Recursion

Good Evening

Today's content

- Recursion Introduction
- Function call Tracing
- Factorial of a no.
- Increasing
- Decreasing
- Print Decreasing Increasing
- Fibonacii no.

* Recursion → Function calling itself

→ Solving a problem by using a subproblem

a smaller instance of the same problem.

Why Recursion?

→ Pre-requisite for Backtracking, DP, Trees, Graphs

→ Sorting Algo → Merge Sort & Quick sort

* Sum of first 5 natural no.

$$sum(5) = \underbrace{1+2+3+4}_{sum(4)} + 5$$

$$sum(4) = 1+2+3+4$$

$$sum(5) = \underbrace{sum(4)}_{\text{subproblem}} + 5$$

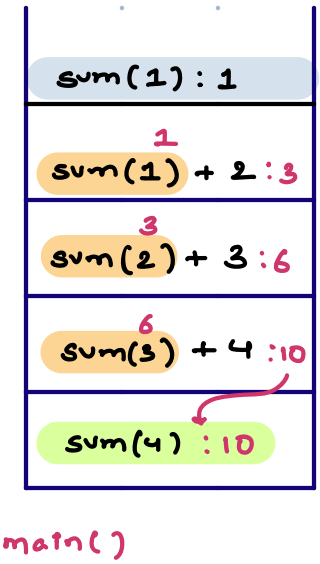
$$sum(n) = \underbrace{1+2+3+\cdots+(n-1)}_{sum(n-1)} + n$$

$$sum(n) = sum(n-1) + n$$

Expectation ⇒ Calculate & return sum from 1 to N

```
int sum(N){
    if (N==1) return 1
    return sum(N-1) + n
}
```

3



Steps to follow to Recursive codes

01 Assumption / Expectation → Decide what our function is supposed to do.

02 Main logic ⇒ Breaking the problem into a subproblem & then using it to solve the larger problem.

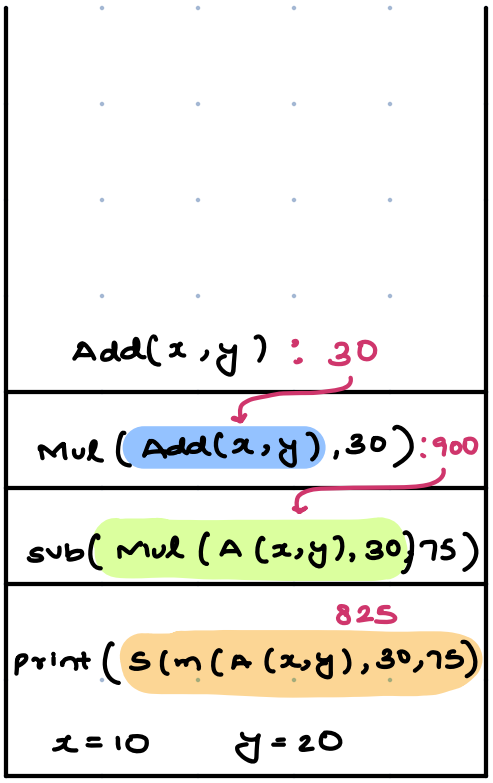
03 Base Case ⇒ Last valid input for which the recursion has to stop

TOPIC 1
Recursion

Function Call Tracing

Code-block

```
int add ( int x, int y ){
      return x + y ;
}
int mul ( int x, int y ){
      return x * y ;
}
int sub ( int x, int y ){
      return x - y ;
}
void print ( int x ){
     print (x) ;
}
```




```
main (){
    x = 10, y = 20 ;
    print ( sub ( mul (add (x,y) , 30) , 75) ; : 825
                ↳ mul ( add (x,y) , 30) : 900
                      ↳ add (x,y)  : 30
}
```

Ans = 825

* Conclusion
* Function will get added on the top
* the child will always return its answer to its parent & once child call is executed, then only parent call will execute.

