Recursion

"We achieve more when we chase the dream instead of the competition."

Simon Sinek

Good Morning

Recursion - 1

TABLE OF CONTENTS

Why Recursion?

→ Pre-requisite of Backtracking , Trees , D.P , Graphs

→ Sorting algo's [Merge , Sort , Quick Sort]

Steps to follows for Recursive code

01. Assumption / Expectation → Decide what your fnction is suppose to do.

02. Main logic → Breaking the problem into sub problem & using it to solve larger problem

03 Base case → Last valid input for which the recursion has to stop.

Recursion

→ Function calling itself

→ Repetition of same task

→ Repetitive using the same pattern

↳ Solving a bigger problem using the smaller instances of the same problem

subproblem

Sum of first N natural no's

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

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

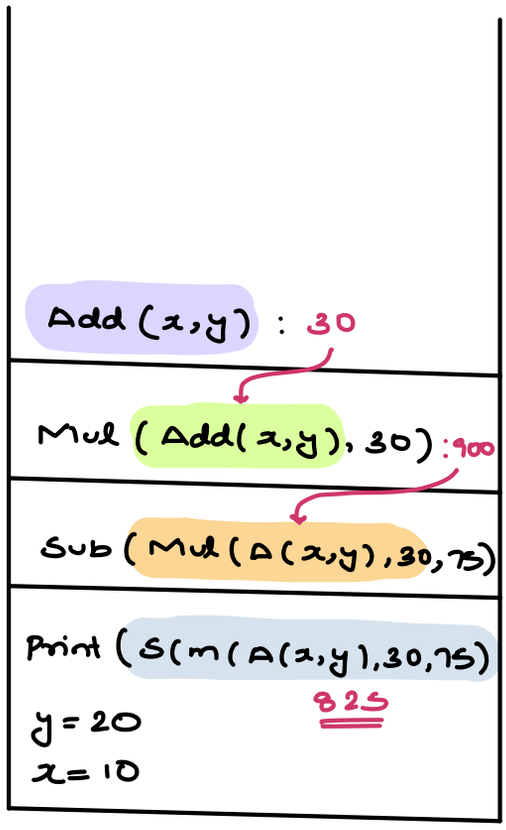
$$sum(n) = \underbrace{1+2+3+4+\ldots+(n-1)}+n$$

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

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
(900)

↳ (mul (add(x,y), 30) = 900
(30)

↳ add (x, y) = 30

Ans = 825

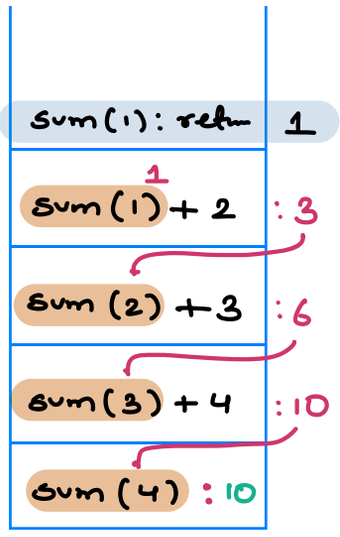
}

01. Function will get added at the top.

02 Once a child call is executed, then only parent call will resolve.

// Assumption → Calculate & return sum from 1 to N

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



N Factorial = Given N (whole number), find the factorial of N

fact(5) = 5 * 4 * 3 * 2 * 1 = 120

fact(4)

fact(n) = n * (n-1) * (n-2) * ... * 3 * 2 * 1

fact(n-1)

