + 1 = 2^n = O(2^n)$

- For Dynamic programing

  Time Complexity: O(n) , Space Complexity :O(n)

Graph:-



Input/Output:-

Input(n)	T(n)
10	3.180
50	1.976
100	2.056
500	1.022
1000	2.182
10000	3.816
100000	8.461

Conclusion/Remark**:-**

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 7:-An array of integers is said to be a straight-K, if it contains K elements that are K consecutive numbers. For example, the array {6, 1, 9, 5, 7, 15, 8} is a straight because it contains 5, 6, 7, 8, and 9 for K=5. Write a program to finds the maximum value of K for the given number of integers.

Algorithm:-
Input:- Integer n and A[n]
output:- Maximum value of K.

```
count=0
SORT(A)
FOR i=0 to n
        IF (A[i+1]-A[i])==1
                count++
        ELSE
                ans=MAX(ans,count)
                count=0
        END IF
END FOR
PRINT ans+1
END ALGORITHM
```

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
#define MAX_SIZE 1000000
using namespace std;

int main(){
```

```
        int n,ans=0,c=0;
        cout<<"Enter number of elements"<<endl;
        cin>>n;
        int arr[n];
        cout<<"Enter n elements"<<endl;
        for(int i=0;i<n;i++){
                cin>>arr[i];
        }
        sort(arr,arr+n);
        for(int i=0;i<n;i++){
                if((arr[i+1]-arr[i])==1)
                        c++;
                else{
                        ans=max(ans,c);
                        c=0;
                }
        }
        ans++;
        cout<<"Maximum K is : "<<ans<<endl;
}
```

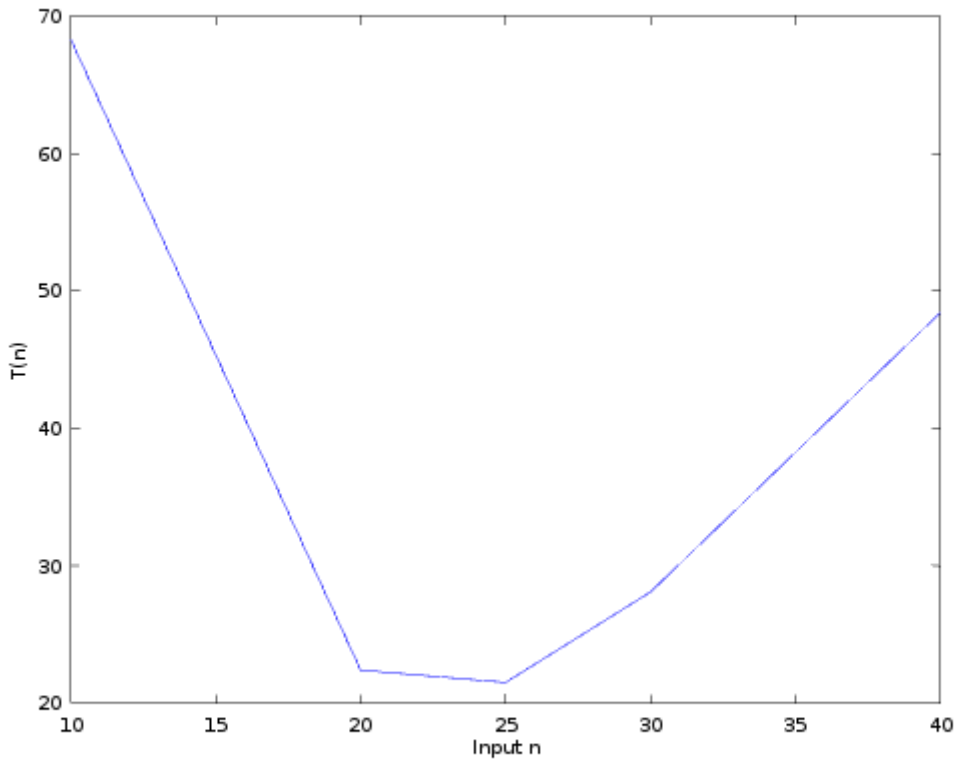
Algorithm Analysis:-

- For Sorting T(n)=O(nlogn)
- O(n) for consicutive searching max k
- Total T(n)=O(n)+O(nlogn)=O(nlogn).

Input/Output:-

Input(n)	T(n)
10	68.306
20	22.406
25	21.505
30	28.090
40	48.424

Graph:-



Conclusion/Remark**:-**

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 8:- You have an array like :-{ 5,5,5,13,6,13,6,7,8,7,-1) Write and analyze an algorithm to arrange this array in sorted form based upon the number of occurrence e.g.: above should look like this after execution of your also {5,5,5,13,13,6,6,7,7,8,-1) 13 comes before 6 because it has same number of occurrence as 6 but it come first in the parent array.

Algorithm**:-**
Input **:-** Integer n
array arr[] of n elements
Output**:-** an array of n integers based on question.
Assumptions:- map<int,int> mp contains number and its frequency.
vector<int> s[MAX_SIZE] for maintain the order of elements.

