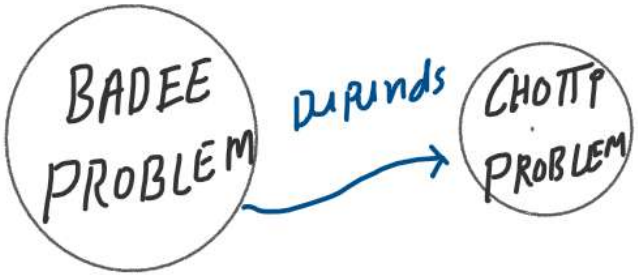
9/10/2023

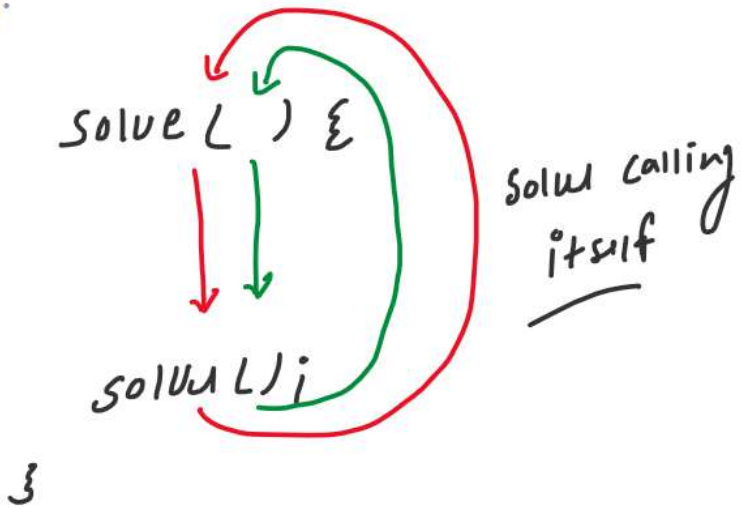
Recursion Class 1

✅ 1. Bookish definition of recursion

↳ Recursion refers to a function that calls itself either directly or indirectly.

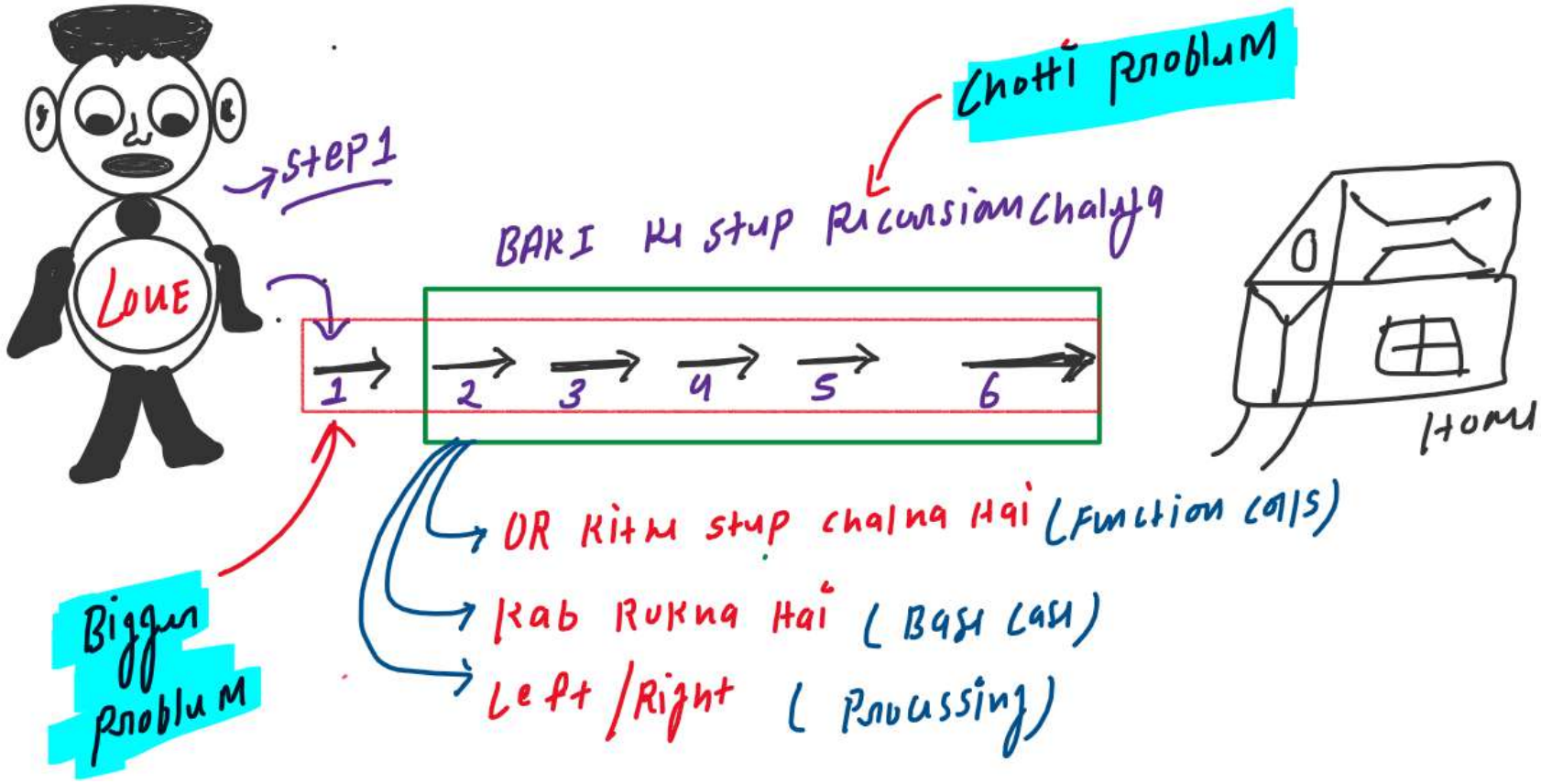
↳ Solution of bigger problem depends on solution of small problem.





