RECURSION:

- Recursive Algorithm
- Recurrence Relation
- Recurrence Relation Solving

1. A Recursion is calling itself is called Recursion to solve a particular problem.
2. Solving the bigger problem in terms of smaller problems

   Example: f(6) = 6 * f(5) = 6 * 5 * f(4) = 30 * 4 * f(3) = 120 * 3 * f(2) = 360 * f(1)
3. To execute the recursion program, we are using stack Data Structure.

1	f(1)
2 * f(1)	f(2)
3 * f(2)	f(3)
4 * f(3)	f(4)
5 * f(4)	f(5)
6 * f(5)	f(6)

Recurrence Relation

$$f(n) = \begin{cases} 1, \text{ if } n=1 \\ n * f(n-1) \text{ OTHERWISE} \end{cases}$$

4. Every recursive program should have termination condition otherwise it will get into infinite loop and at the end you will get message stack overflow.
5. In recursion, one function call to another function call . Number of parameters will not change, parameter's name will not change, code will not change but parameter values will change.
6. For every Recursive program, equal & non-equal recursive program is possible.
7. Recursive program takes more stack space because more function calls.

Recursive Program

```
fact(n) 1
{
 if(n==1) return (1)
 else
 return (n * fact(n-1)
}
```

Non-Recursive Program

```
f(n)
{
 for( i=1; i < = n ; i++ )
  {
     s = s * i ;
  }
  return(s)
}
```

Example 1: Write a Recursive C Program & Reurrence Relation to multiply 2 positive numbers m & n where m, n > 0

EXAMPLE: 4 * 7 = 7 + 7 + 7 + 7 …………..4 times

28

MUL(4,7) = 7 + MUL(3,7) 21

↓

7 + MUL(2,7) 14

↓

7 + MUL(1,7) 7

↓

7 + MUL(0,7)

RECURSIVE ALGORITHM:

```
MUL(m,n)
{
  if(m==0 || n==0)   ( only for Stopping  Condition)
  return (0)
  else                     ( Hero of Recursion )
  return (n + MUL(m-1,n))
}
```

Time Complexity = O(m)

Stack Complexity = m - stack slots

Recurrence Relation:

$$MUL(m,n) = \begin{cases} 0, \text{ if } m=0 \text{ or } n=0 \\ n + MUL(m-1,n) \text{ otherwise} \end{cases}$$

Example 2: Write a Recurrence program & Recurrence Relation to find the Fibonacci number.

Sample Input :

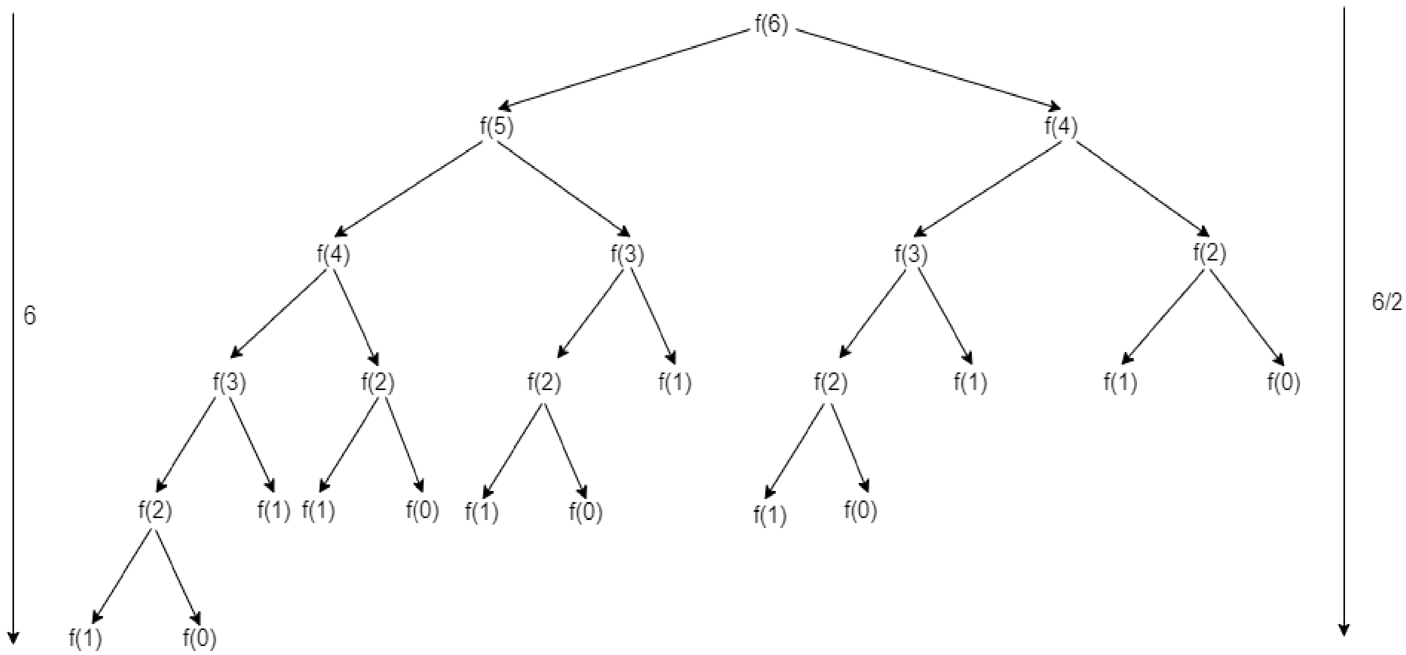
n	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
fib	**0**	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**

Recursive Relation:

$$f(n) = \begin{cases} 0, \text{ if } n=0 \\ 1, \text{ if } n=1 \\ fib(n-1) + fib(n-2) \text{ otherwise} \end{cases}$$

Recursive Program

```
fib(n)
{
  if  (n==0) return 0
  if  (n==1) return 1
  else
  return ( fib(n-1)  + fib(n-2) )
}
```



In Recursion,

Time Complexity = Number of Function calls

= n-level

= 2^n-1 nodes

= $O(2^n)$ [UPPER BOUND]

n-level Binary Tree contain (2^n-1) nodes

n-level Ternary Tree contain (3^n-1) nodes

There is a gap in Fibonacci binary tree that’s why we are taking big-oh with 2^n because exact complexity is less than 2^n so we took Big-oh

Space Complexity = Number of levels (n)

Example 3: Write a Recurrence program & Recurrence Relation to find the GCD OF m & n.

Sample Input : GCD(10,20) = 10 GCD(5,7) = 1 GCD(0,0) = undefined

GCD(0,10) = 10 Where one is zero other is answer

GCD(10,0) = 10 Where one is zero other is answer

Recursive Relation:

$$\text{GCD(m,n)} = \begin{cases} \text{undefined, if } m=0 \ \& \ n=0 \\ \text{m, if } n=0 \\ \text{n if } m=0 \\ \text{GCD}(n\%m, m) \text{ otherwise} \end{cases}$$

Recursive Program

```
GCD(m,n)
{
  if (m==0 && n==0) return (∞)
  if (n==0) return (m)
  if (m==0) return (n)
  else
  return ( GCD ( n%m , m))
}
```

Time Complexity = O ($\log_m n$)