# RECURSION - CLASS 1

Da / /.....

**Recursion :-**

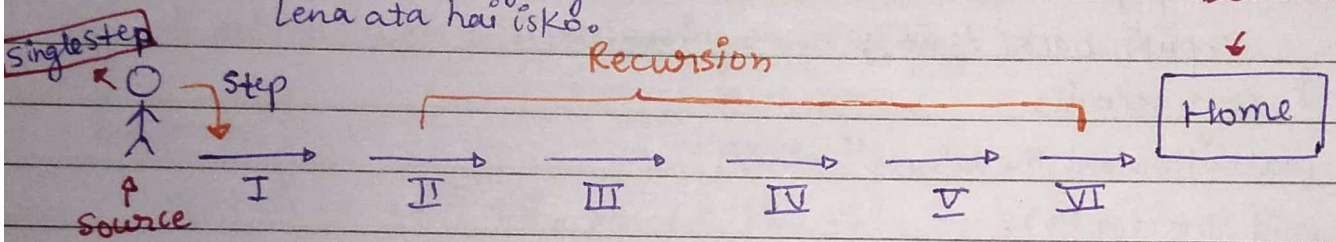
↳ Bookish Term → when a function calls itself directly/indirectly

↳ In-depth → Solution of Bigger Problem **depends** → Solution of chotti problem of same type.

∴ we can apply recursion in this case.

↳ Bhaiya ki Line → 1 case tum solve Karo, baki recursion sambhal lega

Problem statement - Isko ghr jaha hai but ek time pr single step Lena ata hai isko.




Bigger Problem → Step(6) = 1 + step(5)

∴ we can apply recursion in this case.

Single step — Recursion

Problem Statement - solve(n) **find** → $2^n$

Bigger Problem → $2^n$

$solve(n) = 2^n$

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

∵ $Solve(n-1) = 2^{n-1}$

$Solve(n) = 2 \times solve(n-1)$

Big Problem — Choti Problem

∴ same type of problems so we can apply recursion here.

Spiral

Date ..../..../.....

PS → solve (n) ⟶ (n ⟶ 1) counting Print

i.e. solve (n-1) ⟶ (n-1 ⟶ 1) counting Print krega

Solve (n) → n, n-1, n-2, - - - - , 1

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

Recursion

Big Problem ← Solve (n) = n, solve (n-1) → small problem of same type

∵ we can apply Recursion in this case

→ Factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

Solve (5) = 5!

solve(5) = 5 x 4 X 3 X 2 X 1

4!

solve(4) = 4! = 4X3X2X1

solve(5) = 5X4!

→ Ye Recursion Krdega

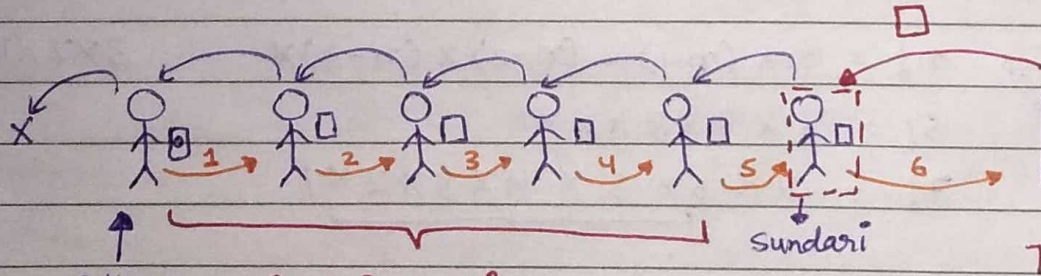
Solve (5) = 5 x solve(4)