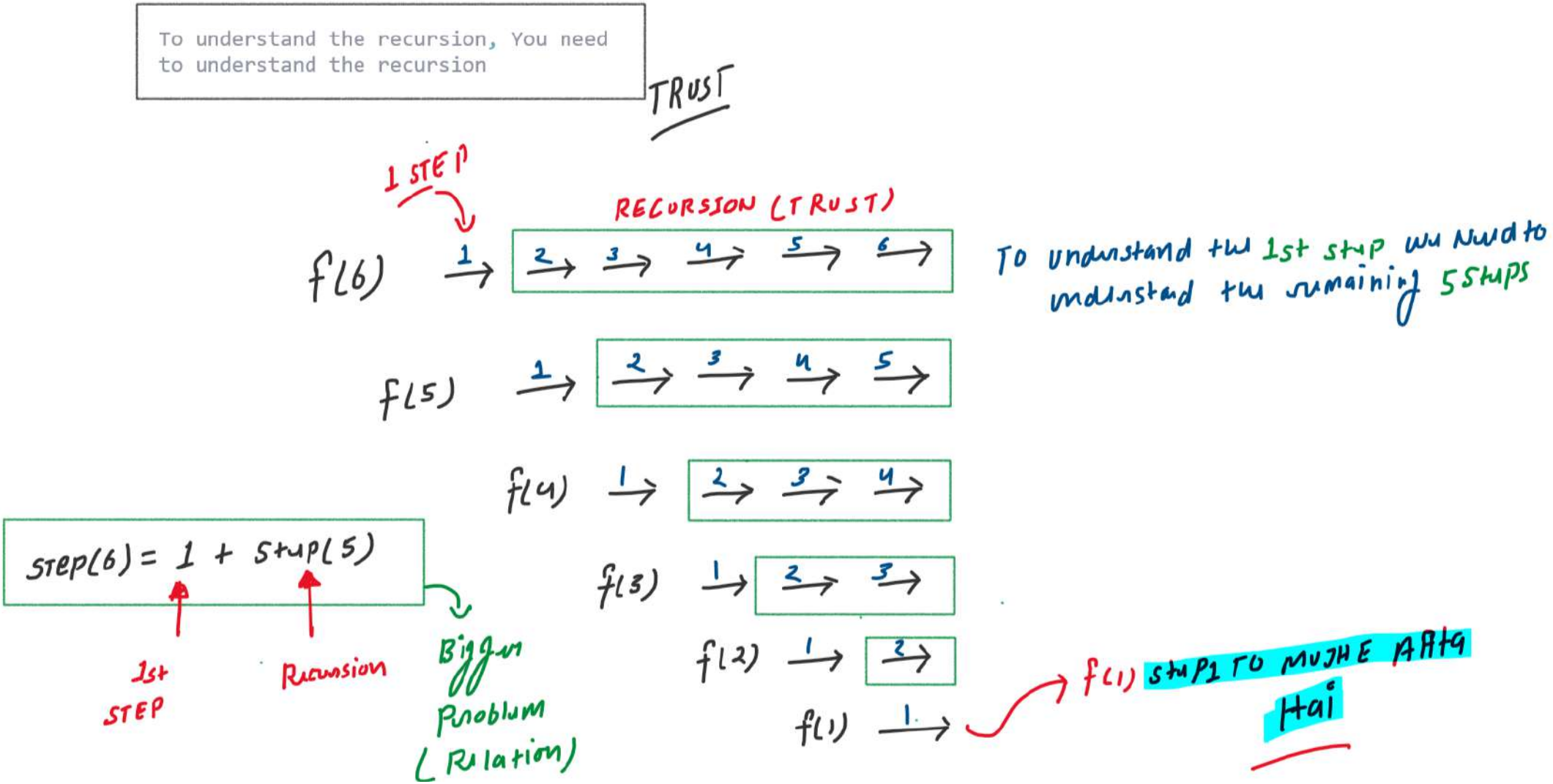
✅ 2. Love Bhaiya's definition of recursion

Ek *case* solve tum karlo baaki ka recursion khud solve kar lega.



To understand the recursion, You need
to understand the recursion

TRUST



Example

$Solve(n) \longrightarrow 2^n$

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

Bigger problem $= 2^n$

Chotti problem $= 2^{n-1}$

1

$Solve(n) = 2^n$

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

Recursion Relation → $Solve(n) = 2 * Solve(n-1)$

Bigger — Chotti

Example

Counting Print from N to 1

Solve(N) = (N — 1) Counting Print

Solve(N-1) = (N-1 — 1) Counting

Solve(N) = N, N-1, N-2, N-3, ..., 1

Recursion Relation

Solve(N) = N, [N-1, N-2, N-3, ---, 1]

Bigger

Chotti

Example

Factorial

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

solve(5) = 5!

solve(5) = 5 × 4 × 3 × 2 × 1

solve(5) = 5 × 4!

Recursion Relation:

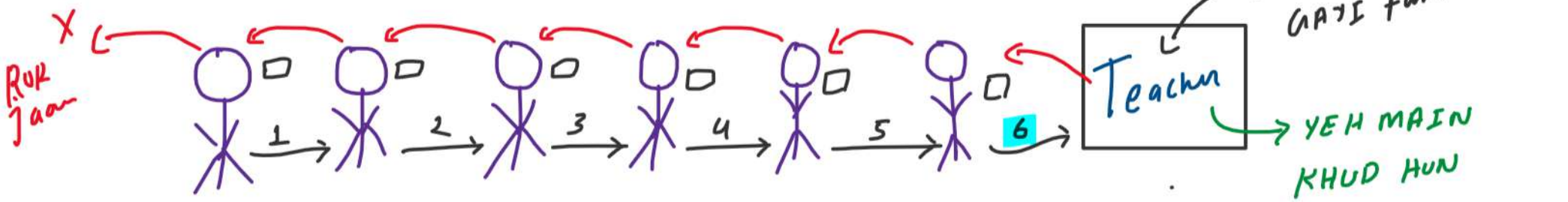
solve(5) = 5 × solve(4)

Biggar Pro.

