SCALER

Recursion - 1

TABLE OF CONTENTS

Class starts at 7:05 IST

Why Recursion ?

- Pre-requisite of Backtracking , Trees , D.P , Graphs
- Sorting algo’s [Merge , Sort , Quick Sort]

Recursion

- Function calling itself.
- Breaking bigger problems into smaller problems (subproblems) and the solution of bigger problem is generated using subproblem.

Sum of first N natural numbers

N = 5

sum(5) = 1 + 2 + 3 + 4 + 5

sum(4)

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

sum(5) = sum(4) + 5

Subproblem

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

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

Base Case : N = 0

return 0

Steps to write recursive code

1. ASSUMPTION : Decide what your function should do & assume that it is working for subproblems.

2. MAIN LOGIC : Breaking the problem into subproblem and solve the bigger problem using subproblem.

3. BASE CASE : Input for which recursion should stop.

Function Call Tracing

Tracking the sequence of function calls that are made when a program is executed.

```
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) {
    cout << x << endl;
}

int main() {
    int x = 10;
    int y = 20;
    print(sub(mul(add(x, y), 30), 75));
    return 0;
}
```

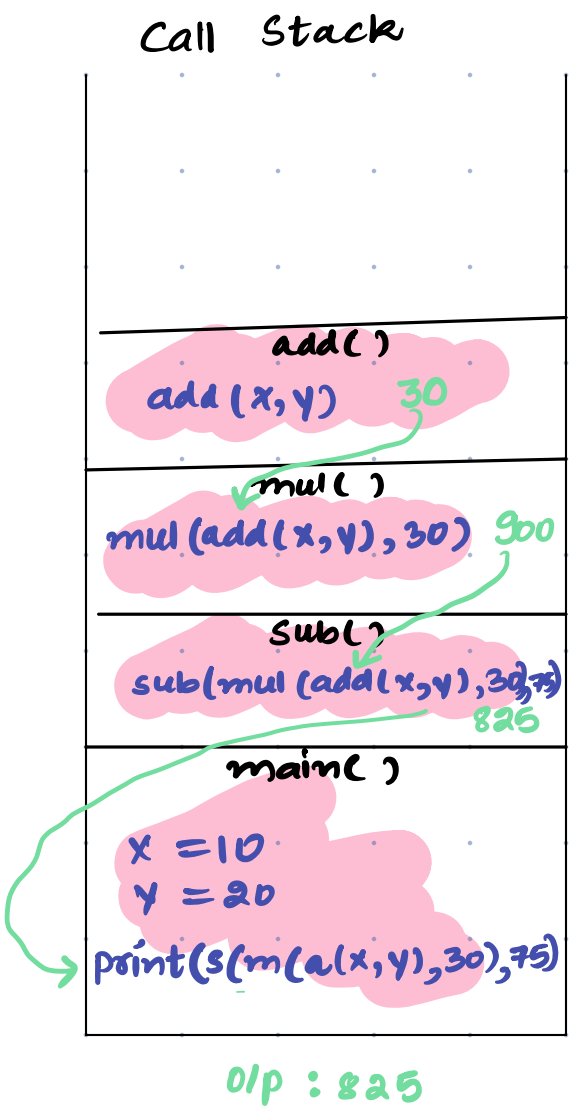
add(10, 20) → 30

mul(30, 30) → 900

sub(900, 75) → 825

print(825)

o/p : 825



o/p : 825

N Factorial

Given a whole number N, find the factorial of N.
Note : 0! = 1 0, 1 - - -

$N! = N*(N-1)*(N-2) - - - - 2*1$

Quiz : 1

$5!$ $5*4*3*2*1 = 120$

Assumption : Calculate & return factorial of a no.

Main logic :

$N! = N*\overbrace{(N-1)*(N-2) - - - - - 2*1}^{(N-1)!}$

$(N-1)! = (N-1)*(N-2) - - - - - 2*1$

$N! = N*(N-1)!$

Base Case : I/p for which expression becomes invalid.

