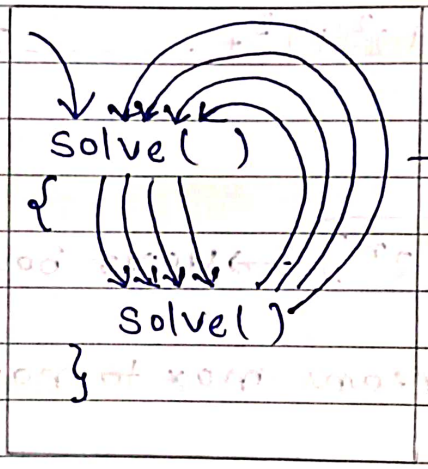
Recursion is the game of trust. So, trust the recursion.

09/10/2023
Monday

RECURSION

- **Bookish language →** when a function calls itself directly / indirectly.
- **In-Depth →** Solution of bigger problem **depends on →** Solution of chotti problem of same type.

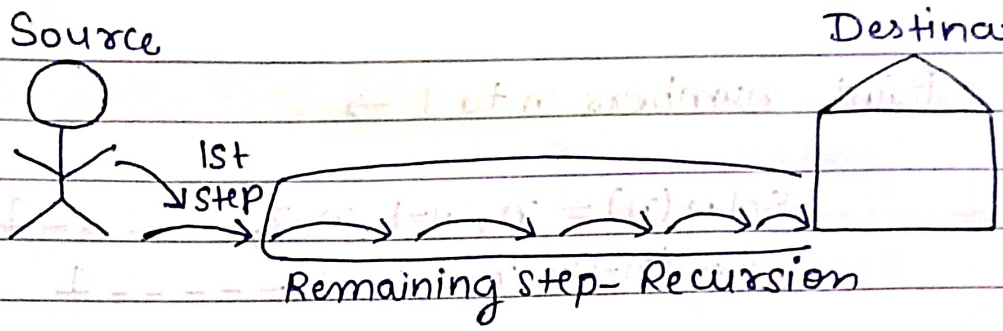


→ Solve( ) function is calling it again & again jab tak base case nahi mil jata.

- **Recursion be like,** 1 case tum solve kar do, baki mai sambhaal lunga.

How Recursion works?

* Assume, ek person ko place A se place B par jana hai, jiske liye 6 steps travel karna pdta hai.
So, 1 step person chalega and remaining 5 steps recursion chalega.



Step (6) = 1 + step (5)

- 1 → Single step travelled by person
- step (5) → Bigger problem
- chotti problem handled by recursion

So, here, OR kitna chalna hai → function calls
Kab rukna hai → Base case
left/right jana hai → Processings

Function recursive calls, base case and processings collectively make a recursive function.

* Calculate 2^n →

$Solve(n) = 2^n$

$Solve(n-1) = 2^{n-1}$

$2^n = 2 * 2^{n-1}$

$Solve(n) = 2 * solve(n-1)$

- $solve(n-1)$ → Bigger problem
- 2 → chotti problem

* **Print numbers n to 1 →**

Solve(n) = n, n-1, n-2, ---- 1
Solve(n-1) = n-1, n-2, ---- 1

Solve(n) = n, [n-1, n-2, ---- 1]
↓
Solve(n-1)

[Solve(n) = n, Solve(n-1)]
↓ Bigger problem
↓ Chotti problem

* **Find factorial →**

Solve(5) = 5 x 4 x 3 x 2 x 1
Solve(4) = 4 x 3 x 2 x 1

[Solve(5) = 5 x Solve(4)]
↓ Bigger problem
↓ Chotti Problem

- **Recursion code comprises →**

(1) Base Case (mandatory)

(2) Recursive call (mandatory)

(3) Processings (optional)

* Calculate factorial →

$fact(n) = n*(n-1)*(n-2)*\_\_\_\_3*2*1$

$fact(n-1) = (n-1)*(n-2)*\_\_\_\_3*2*1$

$fact(n) = n * \underbrace{(n-1)*(n-2)*\_\_\_\_3*2*1}_{fact(n-1)}$

$$fact(n) = n * fact(n-1)$$

Code →

```
#include <bits|stdc++.h>
using namespace std;
int fact(int n){
    // base case
    if(n==0 || n==1) return 1;
    // recursive call
    int recAns = fact(n-1);
    // processings
    int finalAns = n * recAns;
    return finalAns;
}

int main(){
    cout << fact(7) << endl;
}
```

O/P → 5040

Behind the scenes,

fact(4) → n=4

```
if (n==0 || n==1) → false
    return 1;
int recAns = fact(n-1);
int finalAns = n * recAns;
return finalAns;
```

finalAns=24

fact(3) → n=3

```
if (n==0 || n==1) → false
    return 1;
int recAns = fact(n-1);
int finalAns = n*fact(n-1);
return finalAns;
```

6

2

fact(2) → n=1

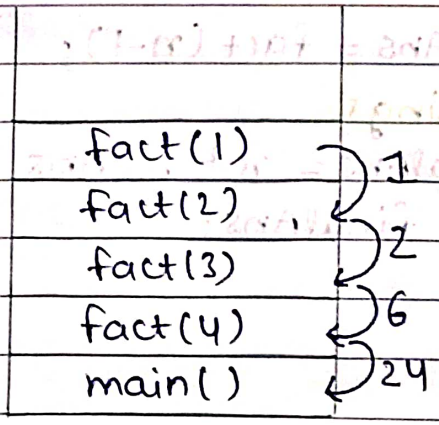
```
if (n==0 || n==1)
    return 1;   → True
int recAns = fact(n-1);
int finalAns = n * recAns;
    return finalAns;
```

1

fact(1) → n=1

```
if (n==0 || n==1) → false
    return 1;
int recAns = fact(n-1);
int finalAns = n * recAns;
    return finalAns;
```

Function call stack →



Entries get removed after returning some value and stack gets empty.