1 step

Chotti Pro.

Reallife Example



Rule

↳ AAGE wale ko apna (+1) karna hai

Rule

if → picha → present hai ✓ pass the paper

if → picha → absent hai ✗ RUK jaao ...

✅ 3. Recursion mandatory terms

1. Base Case (mandatory)
2. Recursive calls (Relation) (mandatory)
3. Processing (optional)

✅4. Factorial of n number

PROGRAM:1

$$N! = N \times (N-1) \times (N-2) \times (N-3) \times \cdots \times 3 \times 2 \times 1$$

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

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

$7! = 7 \times 6!$

Recursion call | Relation

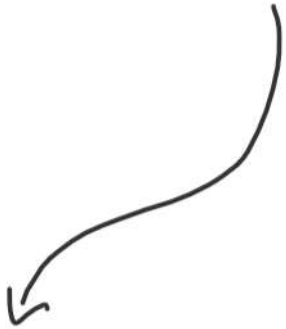
$fact(N) = N \times fact(N-1)$

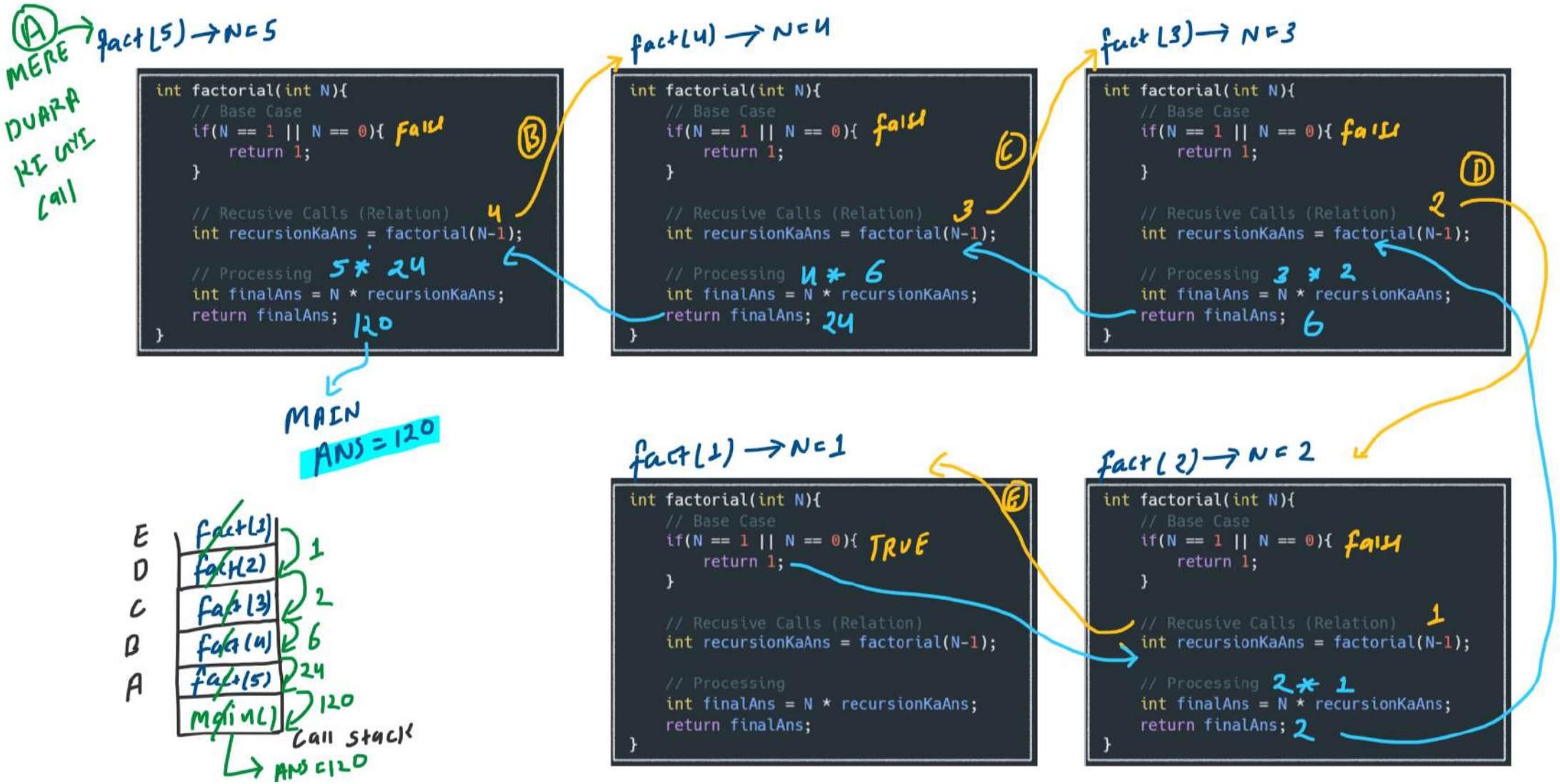
Processing

Base case ⇒ if(fact(1))
↳ Return 1
if(fact(0))
↳ Ret 1

```cpp
// ✅PROGRAM 01: Factorial of n number
#include<iostream>
using namespace std;

int factorial(int N){
    // Base Case
    if(N == 1 || N == 0){
        return 1;
    }

    // Recusive Calls (Relation)
    int recursionKaAns = factorial(N-1);

    // Processing
    int finalAns = N * recursionKaAns;
    return finalAns;
}

int main(){
    int N = 5;
    int factorialAns = factorial(N);
    cout<<"Factorial is "<<factorialAns<<endl;

    return 0;
}
```

✅5. How recursion work and function call stack




A
MERE DUARA KI GAYI CALL
fact(5) → N=5
fact(4) → N=4
fact(3) → N=3
fact(2) → N=2
fact(1) → N=1
int factorial(int N){
// Base Case
if(N == 1 || N == 0){ FALSE
return 1;
}
// Recusive Calls (Relation)
int recursionKaAns = factorial(N-1);
// Processing
int finalAns = N * recursionKaAns;
return finalAns;
}
B
C
D
E
F
4
3
2
1
5* 24
4 * 6
3 * 2
2 * 1
120
24
6
2
TRUE
MAIN
ANS = 120
E
D
C
B
A
fact(1)
fact(2)
fact(3)
fact(4)
fact(5)
main()
1
2
6
24
120
Call stack
ANS = 120