Big Problem — Ye hum Krange — choti problem

Rule → if piche → Present → pass the paper

→ if → piche → absent → Rukjao

Ex:-



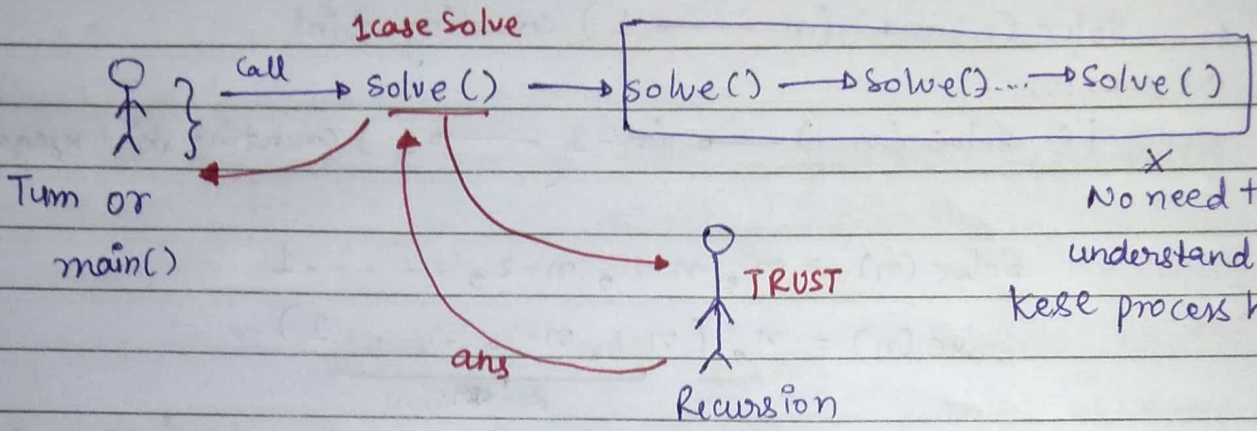

Rule → Aage wale Ko is blank paper pe +1 likh Kedo

Teacher ne sundari Ko paper diya tha to usse himapis chaiye !!

Piche K process se iskokoi Lena dena nhi !!

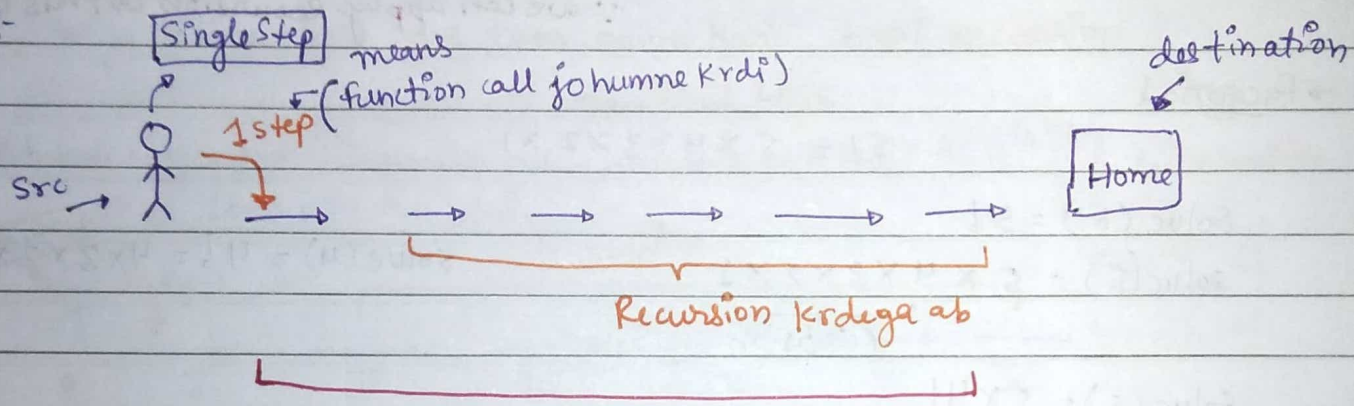
Spiral

Date ..../..../.....




∴ Recursion pe trust rakho ki ye ans laake dega bs.

Ex.-




| Overall Ans | = | 1 | + | Recursion Ans |
|---|---|---|---|---|
| Big | = | 1 | + | choti |

Q factorial → $n! = n \times (n-1) \times (n-2) \times (n-3) \times \cdots 3 \times 2 \times 1$

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$

$fact(7) = 7 \times 6!$

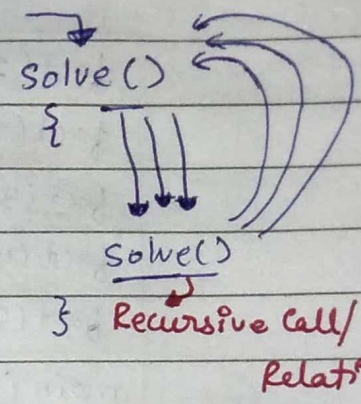
$fact(7) = 7 \times fact(6)$

$fact(n) = n \times fact(n-1)$

Date ..../..../.....

