August 17, 2025

# RECURSION FOUNDATIONS

## * Recursion

A function calling itself in its body is called Recursion.

- Example:

1.
```
fact(n) {
if (n==0) return 1;
return n * fact(n-1);
}
```

2.
```
fib(n){
if(n<=1) return n;
return fib(n-1)+fib(n-2);
}
```

→ Coding

→ Visualization : Harder as there is no real world examples

- Frameworks to visualize : Using Ex 2 (Fib)

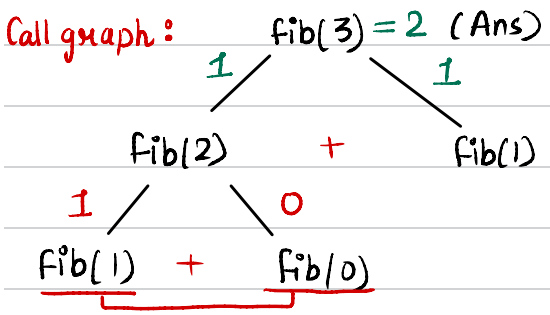
→ Code, visualize the code.

Easier to do if expression is given.

You only follow the other frameworks if you are not able to understand.

→ Recursion Tree, a diagram that shows different function calls and its return value.

Call graph:



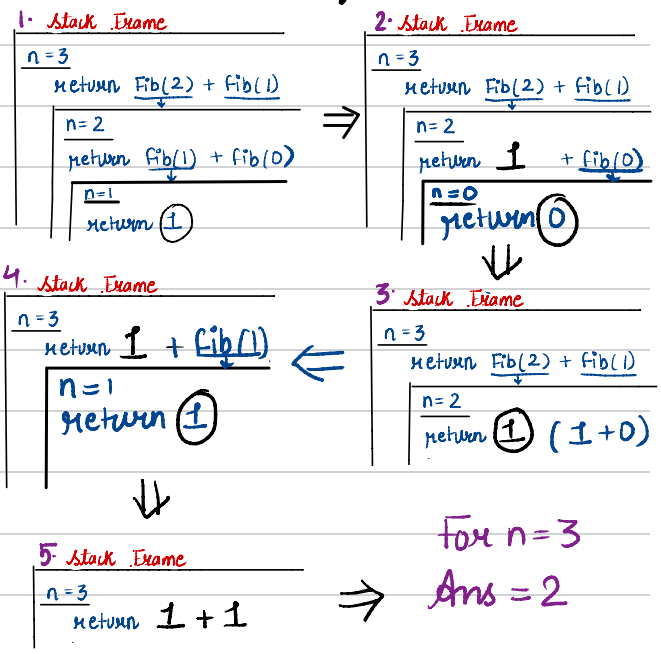

These two are base cases and therefore will return n (shown in Ex 2)

Visualization tool : https://visualgo.net/en/recursion

Even after Recursion Tree, its complex to visualize then go to next paradigm.

## → Recursion Stack

Recursion is implemented with the help of stack. There is no parallel execution, but sequential




Visualizer : https://pythontutor.com/visualize.html

## * Types

When we are dealing with Recursion, there are only certain types of code.

Four Forms of Recursion problems.

1. Encode a logic
2. Use Recursion to [Do, Recurse, Comeback]
3. K^th Finding
4. Fractals

The first two can be seen as forms of Backtracking

* Encode logic

Writing a function in a recursive manner like fibonacci.

Example:

1. Given a string s, check whether it is a palindrome or not using recursion.

S = 'madam'

The structure of our function:

1. Pruning

Check for if the parameters have valid values.

2. Base Case

Trivial, easy to determine

3. Calculate

implementation of the recursive mathematical fxn

```
is_palindrome ( first, last) {
        // Base Case          ↳ valid parameters hence no pruning
        if (first >= last) return 1;
        // Calculate
        if (s[first] != s[last]) return 0;
      return is_palindrome (first+1, last-1);
      }
```

2. Given value of n and r. Calculate $^nC_r$ recursively.

Mathematically,

$$^nC_r = \frac{n!}{r!\,(n-r)!}$$

Recursive Relation:

$$\left[ {}^nC_r = {}^{n-1}C_r + {}^{n-1}C_{r-1} \right]$$

For pruning, think when does r becomes invalid? $^nC_{(r)}$ → valid?

r becomes invalid when:

→ $r < 0$

→ $r > n$

For base case, we think of the simplest value that we can find aiming towards 0.