✅6. Why base case important (Due to Stack Overflow)

Don't know Base case

Stack Overflow

Call stack

Have Base case

No stack overflow

✅7. Reverse counting from n to 1

Program: 02

Example

N = 5 output = 5 4 3 2 1

N = 1 output = 1

Base case

if(N==1)

↳ Print N=1

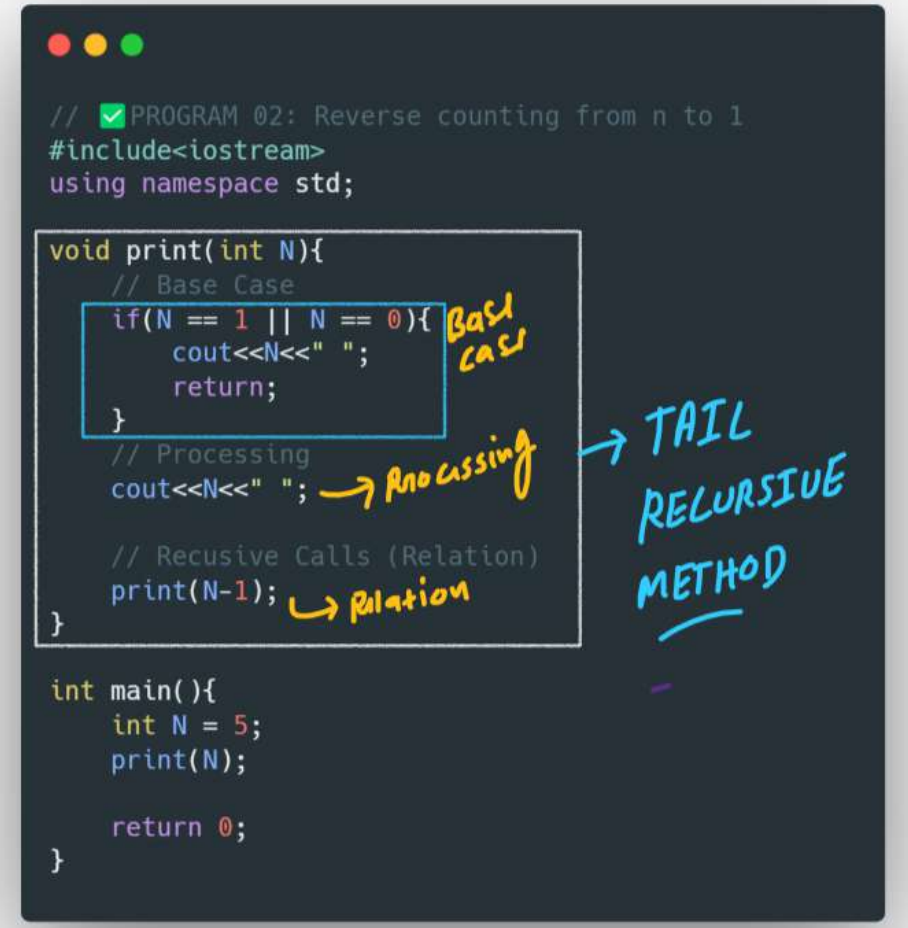
↳ return

When N=5

1st STEP ↓ RECURSION ↓ solve karlega

Print(N) = 5, 4, 3, 2, 1

↳ Print(N-1)



Ex when N=0, output:0

Basi cas

```
if(N==0){
cout<<N<<" ";
return;
}
```

print(5) → N = 5

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){ false
        cout<<N<<" ";
        return;
    }
    // Processing
    cout<<N<<" "; = 5

    // Recusive Calls (Relation)
    print(N-1);
}        4
```

print(4) → N = 4

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){ false
        cout<<N<<" ";
        return;
    }
    // Processing
    cout<<N<<" "; = 4

    // Recusive Calls (Relation)
    print(N-1);
}        3
```

print(3) → N = 3

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){ false
        cout<<N<<" ";
        return;
    }
    // Processing
    cout<<N<<" "; = 3

    // Recusive Calls (Relation)
    print(N-1);
}        2
```

MAIN

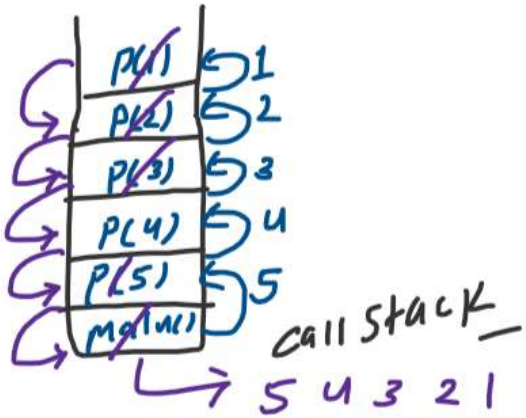
5 4 3 2 1

print(1)

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){ True
        cout<<N<<" ";
        return;
    }
    // Processing
    cout<<N<<" ";

    // Recusive Calls (Relation)
    print(N-1);
}
```

print(2) → N = 2

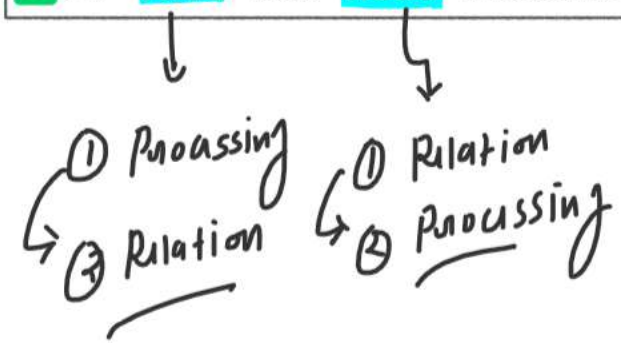
```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){ false
        cout<<N<<" ";
        return;
    }
    // Processing
    cout<<N<<" "; = 2

    // Recusive Calls (Relation)
    print(N-1);
}        1
```



call stack

5 4 3 2 1

✅ 8. Tail and head recursion



Ex N=5 output: 1 2 3 4 5

```cpp
// ✅PROGRAM 02: Reverse counting from n to 1
#include<iostream>
using namespace std;

void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);

    // Processing
    cout<<N<<" ";
}

int main(){
    int N = 5;
    print(N);

    return 0;
}
```