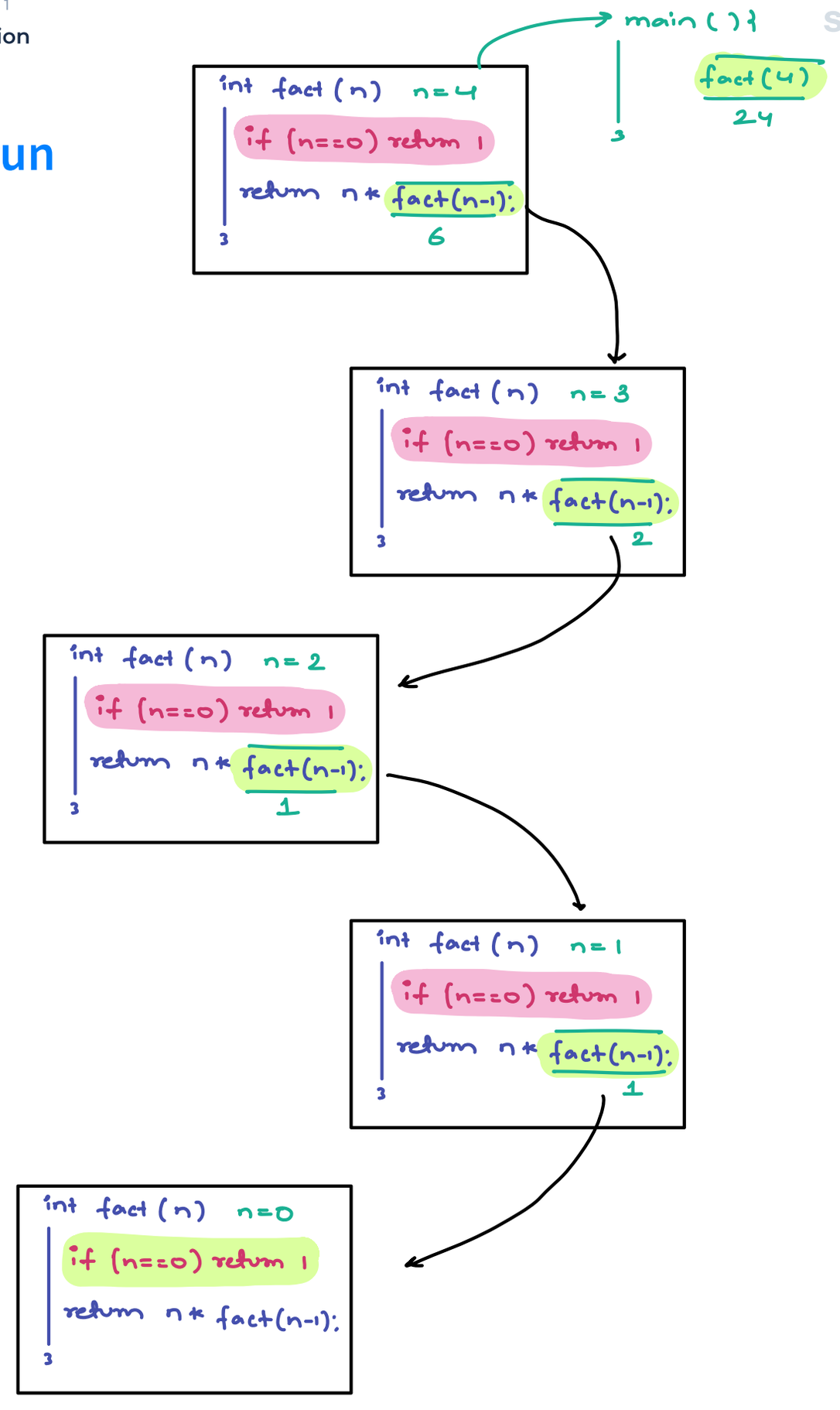
N > 0

```
int fact (n)
    if (n==1) return 1
    return n * fact(n-1);
3
```

N ≥ 0

```
int fact (n)
    if (n==0) return 1
    return n * fact(n-1);
3
```

dry-run



Nth Fibonacci

N =	0	1	2	3	4	5	6	7	8	9	10
F.b →	0	1	1	2	3	5	8	13	21	34	55

Given value of N. Write a recursive function of find N^{th} fibonacci number.

fib(n) = sum of previous two fib no.

