Recursion

- Recursion is the technique which allows us to break down the problem into one or more similar sub problems that are similar in form to the original problem.
- A function which calls in terms of itself is known as recursive function and this process is known as recursion.
- The main advantage of recursion is to avoid the length of the code.

Recursion Advantages:

i. It is easily, simple and understandable.
ii. Using recursion, we can avoid unnecessary calling of functions.
iii. The recursion is very flexible in data structure
iv. Using recursion, the length of the program can be reduced.

Recursion Disadvantages:

i. It requires extra storage space. The recursive calls and automatic variables are stored on the stack.
ii. For every recursive call separate memory is allocated to automatic variables with the same name.
iii. If the programmer forgets to specify the exit condition in the recursive function, the program will execute out of memory.
iv. The recursion function is not efficient in execution speed and time.

Principle of Recursion:

For implementing and designing a good recursive program, we must assume certain assumption which are as follows

a) **Base Case:** Base case is the terminating condition for the problem while designing any recursive algorithm.

b) **If Condition:** If condition is the recursive algorithm that defines the terminating condition.

c) Every time a new recursive call is made, a new memory space is allocated to each automatic variable used by recursive routine.

d) Each time a recursive call is there, the duplicate values of the local variables of the recursive call are pushed on to the stack, within the respective call and all these values are available to the respective function call, when it is popped off from the stack.

e) **Recursive Case:** This is the else part of the recursive definition that calls the function recursively.

Example:

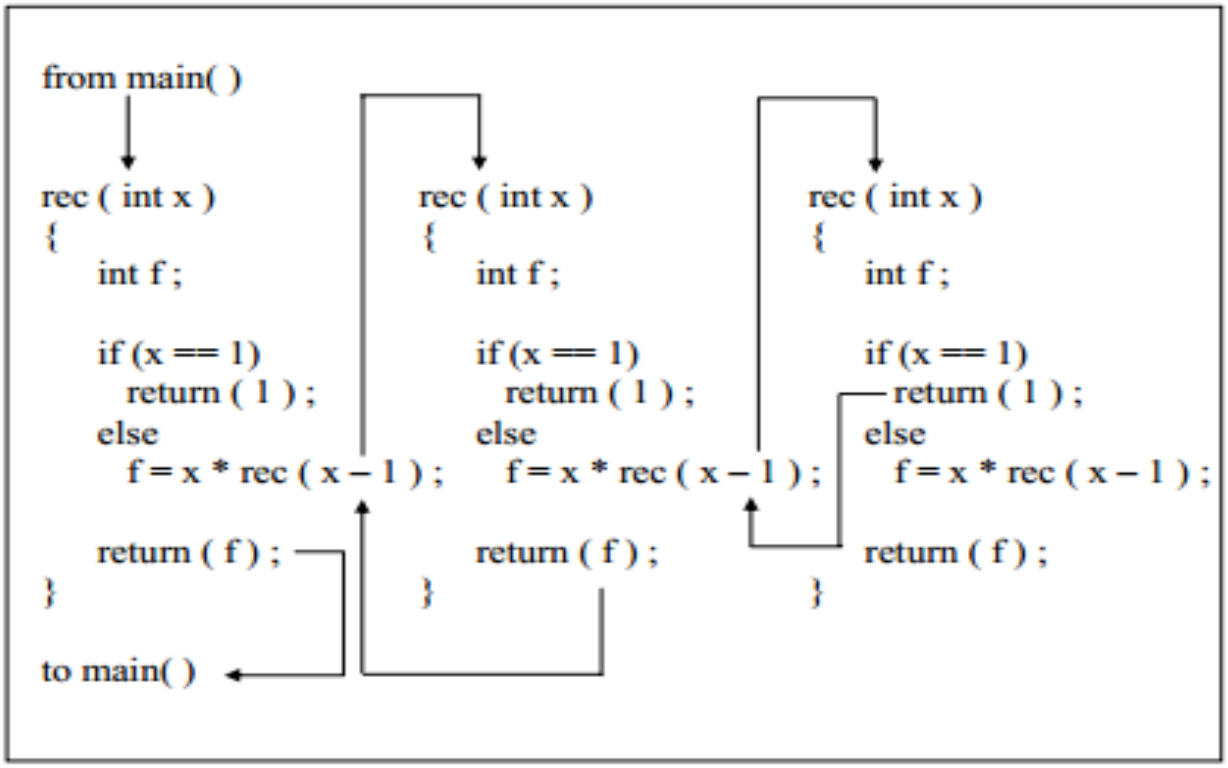
```
int factorial(int a)
{
int fact;
if(a==0)
return 1; // base case
else
fact= a* factorial(a-1); // recursive call
return fact;
}
```