$^0C_0 = 1$ , $^1C_1 = 1$

For calculate, we can use our recursive relation

Pseudocode:

```
int ncr ( int n , int r) {
  // Pruning
if( r<0 || r>n) return 0;
  // Base Case
if (n==0) return 1;
  // Calculate
return ncr(n-1, r-1) + ncr(n-1, r);
}
```

## * Use Recursion to Do, Recurse, Comeback.

Ques 1. Write recursive code to print the following for a general n.

→ N, N-1, - - - - 2, 1, 2 - - - -, N-1, N

→ Ex: 4, 3, 2, 1, 2, 3, 4 (n=4)

```
printer(int N){
  if(n==1){                    → Base Case
    cout<<1<<endl;               (Base case and pruning still needs
    return;                       to be included)
  cout<<N<<endl;          // do
  Printer(N-1);           // Recurse
  cout<<N<<endl;          // Comeback, do more
}
```

Ques 2. Tower of Hanoi



Smaller disk can be placed on the largest disk only.

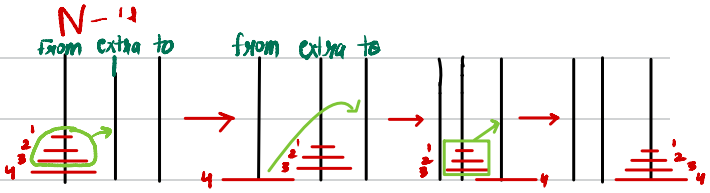
Find recursive delegation

move (N, from, to, extra)

move (N-1, from, extra, to)

N disk → from → to

move (N-1, extra, to, from)




Ex: N=3

move (3, 1, 3, 2) // Move 3 disk from 1 to 3 using 2.

move (2, 1, 2, 3)

1 → 3

move (2, 2, 3, 1)

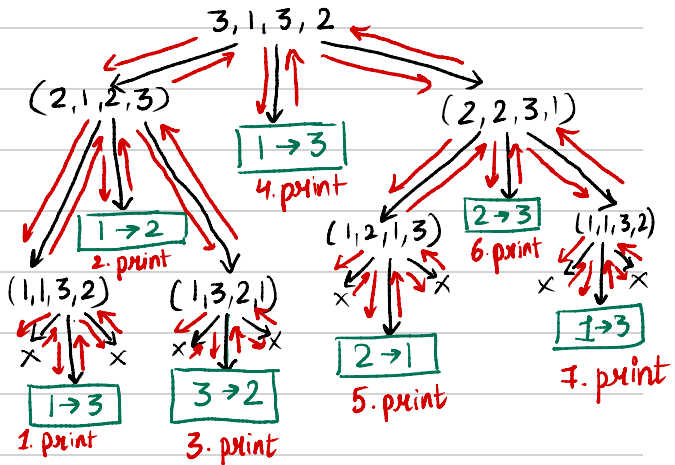
```
void movedisk(int disks,int from,int to, int extra)
{
    if(disks==0)return; // Base Case
    movedisk(disks-1,from,extra,to);
    cout<<"From "<<from<<" To "<<to<<endl;
    movedisk(disks-1,extra,to,from);
}

void solve(){
    movedisk(3,1,3,2);
}
```

Output

```
From 1 To 3
From 1 To 2
From 3 To 2
From 1 To 3
From 2 To 1
From 2 To 3
From 1 To 3
```

Recursion Tree:




1. 1→3
2. 1→2
3. 3→2
4. 1→3
5. 2→1
6. 2→3
7. 1→3

Do, Recurse, Comeback in no particular order (This example is a mix-match of it)