* Print counting n to 1 →

Print (n) = n, n-1, n-2, - - - , 2, 1
Print (n-1) = n-1, n-2, - - - , 2, 1

Print (n) = n, n-1, n-2, - - - , 2, 1
↓
Print (n-1)

Print (n) = n, print (n-1)

Code →
```
#include <bits/stdc++.h>
using namespace std;

void print (int n){
    // base case
    if (n == 0) return;
    // processings
    cout << n << " ";
    // recursive call
    print (n-1);
}

int main ( ){
    print (5);
}
```

o/p → 5 4 3 2 1

Note - To understand recursion, you need to first understand the recursion.

* **Tail and Head Recursion →**

If recursive relation comes after processing, then it is **tail recursion.**
If recursive relation comes before processing, then it is **head recursion.**

* **Calculate n^m →**

$power(n,m) = n^m$
$power(n,m-1) = n^{m-1}$

$power(n,m) = n * \boxed{n^{m-1}}$

$\boxed{n^{m-1}} \rightarrow power(n,m-1)$

$$\boxed{power(n,m) = n * power(n,m-1)}$$

Code →

```
#include<bits/stdc++.h>
using namespace std;

int power(int n, int m){
    if(m==0) return 1; //base case
    int pow = power(n,m-1); //recursive relation
    return (n*pow);
}
int main(){
    cout<<power(5,3)<<endl;
}
```

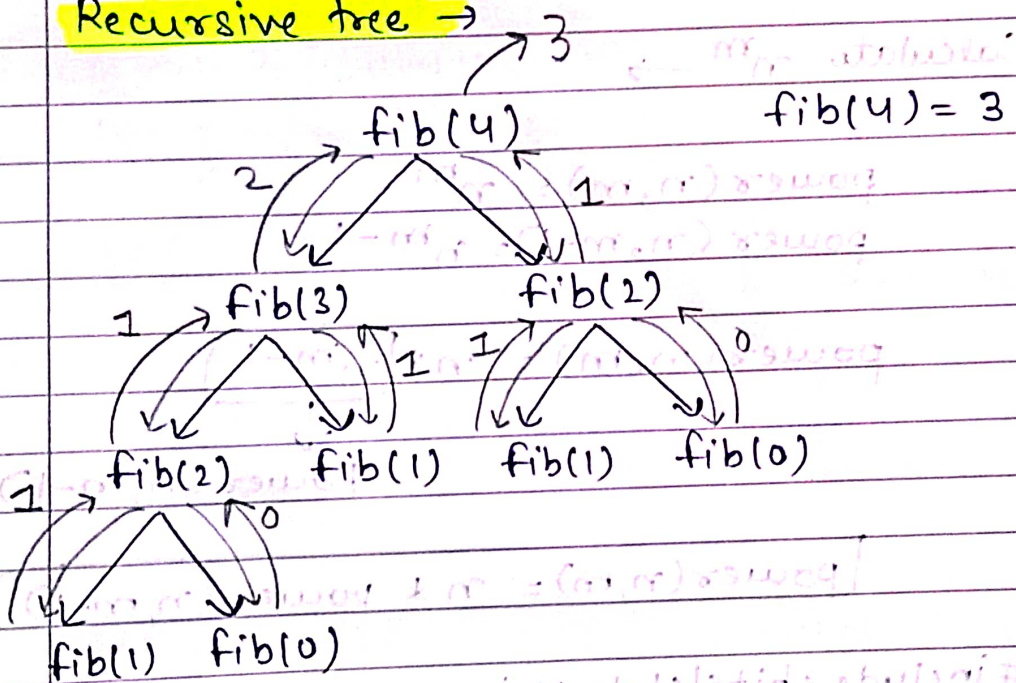
O/P → 125

* **Fibonacci Series →**

0 1 1 2 3 5 8 13 21 34 - - - -

$$fib(n) = fib(n-1) + fib(n-2)$$

Recursive tree →



Code →

```
#include<bits/stdc++.h>
using namespace std;

int fib(int n){
    // base case
    if(n==0 || n==1) return n;
    // recursive call
    int ans = fib(n-1) + fib(n-2);
    return ans;
}
int main(){
    cout << fib(8) << endl;
}
```

⟶ **O/P = 21**

* **Find Sum →**

$$sum(n) = n + (n-1) + (n-2) + \_\_\_\_ + 2 + 1$$
$$sum(n-1) = (n-1) + (n-2) + \_\_\_\_ + 2 + 1$$

$$\boxed{Sum(n) = n + sum(n-1)}$$

Code →

```
#include <bits/stdc++.h>
using namespace std;

int sum(int n){
    //base case
    if(n==0) return 0;
    //recursive calls
    int ans= n+sum(n-1);
    return ans;
}

int main(){
    cout<<sum(5)<<endl;
}
```

O/P → 15

Recursive tree →