Recursion Vs Iteration:

	Recursion	Iteration
Definition	Recursion refers to a recursive function in which it calls itself again to repeat the code.	Iteration is achieved by an iterative function which loops to repeat some section of the code.
Important point	A base case needs to be determined	A termination condition needs to be determined
Performance	Comparatively slow	Comparatively fast
Important point	A base case needs to be determined	A termination condition needs to be determined
Performance	Comparatively slow	Comparatively fast
Memory Usage	Comparatively more	Comparatively less
Code	Smaller	Longer
Infinite repetition	Infinite recursion is capable of crashing the system	Infinite looping consumes CPU cycles repeatedly
Structure	Selection	Repetition
Local variables	Not required	Required

//program to find the factorial of a number

```
#include<stdio.h>
#include<conio.h>
int factorial(int a);   //function Decleration
void main()
{
int a,fact;
printf("enter the value of a\n");
scanf("%d",&a);
fact=factorial(a);    //function call
printf("The factorial of %d is:%d",a,fact);
getch();
}
int factorial(int a)   //function definition
{
int fact;
if(a==0)
return 1;
else
fact= a* factorial(a-1);
return fact;
}
```



// c program to find the sum of first n natural numbers using recursion

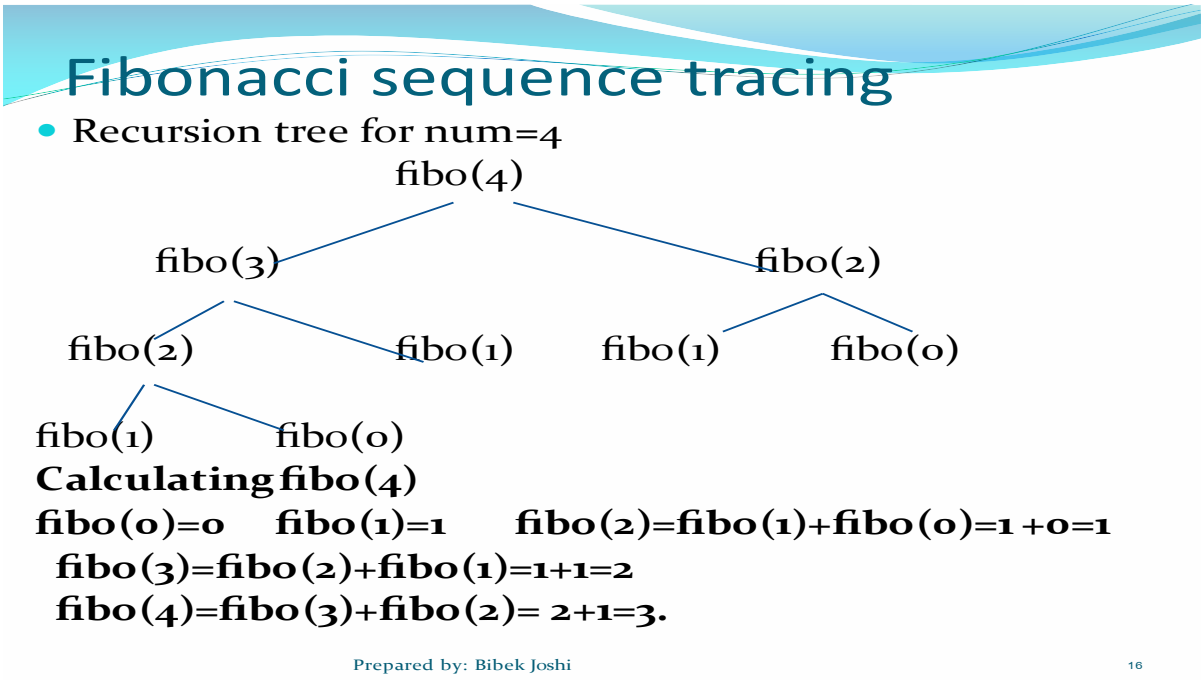
```
#include<stdio.h>
#include<conio.h>
int add(int num);
void main()
{
int num,sum;
printf("Enter a number\n");
scanf("%d",&num);
sum=add(num);
printf("sum of number from 1 to %d is: %d\n",num,sum);
getch();
}
int add(int num)
{
if(num==0)
return 0;
else
return num+add(num-1);
}
```