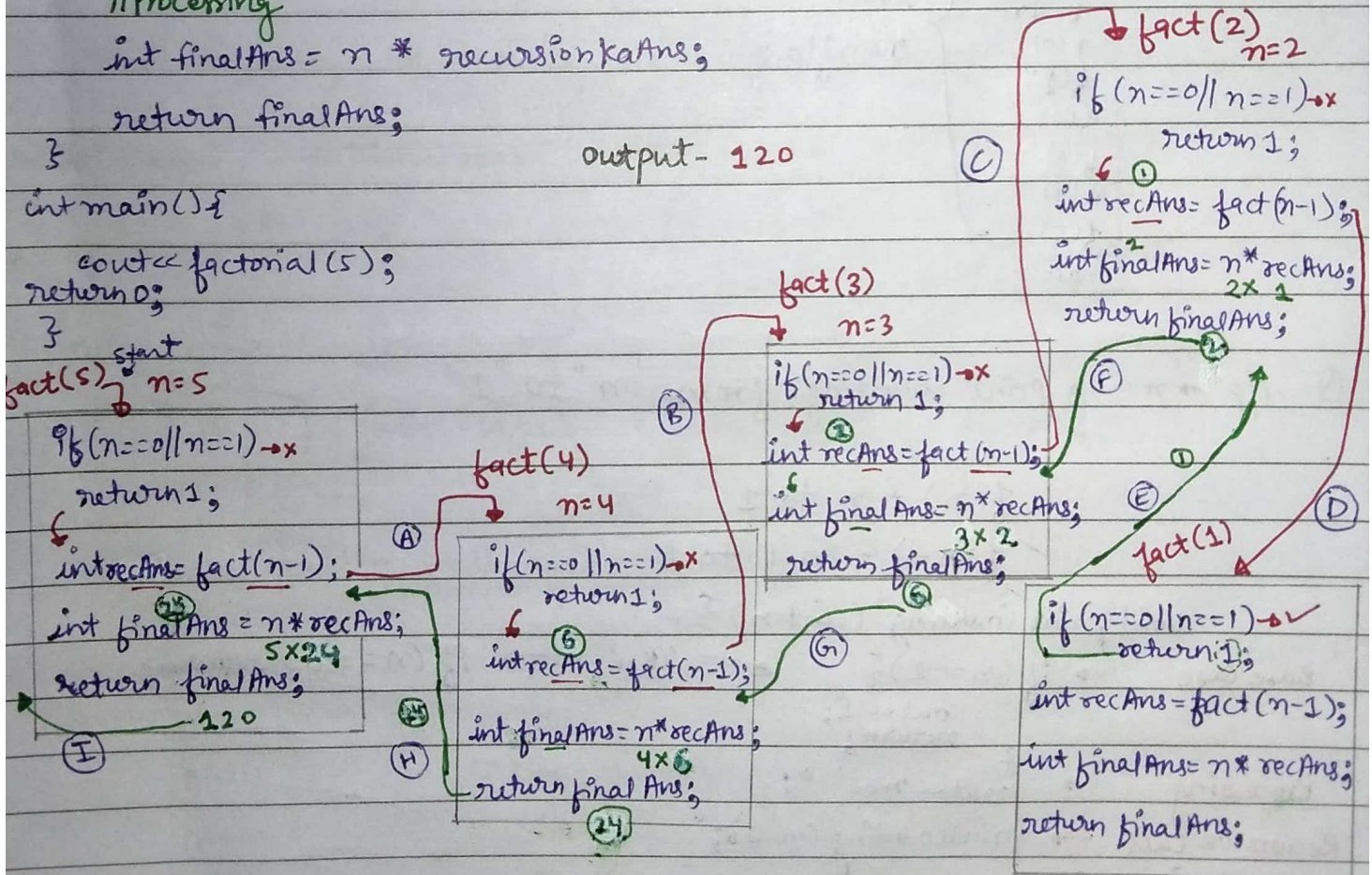
# Parts of Recursive code :-

- Base Case → mandatory
  - ↳ To stop the recursion
- Recursive Call → mandatory
  - or
  - Recursive Relation
- Processing → optional




```
ode⇒ int factorial(int n){
    //baseCase
    if (n==1) return 1;
    if(n==0) return 1;
    //Processing
    //Recursive Relation
    int recursionKaAns = factorial(n-1);
    //Processing
    int finalAns = n * recursionKaAns;
    return finalAns;
}
int main(){
    cout<< factorial(5);
return 0;
}
```