Q Given a positive integer N, find the factorial of N

$$fact(5) = 5 * \underbrace{4 * 3 * 2 * 1}_{fact(4)} \Rightarrow 120$$

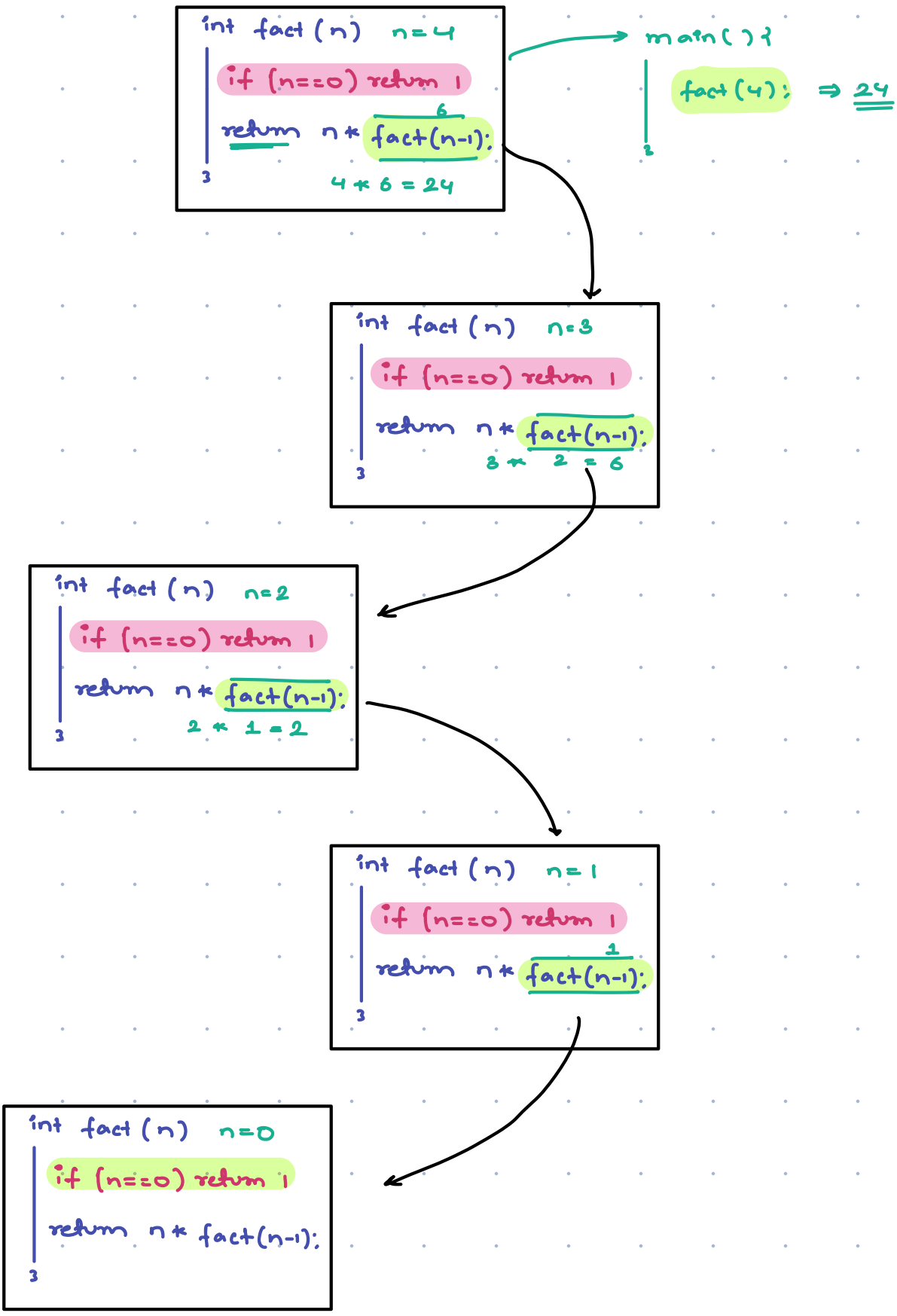
$$fact(n) = n * fact(n-1)$$

n > 0

```
int fact(N){
    if (n==1) return 1;
    return n*fact(n-1);
}
```

n ≥ 0

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


int fact (n) n=4
if (n==0) return 1
return n * fact(n-1);
4 * 6 = 24
main() {
fact(4); ⇒ 24
int fact (n) n=3
if (n==0) return 1
return n * fact(n-1);
3 * 2 = 6
int fact (n) n=2
if (n==0) return 1
return n * fact(n-1);
2 * 1 = 2
int fact (n) n=1
if (n==0) return 1
return n * fact(n-1);
int fact (n) n=0
if (n==0) return 1
return n * fact(n-1);