Fibonacci Sequence:

- A Fibonacci sequence is defined by mathematically as follows Fn=Fn-1+ Fn-2 where the initial values are f1=0,f2=1 or f1=1 ,f2=1.
- Recursive algorithm for finding the nth term of Fibonacci number.

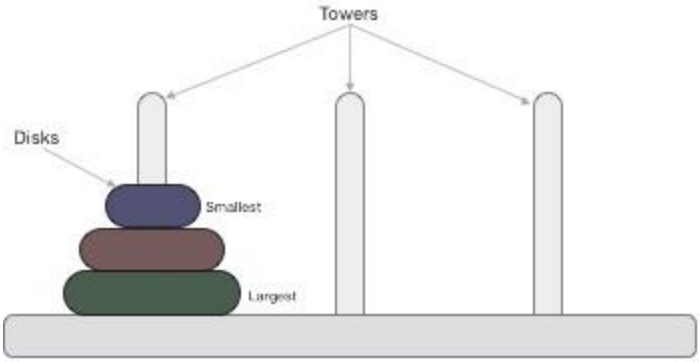
```
#include<stdio.h>
#include<conio.h>
int fibo(int num);
void main()
{
```

```
int num,k;
printf("Enter a number\n");
scanf("%d",&num);
k=fibo(num);
printf("the nth term of fibonacci sequence  is: %d\n",k);
getch();
}
int fibo(int num)
{
if(num==0)
return 0;
else if(num==1)
return 1;
else
return fibo(num-1)+fibo(num-2);
}
```



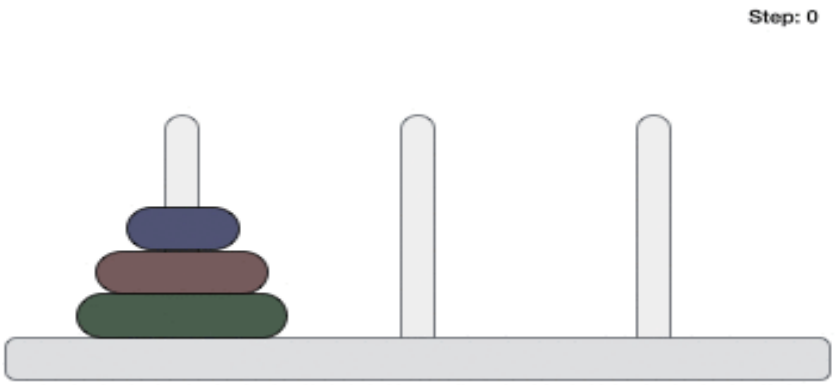
Tower of Hanoi:

Tower of Hanoi, is a mathematical puzzle that consists of three towers (pegs) and more than one rings is as depicted –



These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Following is an animated representation of solving a Tower of Hanoi puzzle with three disks.