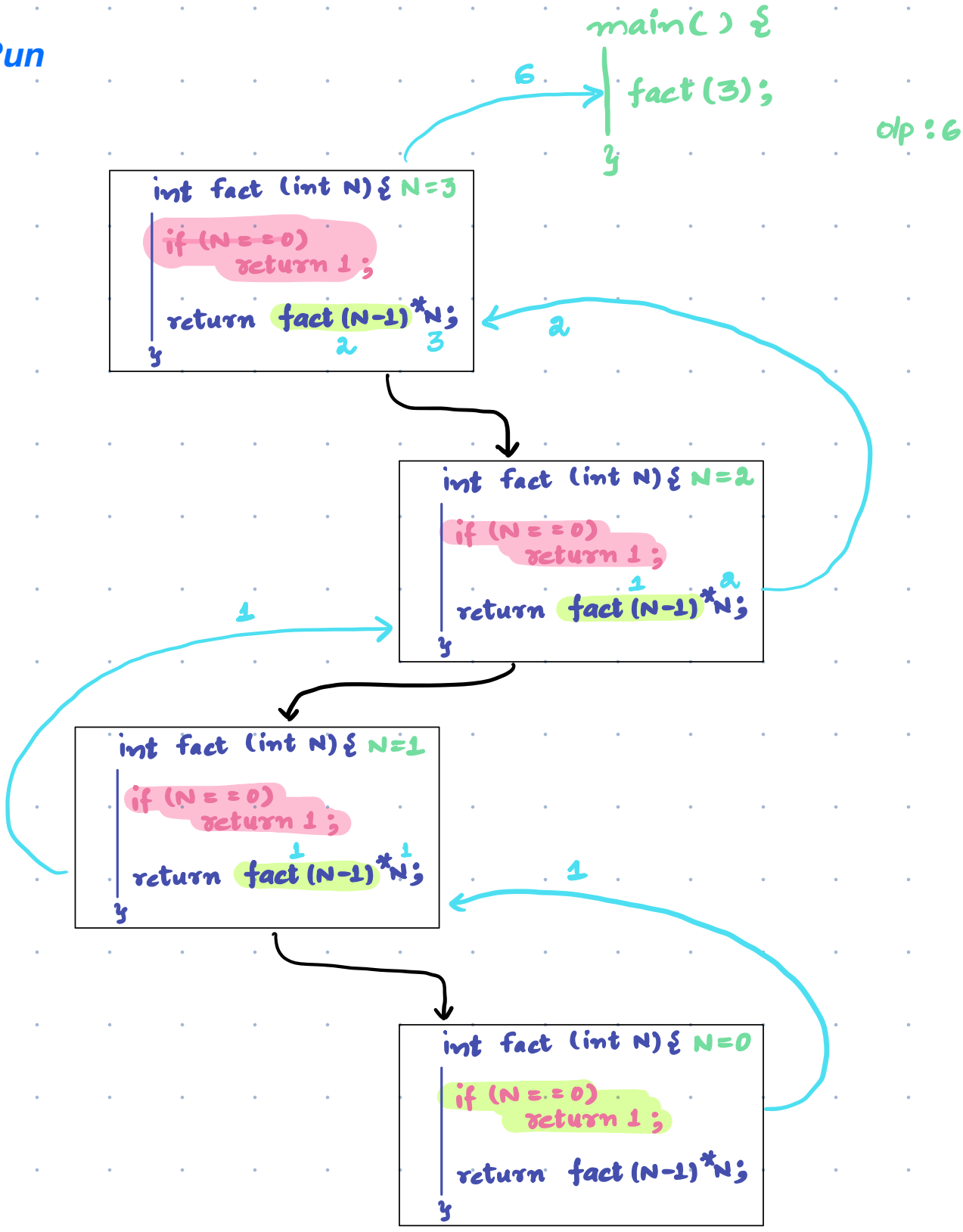
$N = 1$ $\quad N! = N*(N-1)! = 1*0! = 1$