Q Given N, print all no. from 1 to N in increasing order

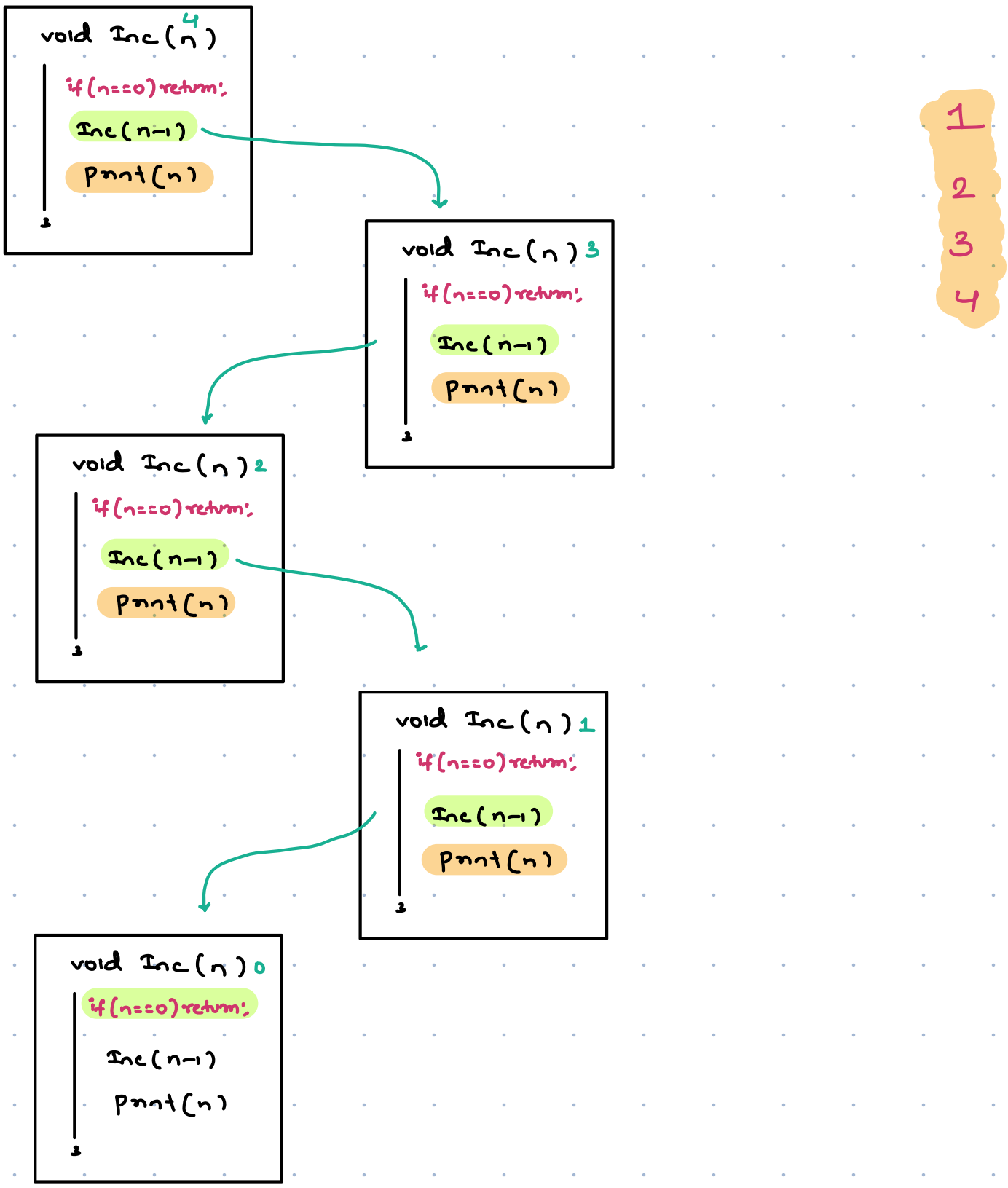
Inc(5)	Subproblem Inc(4)	Main logic
1	1	Inc(4)
2	2	5
3	3	
4	4	
5		