output- 120




Spiral

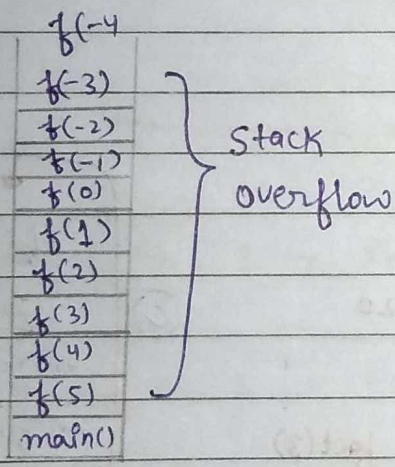
Date ..../..../.....

| |
|---|
| fact(1) |
| fact(2) |
| fact(3) |
| fact(4) |
| fact(5) |
| main() |

| | |
|---|---|
| ~~fact(1)~~ | Popped from stack and returned 1 |
| ~~fact(2)~~ | Popped & returned 2 |
| ~~fact(3)~~ | Popped & returned 6 |
| ~~fact(4)~~ | Popped & returned 24 |
| ~~fact(5)~~ | Popped & returned 120 |
| main() | |

∴ function call stack

→ If we don't apply Base case, then stack will filled with calls and it arise stack overflow condition and memory gets occupied by the program.



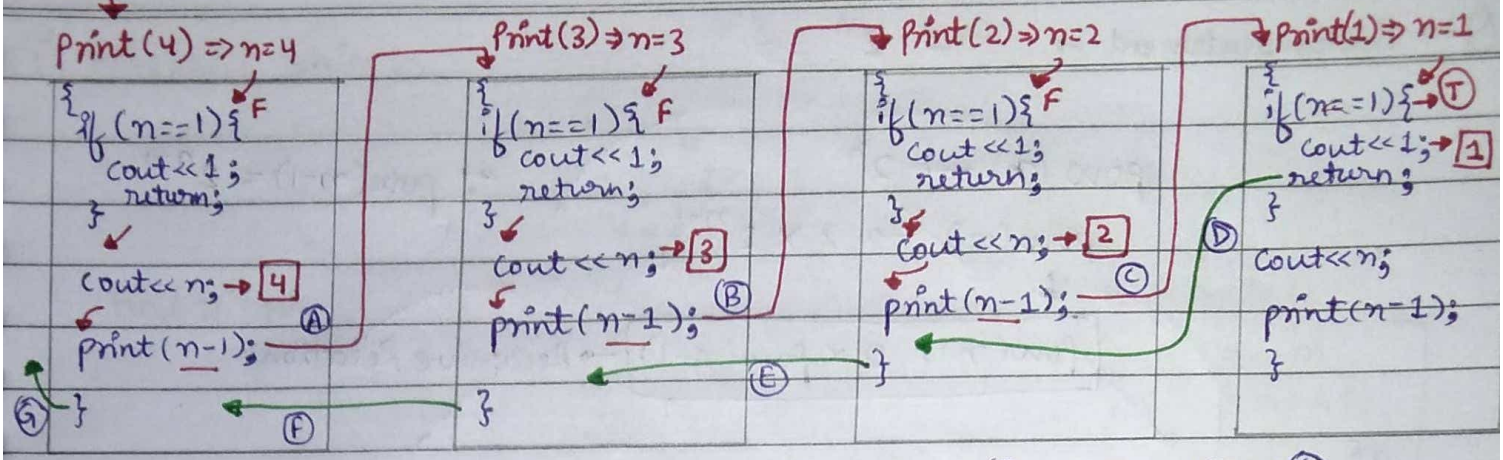

Q i/p → n → print counting from "n" to "1"

f(n) = n to 1

f(n-1) = n-1 to 1

```
Void printCounting (int n) {
Base Case  → if (n==1){        ← //B.C →  if (n==0) return;
               cout<< 1;
               return;
             }
Processing → cout<< n<<" ";
Recursive call → printCounting(n-1);
             }
```

Date ..../..../.....