HEAD RECURSION METHOD

print(5) → N = 5

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);
    // Processing
    cout<<N<<" ";
}
```

fall

4

5

print(4) → N = 4

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);
    // Processing
    cout<<N<<" ";
}
```

fall

3

4

print(3) → N = 3

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);
    // Processing
    cout<<N<<" ";
}
```

fall

2

3

print(2) → N = 2

```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);
    // Processing
    cout<<N<<" ";
}
```

fall

1

↳ 2

print(1) → N = 1

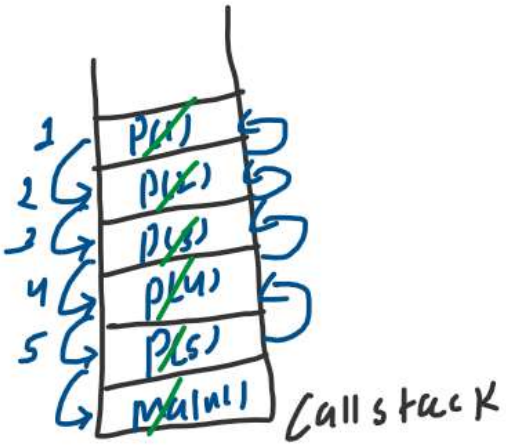
```
void print(int N){
    // Base Case
    if(N == 1 || N == 0){
        cout<<N<<" ";
        return;
    }
    // Recusive Calls (Relation)
    print(N-1);