Inc(5) = Inc(4)
print(5);

Inc(n) = Inc(n-1)
print(n)

```
void Inc(n){
    if(n==0) return;
    Inc(n-1)
    print(n);
}
```
2

```
                1   n
void incr(a, b)
    if(a>b) return;
    print(a)
    incr(a+1, b);
```
3


void Inc(4)
if(n==0)return;
Inc(n-1)
Print(n)
3
void Inc(3)
if(n==0)return;
Inc(n-1)
Print(n)
3
void Inc(2)
if(n==0)return;
Inc(n-1)
Print(n)
3
void Inc(1)
if(n==0)return;
Inc(n-1)
Print(n)
3
void Inc(0)
if(n==0)return;
Inc(n-1)
Print(n)
3
1
2
3
4

Problem

Whirlpool wants to design a timer for their **washing machines**. This feature is a simple countdown timer. When a user sets a time, for example, 10 minutes, the washing machine needs to show each minute passing, counting down until it reaches 0.

Your task is to write a program that takes an integer **A** (the time in minutes set by the user) and then prints out each minute as it counts down to **0**.The requirement is that after a user set a timer for the washing machine for some time say **A** , the washing machine should display each minute after that decremented one by one till the time becomes **0**.

```
void dec (int n){
    if (n<0) return;
    print (n)
    dec(n-1);
}
```

code will print 0

```
if (n==0){
    print(n);
    return;
}
```

Another valid base case

10:25 → 10:35 pm

* Time & Space Complexity

Time complexity → Time taken by our code

Space complexity → Amount of calls that we are making in recursive stack

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

$T(0) = 1$

$T(N)$ → Time taken by recursion to calculate fact(n)

$T(N-1)$ → Time taken by recursion to calculate fact(n-1)

$T(0) = 1$

$T(n) = T(n-1) + 1$ } Recurrence relation

$T(n) = T(n-1) + 1$ ⟶ 1st subs

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

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

$T(n) = T(n-2) + 2$ ⟶ 2nd subs

$T(n-2) = T(n-3) + 1$

$T(n) = T(n-3) + 1 + 2$

$T(n) = T(n-3) + 3$ ⟶ 3rd substitution

After k substitution

$T(n) = T(n-k) + k$

$T(0) = 1$

$n - k = 0$

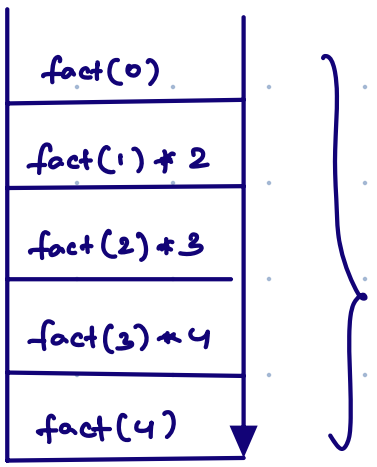
$k = n$

$T(n) = 1 + n$

$T(n) = n + 1$

TC : O(n)

SC : O(n)



fact(n) = n+1 calls

Q Given N, print all the no. from N to 1 in decreasing order & then 1 to N in increasing order.