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

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

// Assumption → Calculate & return n* fib no.

```
int fib (int N)
{
    if (N==0) return 0;
    if (N==1) return 1;
    return fib(n-1) + fib(n-2);
}
```

if (N ≤ 1) return N

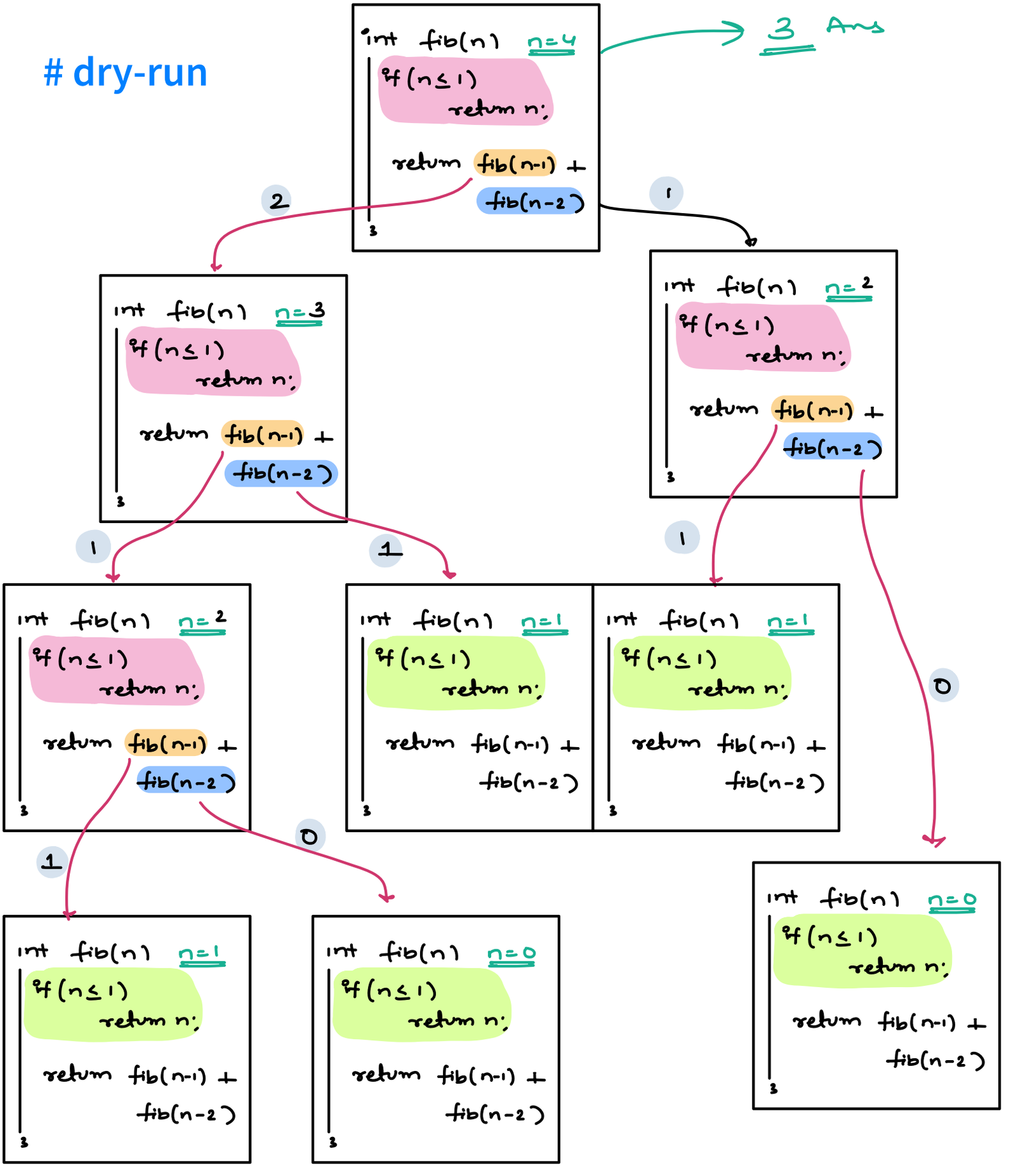
Base case = Last valid input for which recursion has to stop.