```
FOR x in arr
    mp[x]++;
END FOR
ma=0
FOR i=0 to n
    it=mp.find(arr[i])
```

```
        s[it->value].push_back(it->first)
        ma=max(ma,it->second)
END FOR
FOR j=ma to 0
        FOR i=s[j].begin() to s[j].end()
                FOR k=0 to j
                        print *i
                END FOR
        END FOR
END FOR
END ALGORITHM
```

Code:-

```
#include <iostream>
#include <set>
#include <map>
#include <vector>
#define MAX_SIZE 1000
using namespace std;

        map<int,int> mp;
        int arr[MAX_SIZE];
        vector<int>s[MAX_SIZE];
        int ma;

int main(){
        int n,x,k=0,j,flag=0;
        cin>>n;
        for(int i=0;i<n;i++){
                cin>>x;
                for(j=0;j<n;j++){
                        if(arr[j]==x)
                                break;
                }
                if(j==n)
                   arr[k++]=x;
                mp[x]++;
        }

        map<int,int>::iterator it;
        for(int i=0;i<k;i++){
                cout<<arr[i]<<" ";
                it=mp.find(arr[i]);
                s[it->second].push_back(it->first);
                //cout<<it->first<<" "<<it->second<<endl;
                ma=max(ma,it->second);
        }
//        cout<<endl;
        for(int j=ma;j>=0;j--){
             for(vector<int>::iterator i=s[j].begin();i!=s[j].end();i++)
                {
                        for(int k =0;k<j;k++)
                                cout<<*i<<" ";
                }
        }
        return 0;
}
```

Algorithm Analysis:-

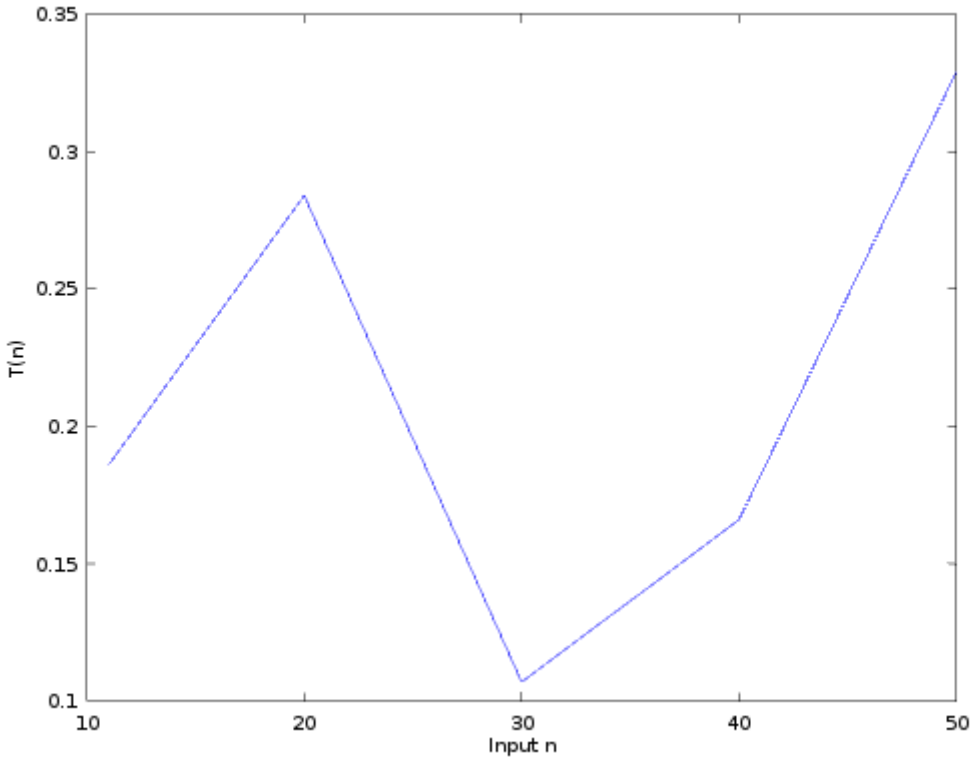
$T(n) = C_1n^2+C_2n$

$= O(n^2)$

Input/Output**:-**

Input(n)	T(n)
11	0.186
20	0.284
30	0.107
40	0.166
50	0.329

Graph**:-**



Conclusion /Remark:-

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que 9:- Write a program to find the determinant of the given matrix of size NxN using recursion.