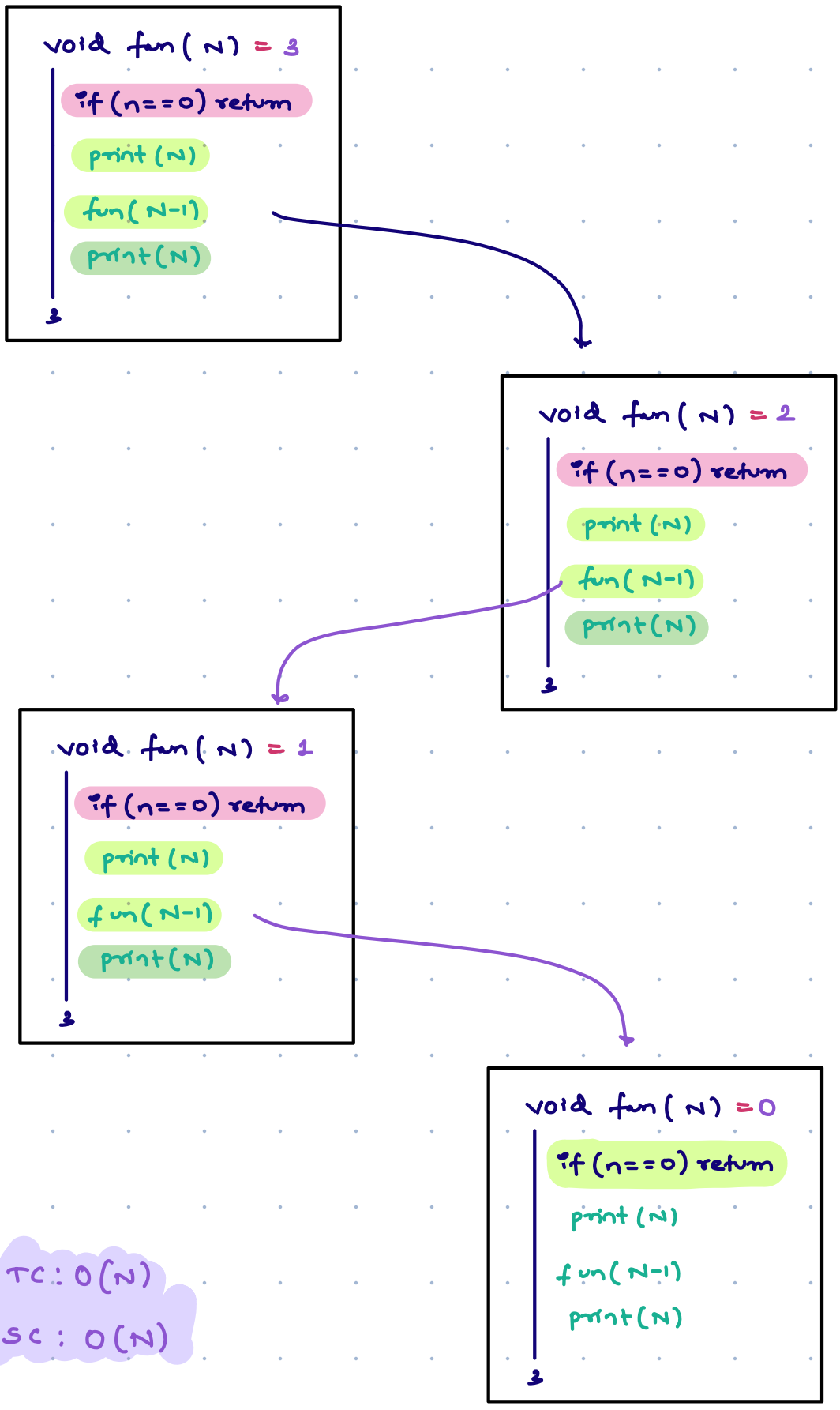
N = 3 ⇒ 3 2 1 1 2 3

```
void fun(int n){
    if (n==0) return;
    print(N)        1
    fun(n-1);       2
    print(N);       3
}
```

```
if (N==1){
    print(N)
    return
}
```

base case when we wants to print a single 1

3
2
1
1
2
3



TC : O(N)

SC : O(N)

* Fibonacci series

N	=	0	1	2	3	4	5	6	7
fib( )		0	1	1	2	3	5	8	13

fib(N) = sum of previous 2 fib no.

fib(5) = fib(4) + fib(3)

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

// Assumption → Calculate & return the n^th fib no.

```
int fib(N){
    if (N==0) return 0
    if (N==1) return 1

    return fib(N-1) + fib(N-2);
}
```

if (N≤1) return N

Base case → last valid input for which recursion has to stop

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

N = 1 fib(0) + fib(-1)

N = 2 fib(1) + fib(0)

Time & Space complexity

// Assumption = fib(N) is going to take T(N) time.