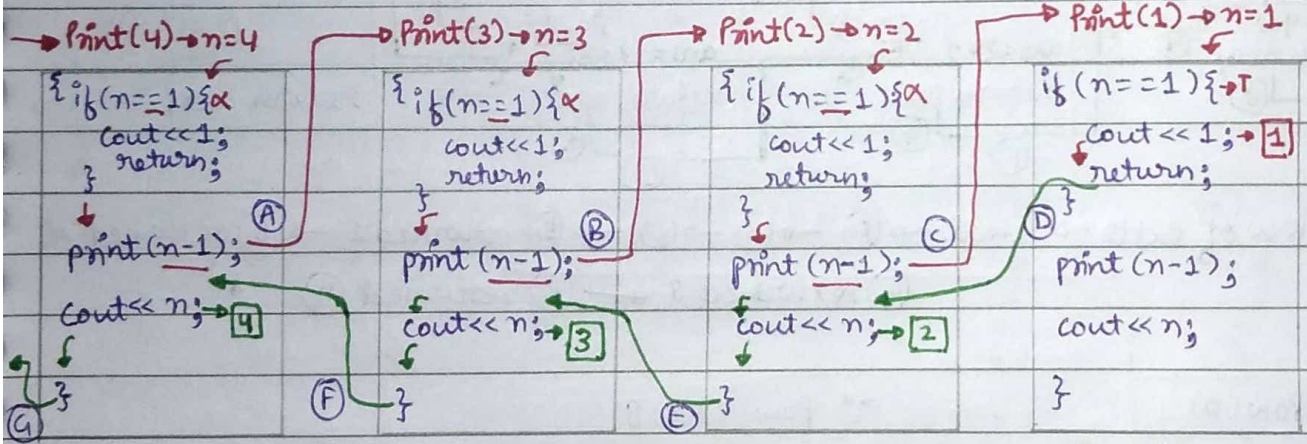

Flow of calls → (A) → (B) → (C) → (D) → (E) → (F) → (G)
↳ Processing BEFORE calling.

| | |
|---|---|
| function call stack | Print(1) |
| | Print(2) |
| | Print(3) |
| | Print(4) |
| | main() |

∴ It will print → 4 3 2 1 and popped out accordingly.

∵ Here, the function is ending with Recursive Relation. So, it's called Tail Recursion.

→ when we change the order of Processing and Recursive Relation.




Flow of calls → (A) → (B) → (C) → (D) → (E) → (F) → (G)
↳ Processing AFTER calling.

| | |
|---|---|
| function call stacks | Print(1) |
| | Print(2) |
| | Print(3) |
| | Print(4) |
| | main() |

∴ It will print → 1 2 3 4 & popped out

∵ Here, the Recursive Relation is before processing it's called Head Recursion.

Spiral

Date ..../..../.....