N = 0 → fib(-1) + fib(-2)

N = 1 → fib(0) + fib(-1)

N = 2 → fib(1) + fib(0)

dry-run



8:17 AM ⟶ 8:27 AM

Time Complexity of Recursion - Using Recurrence relation

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

T(N-1)

T(N) → time taken to calculate factorial of N.
time taken to calculate factorial of N-1 => ? T(N-1)

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

$T(n) = T(n-1) + 1$ → 1st substitution

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

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

$T(n) = T(n-2) + 2$ → 2nd substitution

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

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

$T(n) = T(n-3) + 3$ → 3rd subs

After K' substitution

$T(n) = T(n-K) + K$ $\quad T(0) = 1$

$n-K=0$

$K=n$

$T(n) = T(0) + n$ $\quad T(n) = 1 + n \approx n$

TC = O(n)
SC : O(n)

T.C Fibonacci

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

T(N-1) T(N-2)

Recurrence relation ⟶

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

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

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

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

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

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

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

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

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

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

After k substitution

$$T(n) = 2^k T(n-k) + 2^k - 1 \qquad T(1) = 1$$

$$n - k = 1$$

$$k = n - 1$$

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

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

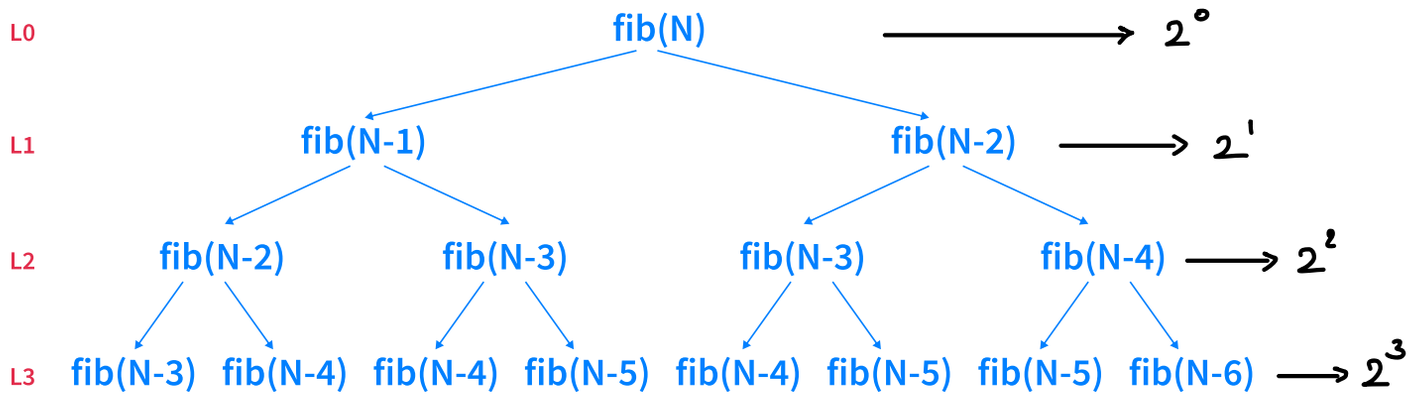
$$= 2^n - 1$$

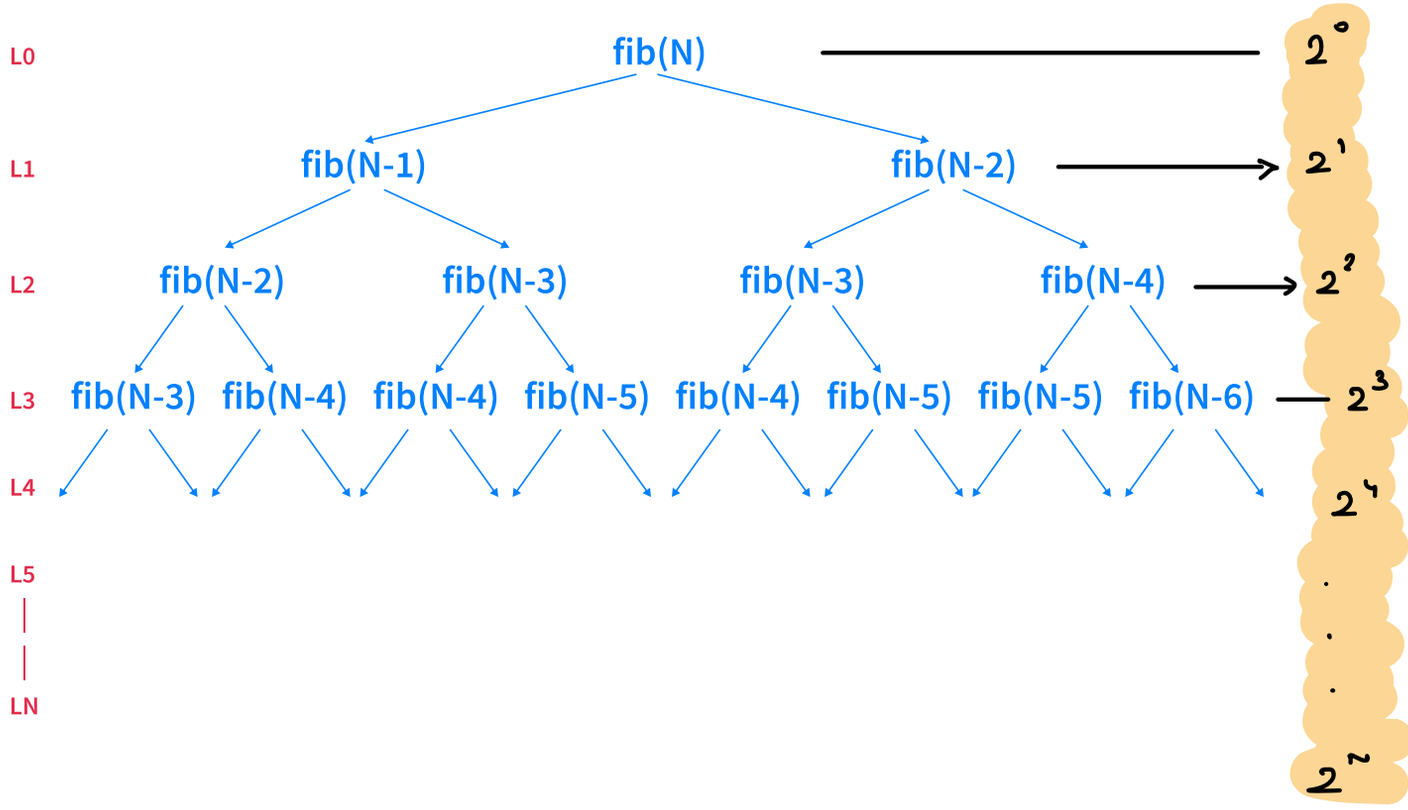
$$T(n) \leq O(2^n)$$

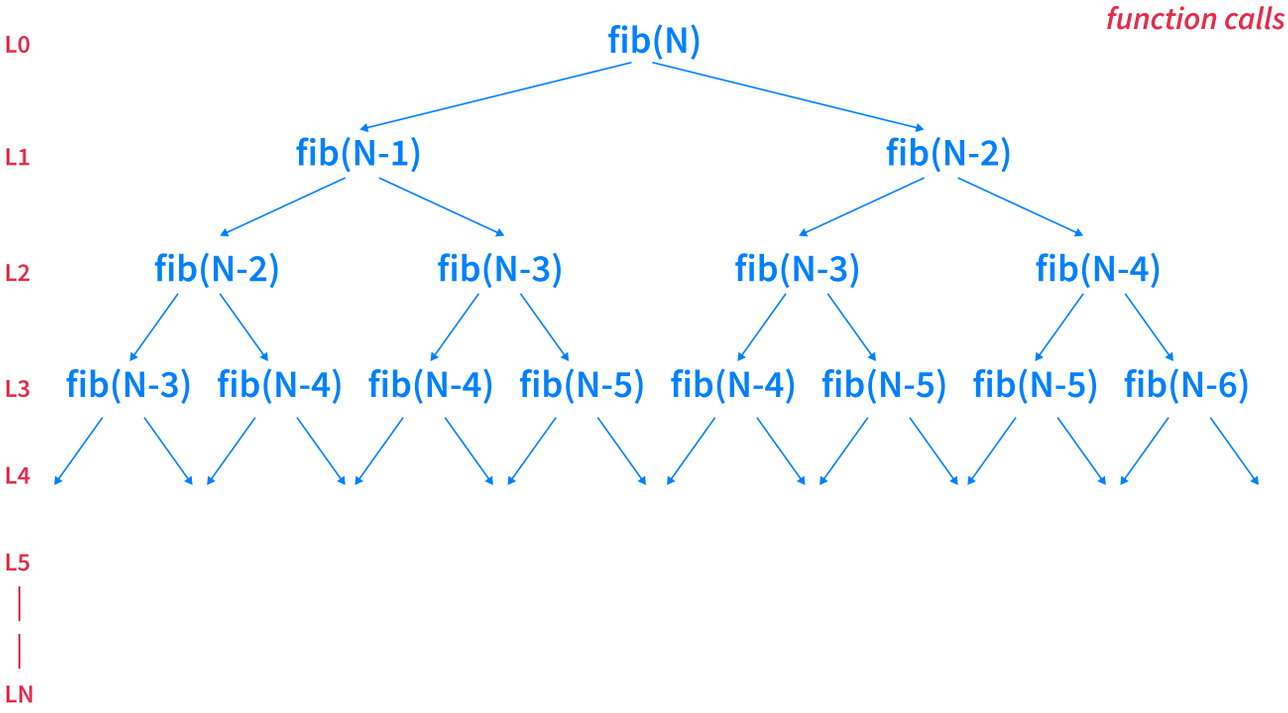
* Another definition of Time complexity

TC = Time taken on a single fn call *

No. of function calls


L0
fib(N)
2^0
L1
fib(N-1)
fib(N-2)
2^1
L2
fib(N-2)
fib(N-3)
fib(N-3)
fib(N-4)
2^2
L3
fib(N-3)
fib(N-4)
fib(N-4)
fib(N-5)
fib(N-4)
fib(N-5)
fib(N-5)
fib(N-6)
2^3


L0
fib(N)
2^0
L1
fib(N-1)
fib(N-2)
2^1
L2
fib(N-2)
fib(N-3)
fib(N-3)
fib(N-4)
2^2
L3
fib(N-3)
fib(N-4)
fib(N-4)
fib(N-5)
fib(N-4)
fib(N-5)
fib(N-5)
fib(N-6)
2^3
L4
2^4
L5
LN
2^N



total function calls :

$$\text{Total calls} = 2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^n$$