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

??

$$T(N) = T(N-1) + T(N-1) + 1$$

$$T(N) = 2T(N-1) + 1 = 2^1 T(N-1) + 2^1 - 1$$

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

$$T(N) = 2 * (2T(N-2) + 1) + 1$$

$$T(N) = 4T(N-2) + 3 = 2^2 T(N-2) + 2^2 - 1$$

$T(N-2) = 2T(N-3) + 1$

$$T(N) = 4 * (2T(N-3) + 1) + 3$$

$$T(N) = 8T(N-3) + 7 = 2^3 T(N-3) + 2^3 - 1$$

After k substitution

$$T(N) = 2^k T(N-k) + 2^k - 1$$

$T(1) = 1$

$N - k = 1$

$k = N - 1$

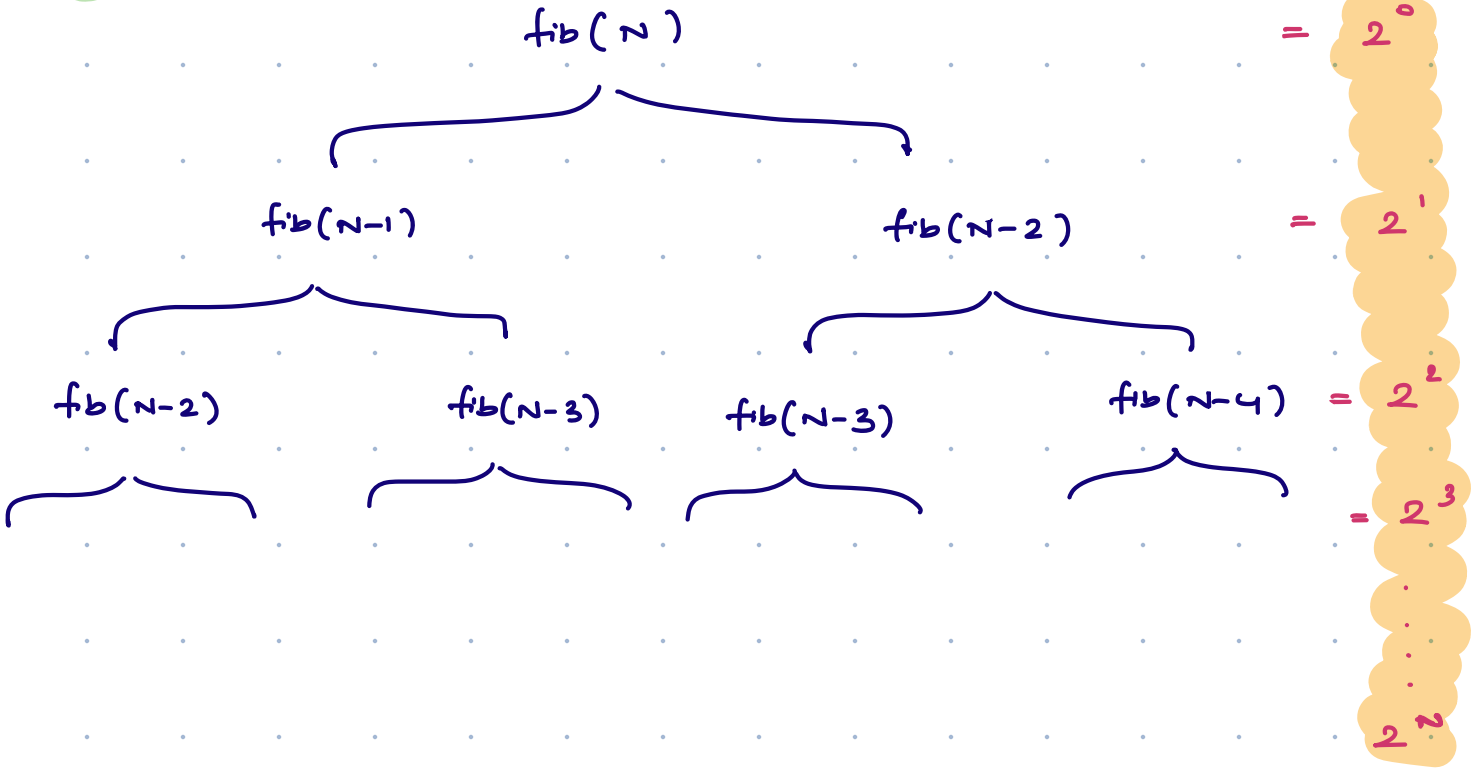
$$T(N) = 2^{n-1} * 1 + 2^{n-1} - 1$$

$$T(N) = 2^{n-1} + 2^{n-1} - 1$$

TC : $O(2^n)$

Tree Approach to calculate TC

TC = Time taken in 1 call * Total fn calls