Q → Problem Statement → find $2^n$

pow(n) → $2^n$ ∵ pow(n-1) = $2^{n-1}$

pow(n) → $2 \times 2^{n-1}$

↓

**pow(n) = 2 × pow(n-1)** → Recursive Relation

$2^5$

↘ $2 \times 2^4$

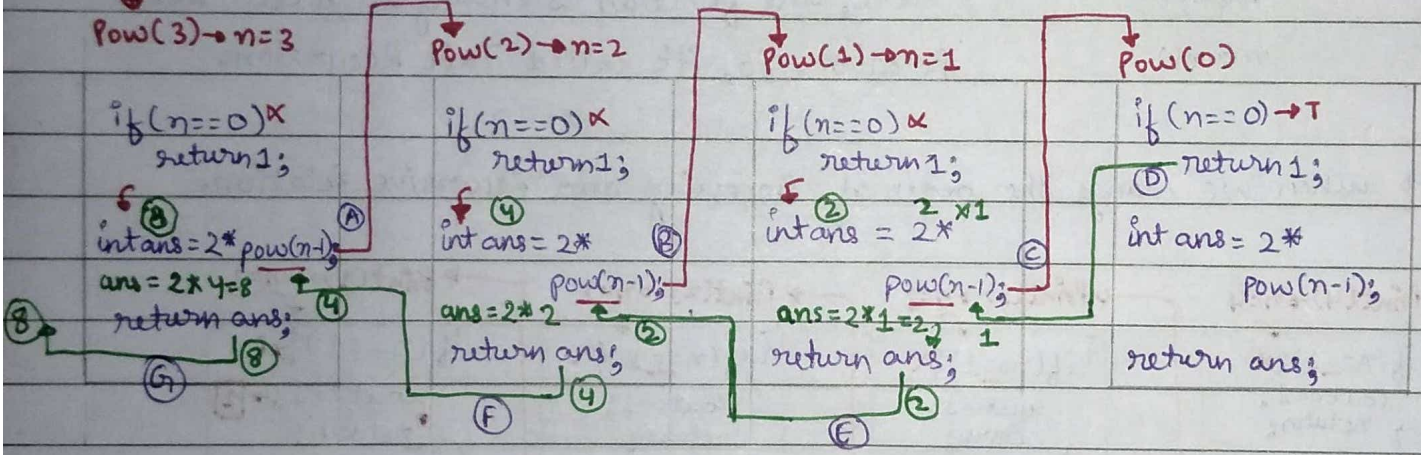
↘ $2 \times 2^3$

↘ $2 \times 2^2$

↘ $2 \times 2^1$

↘ $2 \times 2^0$ = 1 → Basecase → if(n==0) return 1;

Start




Flow of calls → start → (A) → (B) → (C) → (D) returned 1 → (E) returned 2 → (F) returned 4 → (G) returned 8

| |
|---|
| Pow(0) |
| Pow(1) |
| Pow(2) |
| Pow(3) |
| main() |

∵ Pow(3) = 8

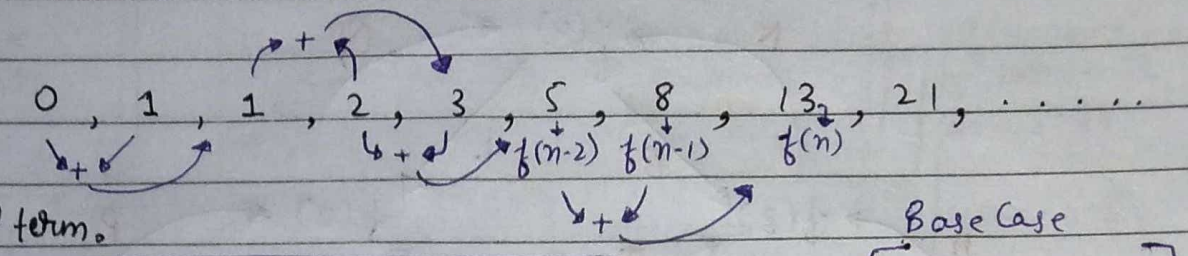
↳ Each call will pop out after returning particular ans.

Function call stack

Spiral

Date ..../..../.....

## Q Fibonacci Series :-




find $n^{th}$ term.

