**Recursion**

When a function calls itself directly or indirectly

**eg**- Tower of hinoi

**Three important rules:-**

1. A recursive algorithm must have a **base case**.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively.

**Base Condition** - to braek the recursion else it will give maximaum callstack error

**Stack overflow problem** - When the base condition is not reached or defined

**eg** - SUM OF N numbers

**Base condition** – Sum of n numbers

- f( n ) = 1 at n = 1 -> Base condition

- f( n )= n + f ( n – 1 ) at n > 1

**Type of recursive calls**

**Direct** – func f(){  
  
 f();  
}

**Indirect** – func f1(){ f2()} and func f2(){ f1()}

**Tail Recursion** – recursive call at the last statement of the function such that no further computation occurs after the recursive method returns.

**Fibonacci** – not tail recursive as fib(n-1) + fib(n-2) - addition occurs.

Factorial – no because n\*f(n-1) – multiplication occurs.

**Recursion disadvantages**-

Space equipped for recursive calls in stack.

Extra time taken in function calls and returns.

**When should we prefer recursion?**

People **use recursion** only when it is very complex to write iterative code. For example, tree traversal techniques like preorder, postorder can be made both iterative and **recursive**. But usually **we use recursive** because of its simplicity. Here's a simple example: how many elements in a set