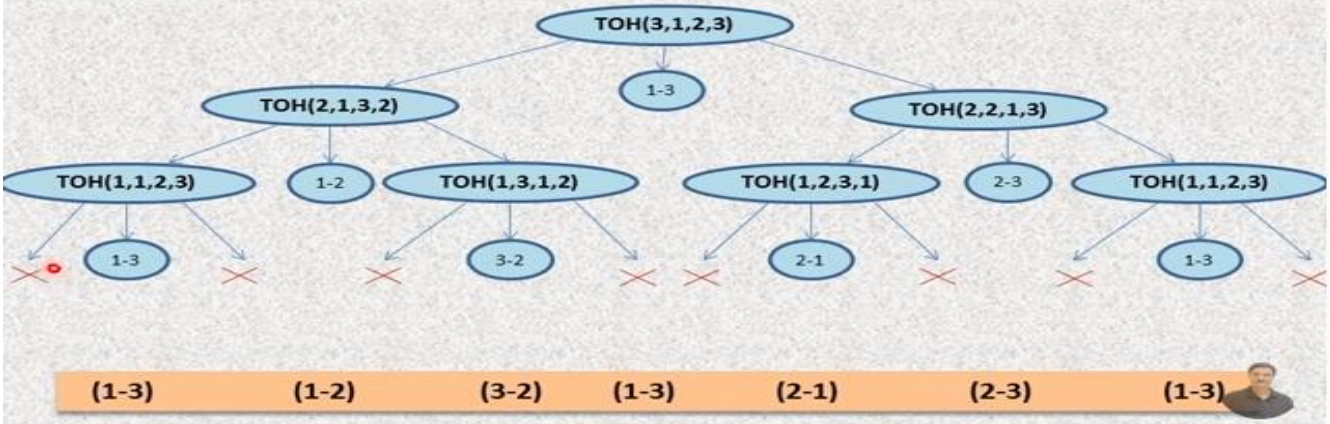
Rules:

- Move n-1 Discs from A to B using C
- Move a Disc from A to C
- Move n-1 Discs from B to C using A

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```

Tracing for 3 Discs

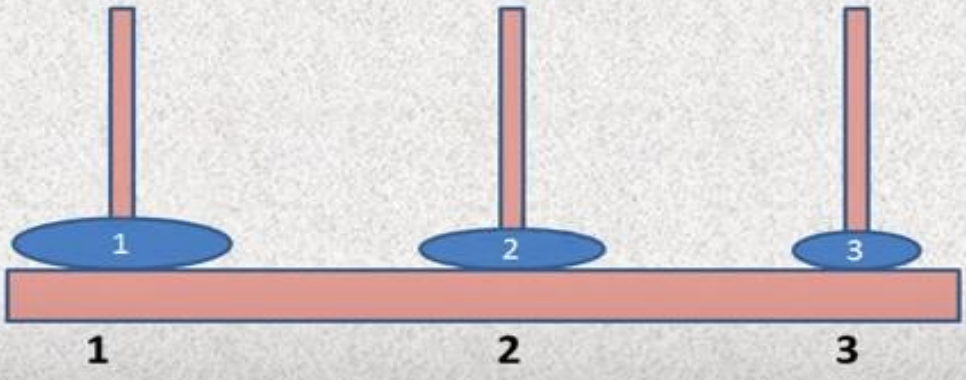
```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)



(3-2)

```c
/* Tower of Hanoi Program in C */
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
    int num;
    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');
    return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
    // Base Condition if no of disks are
    if (num == 1)
    {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    // Recursively calling function twice
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

```
C:\Users\Dell\Desktop\Untitled1.exe
Enter the number of disks : 3
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg C
 Move disk 2 from peg A to peg B
 Move disk 1 from peg C to peg B
 Move disk 3 from peg A to peg C
 Move disk 1 from peg B to peg A
 Move disk 2 from peg B to peg C
 Move disk 1 from peg A to peg C
--------------------------------
Process exited after 2.628 seconds with return value 0
Press any key to continue . . .
```

```c
/* Greatest Common Divisor Program in C */
#include <stdio.h>
int gcd(int, int);
int main()
{
    int x, y, GCD;
    printf("Enter the two numbers to find their GCD: ");
    scanf("%d%d", &x, &y);
    GCD = gcd(x, y);
    printf("The GCD Of The Given Numbers is %d.\n",  GCD);
}
int gcd(int x, int y)
{
    while (x != y)
    {
        if (x > y)
        {
            return gcd(x - y, y);
        }
        else
        {
            return gcd(x, y - x);
        }
    }
    return x;
}
```

```
C:\Users\Dell\Desktop\Untitled1.exe
Enter the two numbers to find their GCD: 8
12
The GCD Of The Given Numbers is 4.

--------------------------------
Process exited after 2.504 seconds with return value 35
Press any key to continue . . .
```