    // Processing
    cout<<N<<" ";
}
```

TRUE

↳ 1

{1 2 3 4 5}

Output



Call stack

✅ 9. Pow(2,N) Program 3

EX

Problem statement $2^N = ?$

Input	$N = 5$	$N = 1$	$N = 0$
Output	$2^N = 32$	$2^N = 2$	$2^N = 1$

$2^5 \rightarrow 2 \times 2^4 \rightarrow 2 \times 2^3 \rightarrow 2 \times 2^2 \rightarrow 2 \times 2^1 \rightarrow 2 \times 2^0 \rightarrow 1$

if(pow(0)) return 1 } Base case

$\Rightarrow$ $Pow(N) = 2^N$

$\Rightarrow$ $Pow(N) = 2 \times 2^{N-1}$

$\Rightarrow$ $pow(N) = 2 * P(N-1)$ Recursion Relation

N = 5

$2^5 = 2 \times 2^4$

$= 2 \times 2 \times 2^3$

$= 2 \times 2 \times 2 \times 2^2$

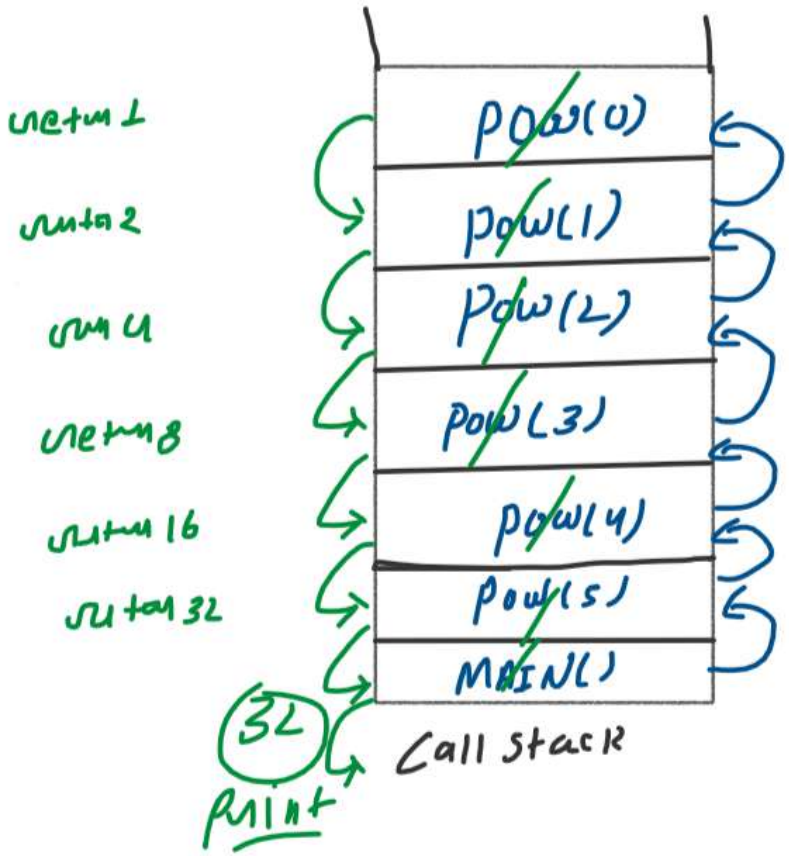
$= 2 \times 2 \times 2 \times 2 \times 2^1$

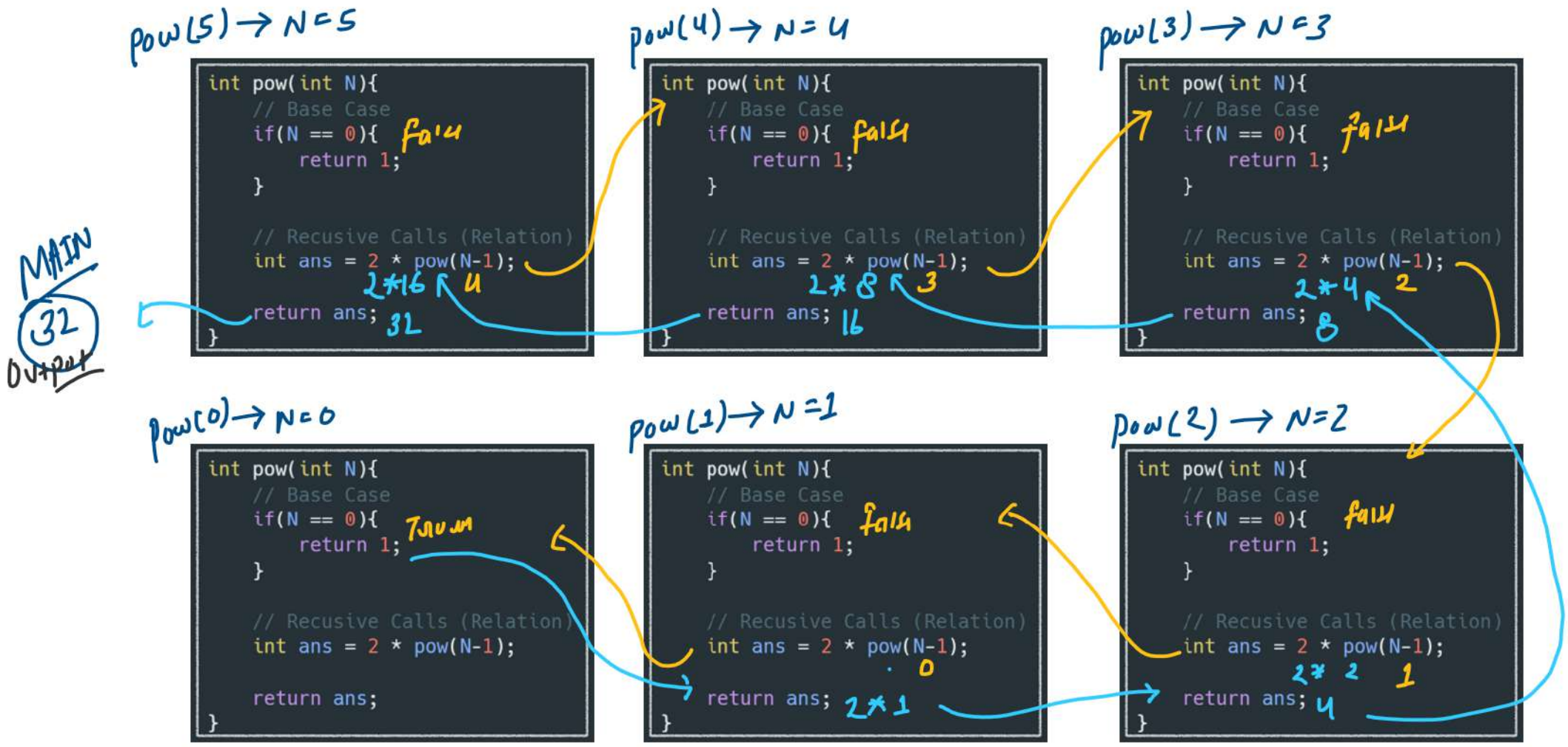
$= 2 \times 2 \times 2 \times 2 \times 2 \times 2^0$

$= 2 \times 2 \times 2 \times 2 \times 2 \times 1$

if(P(0)) → return 1

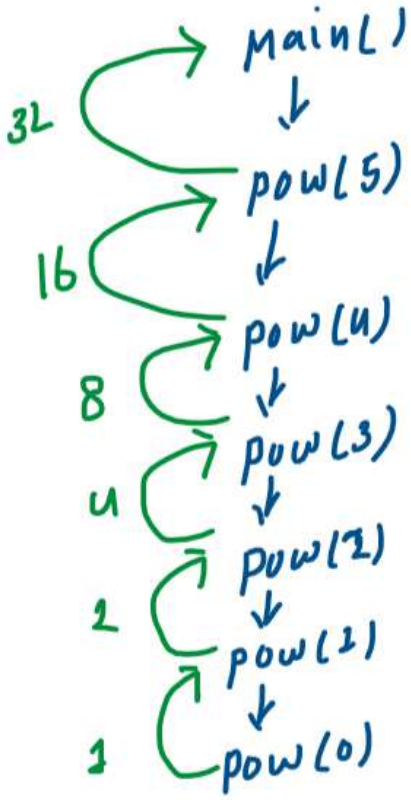
```cpp
// ✅PROGRAM 03: Pow(2,N)
#include<iostream>
using namespace std;

int pow(int N){
    // Base Case
    if(N == 0){
        return 1;
    }

    // Recusive Calls (Relation)
    int ans = 2 * pow(N-1);