Algorithm**:-**
Input:- n integer

output:- Determinant of n*n matrix.

Assume A[n][n] be the matrix .

Cofactor(A,i,j): Calculates the cofactor of A_{ij}

Determinant(A):
IF size of A is 1
RETURN A[0][0]
END IF
initialize D to 0
FOR i=1 to N
D=D+A[0][i]xDeterminant(cofactor(A,0,i))
END FOR
RETURN D
END ALGORITHM

Code:-

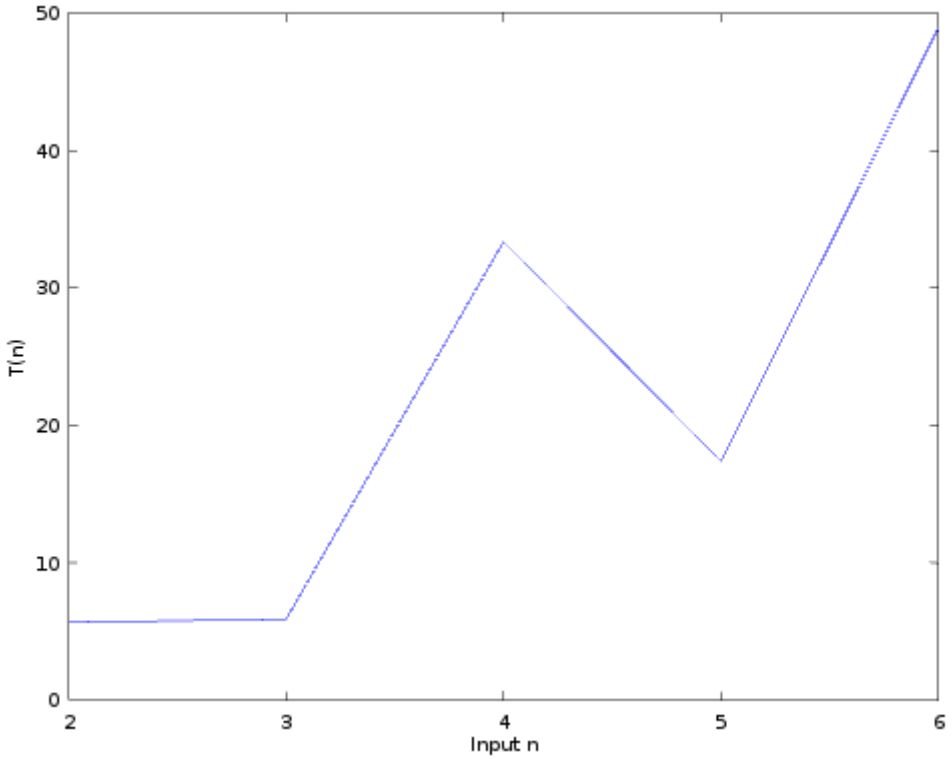
```
#include <iostream>
#include <set>
#include <algorithm>
#include <vector>
#include <map>
#define MAX_SIZE 50
using namespace std;
int determinant(int a[MAX_SIZE][MAX_SIZE],int n) {
int det=0,i,j,k,x,y,temp[MAX_SIZE][MAX_SIZE];
 if(n==2){
   det=(a[0][0]*a[1][1]-a[0][1]*a[1][0]);
   return det;
 }
 else{
   for(i=0;i<n;i++){
      x=0;
    for(j=1;j<n;j++){
      y=0;
       for(k=0;k<n;k++){
         if(k==i){
            continue;
```

```
            }
  //      cout<<a[i][j]<<" ";
          temp[x][y]=a[j][k];
          y++;
 //         cout<<y<<endl;
          }
          x++;
        }
//      cout<<det<<endl;
      det=det+a[0][i]*pow(-1 ,i)*determinant(temp,n-1);
    }
    return det;
  }
}

int main(){
  int n;
  cout<<"Enter N"<<endl;
  cin>>n;
  int arr[MAX_SIZE][MAX_SIZE];
  cout<<"Enter N*N matrix"<<endl;
  for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
      cin>>arr[i][j];
    //  cout<<arr[i][j]<<" ";
    }
    //cout<<endl;
    }
  int ans=determinant(arr,n);
  cout<<"Determinant of given Matrix is : "<<ans<<endl;
  return 0;
}
```

Algorithm Analysis:-

$T(n)= C_1n^2 + T(n-1) = O(n^3)$.

Graph:-



Conclusion /Remark:-

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.

Que:-10 Write a program to display the elements of given matrix of size NxN in clockwise spiral order using recursion.

Algorithm**:-**

Input:- n integer

output:- n*n matrix in spiral order.

Assume A[n][n] be the matrix .