Total Fn calls = $2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^N$

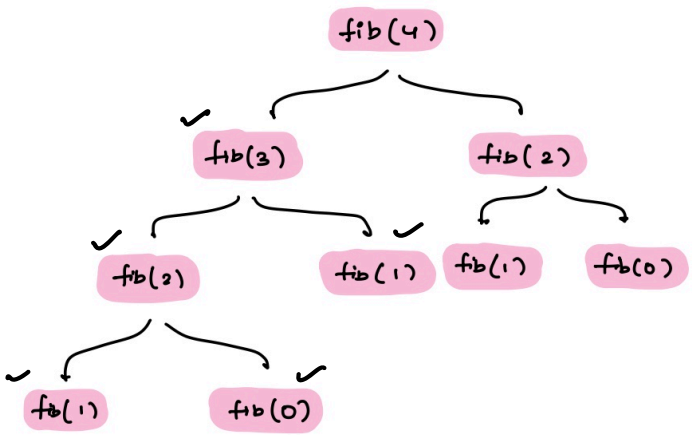
Sum of GP = $\frac{a(r^n - 1)}{r - 1}$

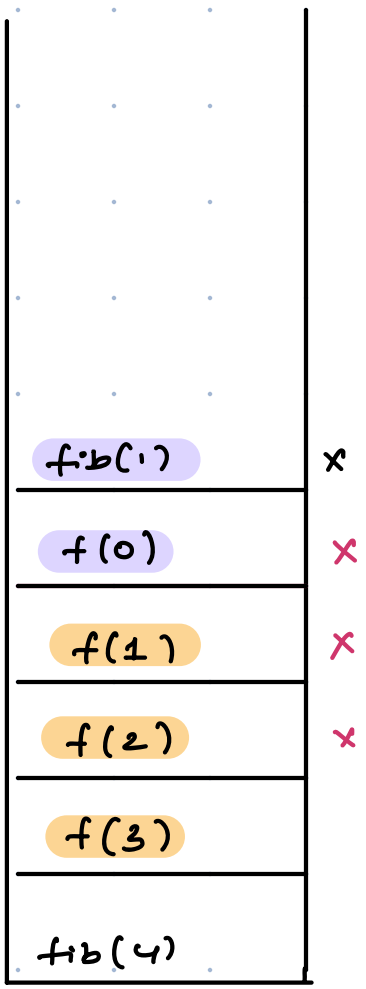
$$= \frac{2^0 * (2^{n+1} - 1)}{2 - 1}$$

$$TC = 2^{n+1} - 1$$

$TC \approx O(2^n)$

Space complexity






```
int fib(N){
    if (N<=1) return N
    return fib(N-1) + fib(N-2);
}
```

Max calls = 4

TC : $O(2^n)$

SC : $O(n)$

↓

1 = 0 0 1

2 = 0 1 0

3 = 0 1 1

4 = 1 0 0

$\delta 2 = 4$

subarrays = $\frac{4 * 4+1}{2}$

1 = {0 0 1} = 1

1, 2 = {0 1 1} = 3

1, 2, 3 = {0 1 1} = 3

1, 2, 3, 4 = {1 1 1}

2 = {0 1 0}

2, 3 = {0 1 1}

2, 3, 4 = {1 1 1}

3 = {0 1 1}

3, 4 = {1 1 1}

4 = {1 0 0}