    return ans;
}

int main(){
    int N = 5;
    cout<<pow(N);
    return 0;
}
```




pow(5) → N=5
pow(4) → N=4
pow(3) → N=3
pow(0) → N=0
pow(1) → N=1
pow(2) → N=2
int pow(int N){
// Base Case
if(N == 0){
return 1;
}
// Recusive Calls (Relation)
int ans = 2 * pow(N-1);
return ans;
}
fall
2*16
4
31
2*8
3
16
2*4
2
8
Return 1
2*1
0
1
MAIN
32
Output

✅ 10. Recursive tree

① P.S. $= 2^n$ when $n = 5$



✅11. Fibonacci series

Chotti problem

0th	1th	2nd	3rd	4th	5th	6th	7th	8th	9th	---nth---
0	1	1	2	3 + 5 = 8			13	21	34	.....

A → fib(N-2)

B → fib(N-1)

Next = A+B → fib(N)

Ex

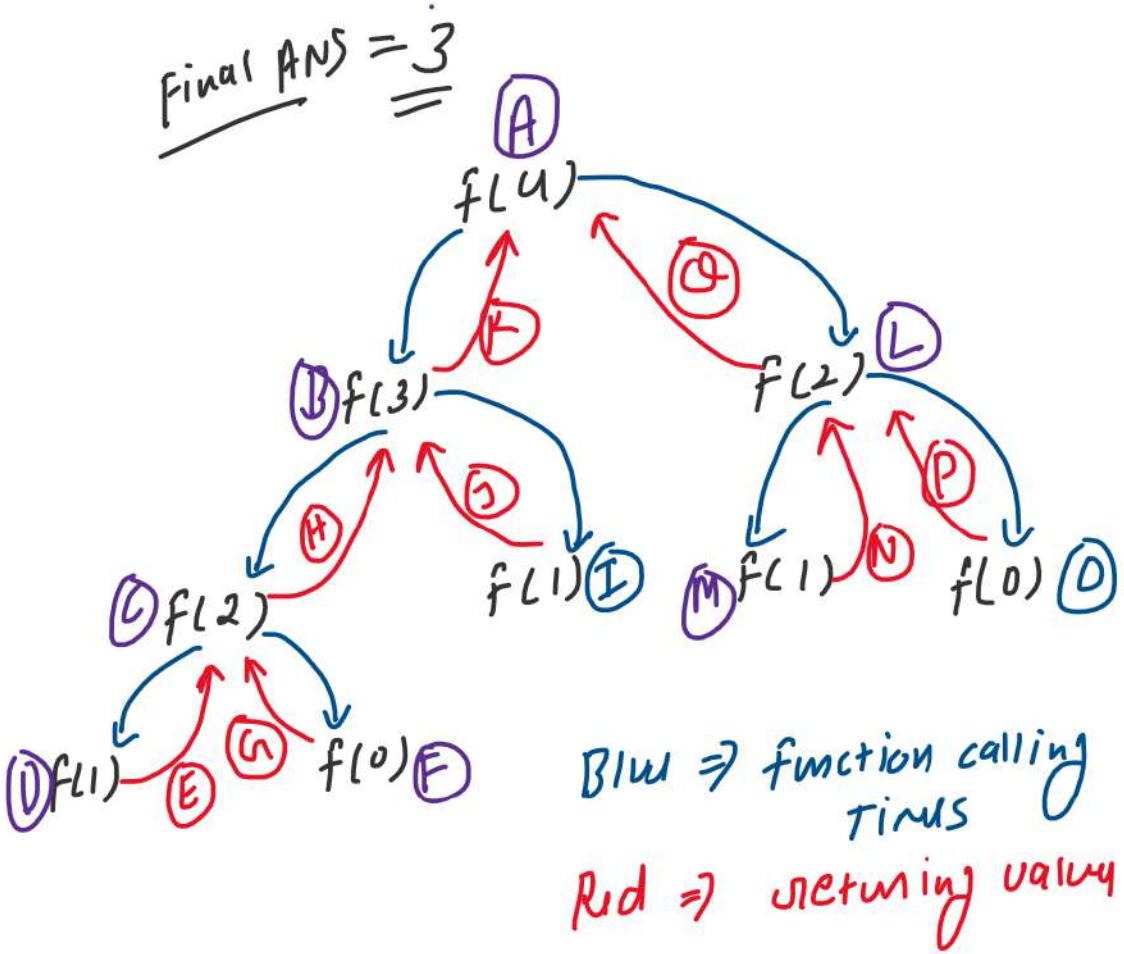
BASE CASE	Output
N = 0	0
N = 1	1
N = 2	1
N = 5	5
N = 6	8

Relation

fib(2) = f(2-1) + f(2-2) = 1 + 0 = 1

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

```cpp
// ✅PROGRAM 04: Fibonacci series
#include<iostream>
using namespace std;

int fib(int N){
    // Base Case
    if(N == 0){
        return 0;
    }
    if(N == 1){
        return 1;
    }

    // Recusive Calls (Relation)
    int ans = fib(N-1) + fib(N-2);
    return ans;
}

int main(){
    int N = 5;
    cout<<"Nth term is "<<fib(N);