```
Spiral(A,i):
        IF i==n/2                   //central cell of matix
                RETURN
        ELSE
                PrintOneRound(A,i,i);
        END IF
        Spiral(A,i+1)
END ALGORITHM
```

Code:-

```
#include <iostream>
#include <set>
#include <algorithm>
#include <unordered_map>
#include <vector>
#include <map>
#define MAX_SIZE 100
using namespace std;
	int n,c=0,nsquare;
	int arr[MAX_SIZE][MAX_SIZE];
	void PrintOneRound(int arr[MAX_SIZE][MAX_SIZE],int x,int y)
	{
		for(int j=y;j<n-x;j++)
		{
			if(c>=nsquare)
				break;
			cout<<arr[x][j]<<"  ";
			c++;

		}
		for(int j=x+1;j<n-y;j++)
		{
			if(c>=nsquare)
				break;
			cout<<arr[j][n-x-1]<<"  ";
			c++;
		}
		for(int j=n-y-2;j>=x;j--)
		{
			if(c>=nsquare)
				break;
			cout<<arr[n-x-1][j]<<"  ";
			c++;
		}
		for(int j=n-x-2;j>y;j--)
```

```
                {
                        if(c>=nsquare)
                                return;
                        cout<<arr[j][x]<<"  ";
                        c++;
                }

        }
void Spiral(int i){
        if(c<nsquare)
                PrintOneRound(arr,i,i);
        else
                return;
        Spiral(i+1);
}
int main(){
        cout<<"Enter N"<<endl;
        cin>>n;
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        cin>>arr[i][j];
                }
        }
        nsquare=n*n;
        cout<<"\nElements in Spiral order are :"<<endl;
        Spiral(0);
        return 0;
}
```

Algorithm Analysis:-

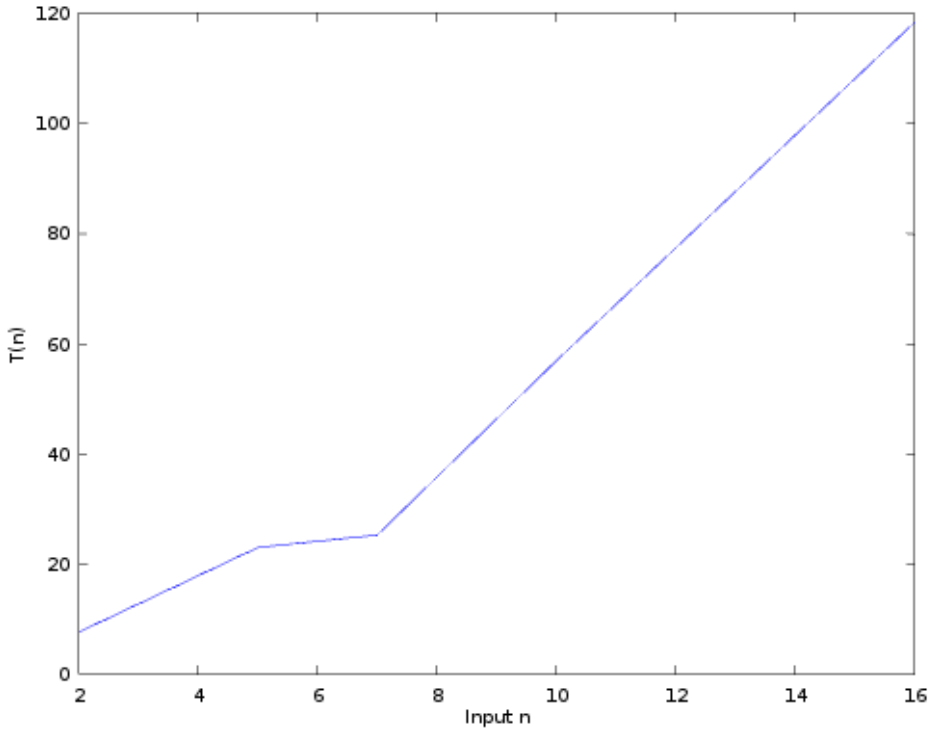
T(n)= 4*n + T(n-1)

Worst case complexity is $O(n^2)$.

Input/Output:-

Input(n)	T(n)
2	7.667
5	23.043
7	25.306
10	56.956
16	118.389

Graph:-



Conclusion /Remark:-

On practically examining the graph of the algorithm we find there is a variation from the thereotical complexity graph as there might be various factors affecting the execution of program like the memory ,cpu,etc.