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

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

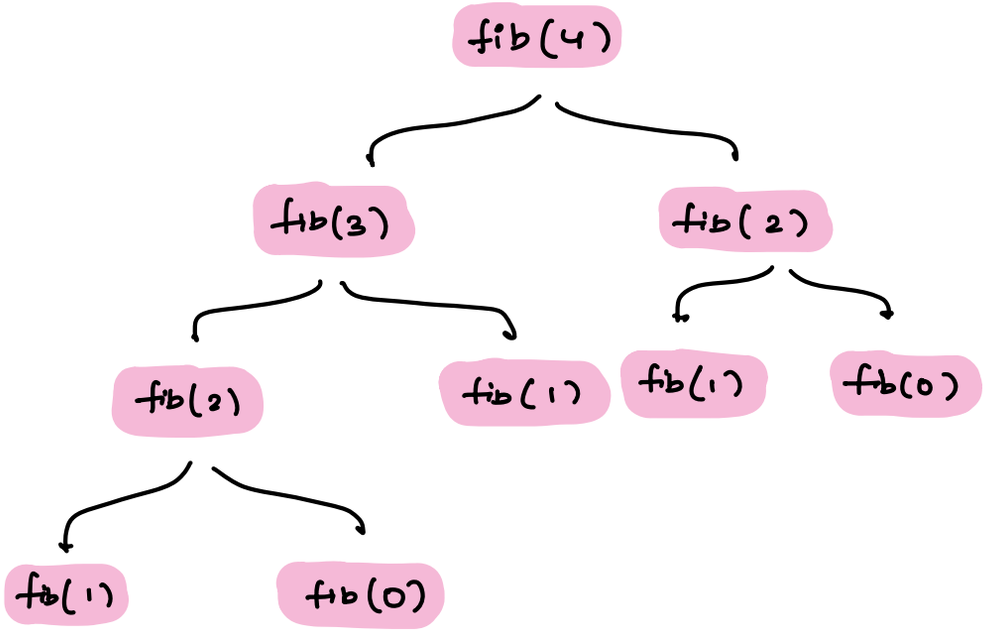
$$= (2^{n+1} - 1)$$

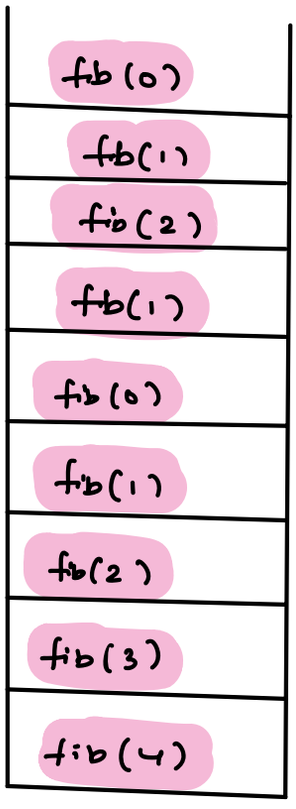
$$TC = (2^{n+1} - 1) * 1$$

$$= O(2^n)$$

Space Complexity

⇒ Max Amount of calls we put in our stack at any point of time is our space complexity of recursive function.



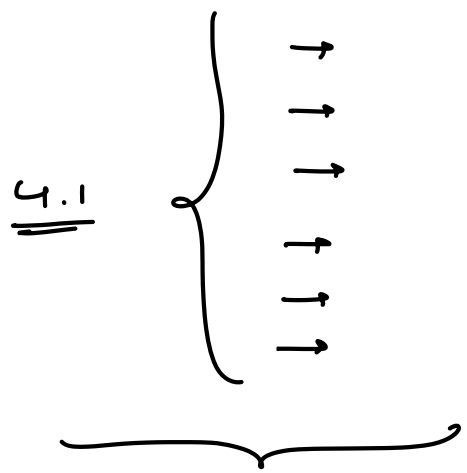


At max, 4 calls

TC : $O(2^n)$

SC : $O(n)$

DSA 4.2



DSA 4.2
Mock + Contest

Backtracking
Inverse modulo
String pattern matching
→ DP on strings
→ Graphs Algo

Subarray OR

$\{1, 2, 3, 4\}$

1 = 0 0 1
2 = 0 1 0 = $\frac{1*(1+1)}{2}$
3 = 0 1 1
4 = 1 0 0 = $\frac{1*(1+1)}{2}$

$\{1\}$ = 0 0 1 $\quad \frac{n*(n+1)}{2}$
$\{1,2\}$ = 0 1 1
$\{1,2,3\}$ = 0 1 1
$\{1,2,3,4\}$ = 1 1 1
$\{2\}$ = 0 1 0
$\{2,3\}$ = 0 1 1
$\{2,3,4\}$ = 1 1 1

$\{1, 3, 4, 6\}$
+
$\{1, 2, 3, 4, 5, 6\}$

```
for ( i=0; i<n; i++){
    xor= xor ^ A[i]
}
```

{3} = 0 1 1

{3, 4} = 1 1 1

{4} = 1 0 0

```
for ( i=1; i≤n+2; i++){
    xor = xor ^ i ;
}
```

xor = xor of two distinct ele

Min XOR value ⇒ Sorting can help

{ 2, 10, 6, 8 }

{ 2, 6, 8, 10 }

2 ^ 6

6 ^ 8

8 ^ 10

} Answer will lie in these three option