    return 0;
}
```



✅12. Return sum from n to 1

Ex

N = 1 output = 1

N = 2 output = 3 → Chotti Problem

N = 5 output = 15

N = 2 ⇒ 2 + 1 = 3

SUM(N) = N + (N-1) + (N-2) + --- + 1 Base case

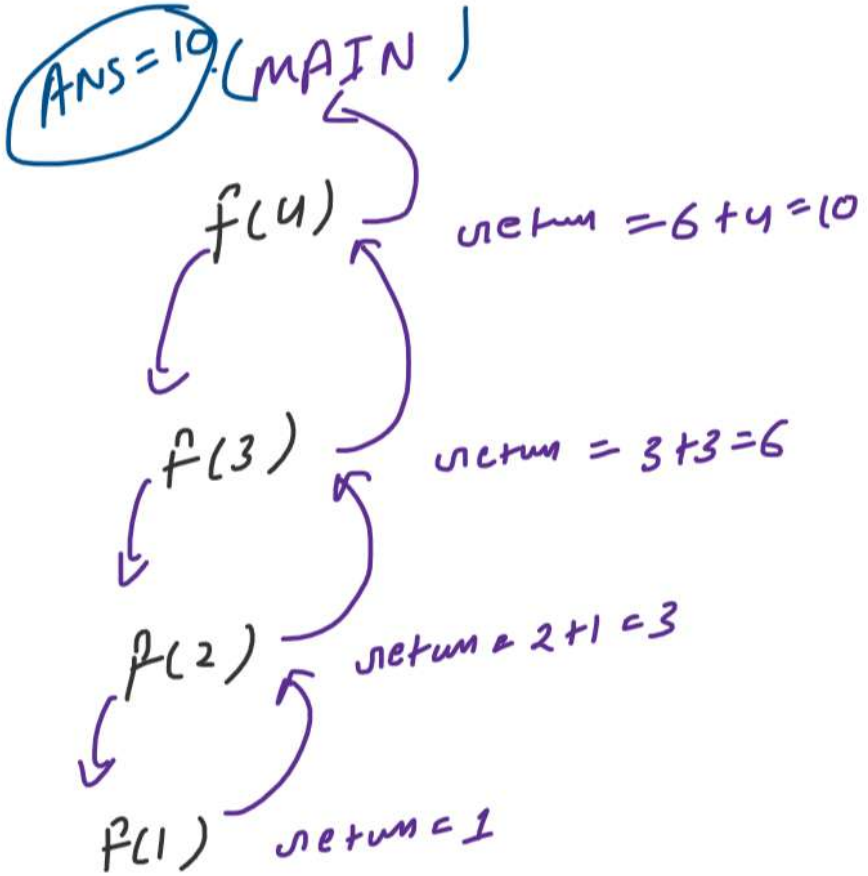
Relation →

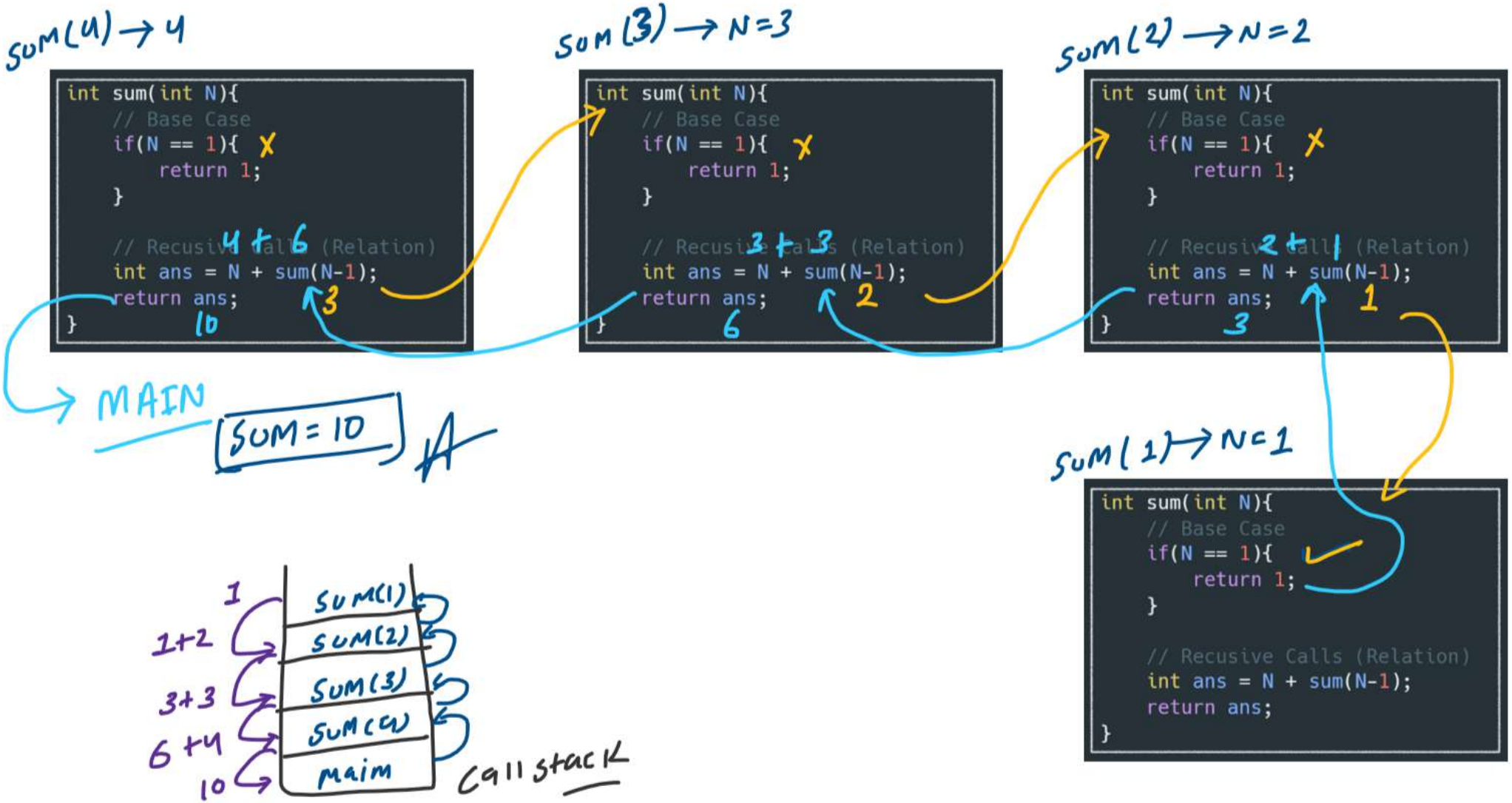
SUM(N) = N + SUM(N-1)

SUM(N) → Bij

SUM(N-1) → Recursion solve kar dega

```cpp
// ✅PROGRAM 05: Return sum from n to 1
#include<iostream>
using namespace std;

int sum(int N){
    // Base Case
    if(N == 1){
        return 1;
    }

    // Recusive Calls (Relation)
    int ans = N + sum(N-1);
    return ans;
}

int main(){
    int sum = 5;
    cout<<"Nth term is "<<sum(N);

    return 0;
}
```




sum(4) → 4
int sum(int N){
// Base Case
if(N == 1){
return 1;
}
// Recusive Calls (Relation)
int ans = N + sum(N-1);
return ans;
}
4 + 6
10
3
sum(3) → N=3
int sum(int N){
// Base Case
if(N == 1){
return 1;
}
// Recusive Calls (Relation)
int ans = N + sum(N-1);
return ans;
}
3 + 3
6
2
sum(2) → N=2
int sum(int N){
// Base Case
if(N == 1){
return 1;
}
// Recusive Calls (Relation)
int ans = N + sum(N-1);
return ans;
}
2 + 1
3
1
MAIN
SUM = 10
sum(1) → N=1
int sum(int N){
// Base Case
if(N == 1){
return 1;
}
// Recusive Calls (Relation)
int ans = N + sum(N-1);
return ans;
}
1
1+2
3+3
6+4
10
sum(1)
sum(2)
sum(3)
sum(4)
main
Call stack