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

↳ Recursive Relation

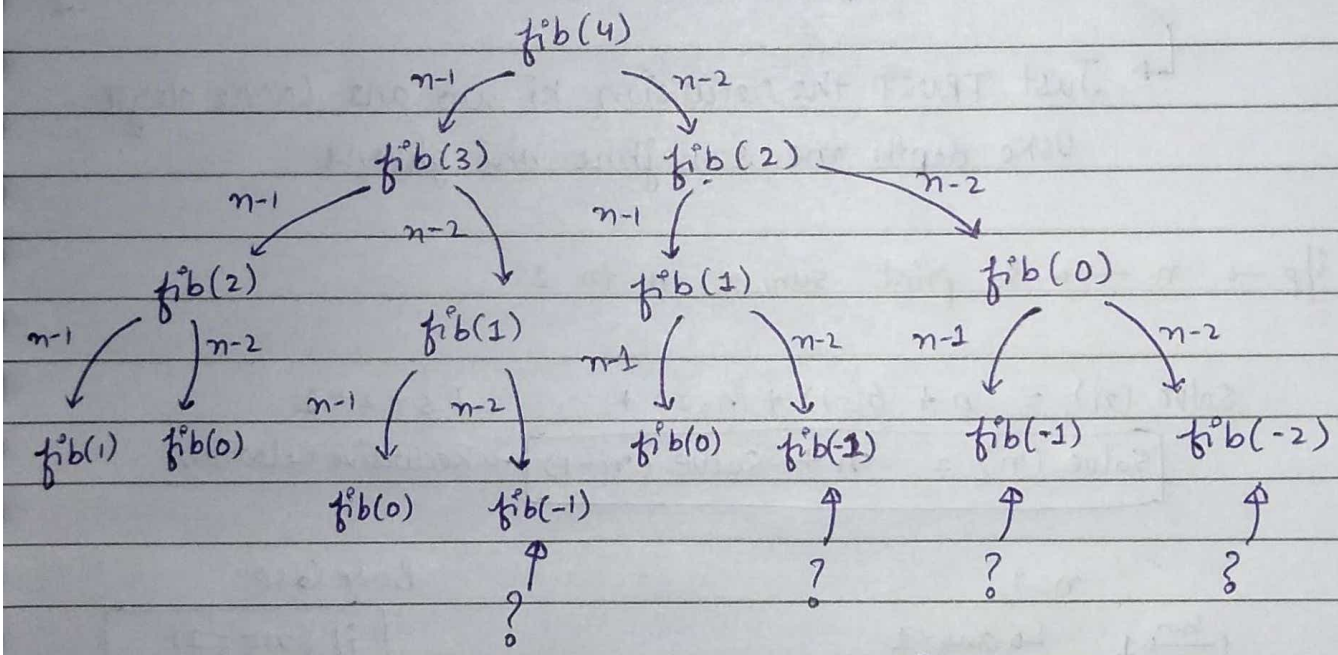
Base Case

```
if (n == 0)
    return 0;
if (n == 1)
    return 1;
```

Code :-

```
void fib (int n) {
//Base Case
if(n==0) return 0;
if(n==1) return 1;
//R.R
int ans = fib(n-1) + fib(n-2);
return ans;
```

Recursion Tree




fib(0) Ke niche ans kese nikalu nhi pta

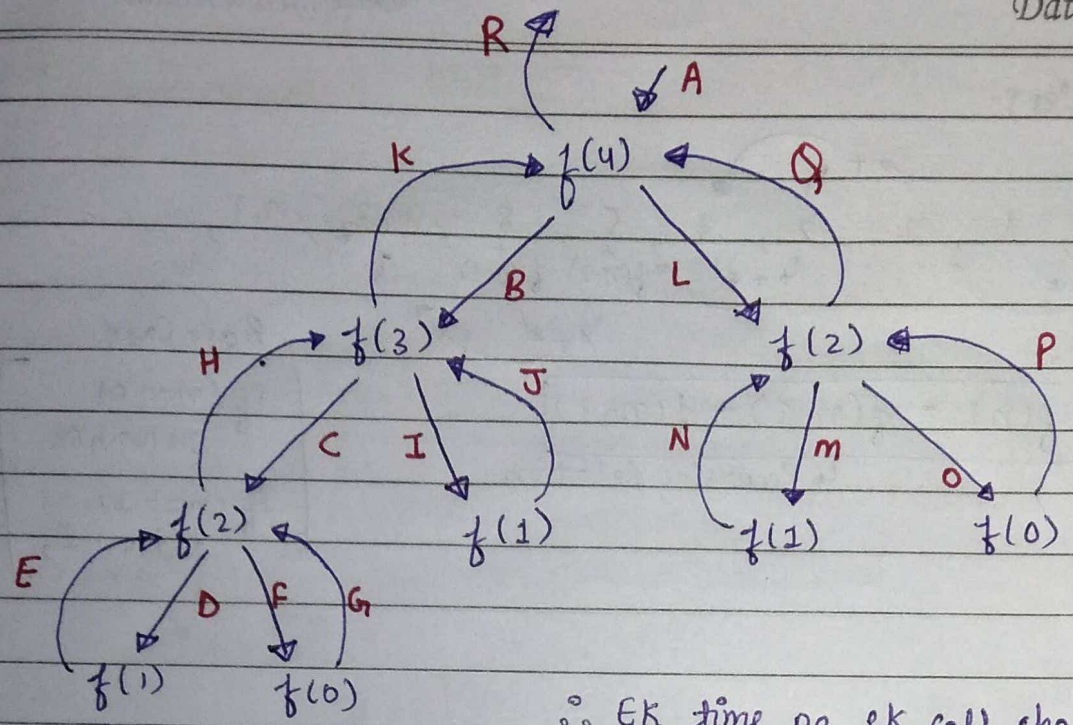
or

fib(1) Ke niche kese nikalu nhi pta so yahi hai Base Case, inki values pta honi chaiye.

```
if(n==0)
    return 0;
if(n==1)
    return 1;
```

Base Case

Date ..../..../.....




∴ EK time pe ek call chalegi

→ [To understand Recursion, you need to first understand Recursion]

↳ Just TRUST the recursion ki wo ans Laake dega. Usko depth me samajhne nhi ghusna

Q i/p → n → so print sum of n to 1

Solve (n) = n + (n-1) + (n-2) + ---- + 3 + 2 + 1

Solve (n) = n + solve (n-1) → Recursive Relation

n = 1

1 —Sum→ 1 Lo ans = 1

Base Case

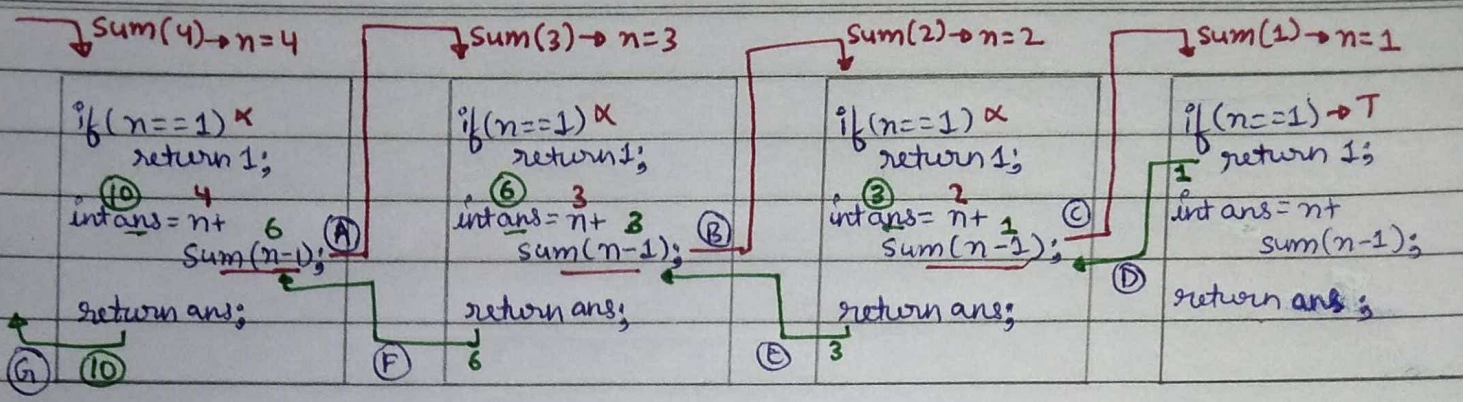
```
if (n==1)
    return 1;
```

Code :-

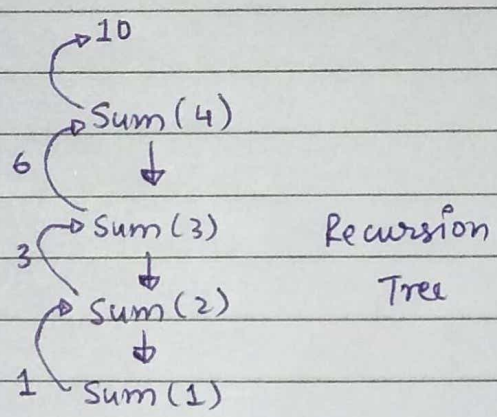
```
int sum (int n){
  //Base Case
  if(n==1)
    return 1;

  int ans= n+ sum(n-1);
  return ans;
}
```

Date ..../..../.....




| | | |
|---|---|---|
| | | Sum(4) = 10 (1+2+3+4) |
| | Sum(1) | • All entries will be pop out after returning particular value. |
| function | Sum(2) | |
| call | Sum(3) | |
| stack | Sum(4) | |
| | main() | |




Recursion Tree

Spiral