$N = 0$ $\quad N! = 0(-1!)$ ✗ return 1

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

    return fact(N-1)*N;
}
```

*Dry-Run



Nth Fibonacci

The fibonacci sequence is a series of numbers in which each number is the sum of two preceding numbers.

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 positive integer N. Write a function of find Nth number in fibonacci sequence using recursion.

Quiz : 2

7th fibonacci no : 13

Assumption : Calculate & return Nth fibonacci no.

Main logic : fib(N) = fib(N-1) + fib(N-2)

Base case :

N = 2 fib(2) = fib(1) + fib(0) ✓

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

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

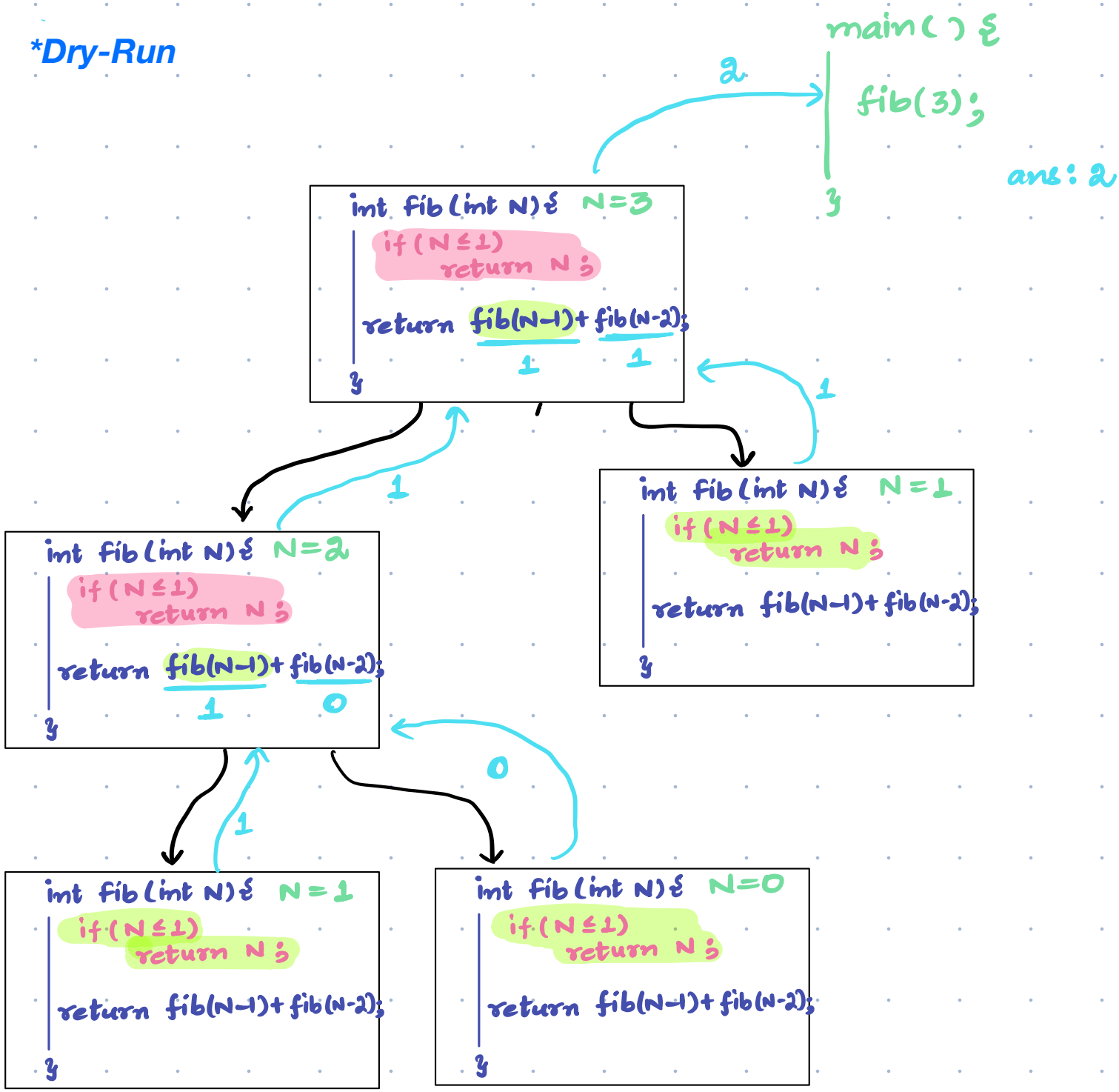
fib(1) = 1

fib(0) = 0

```
int fib(int N){
    if(N<=1)
        return N;

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

*Dry-Run



Time Complexity of Recursion [Using Recurrence relation]

Time Complexity of Factorial

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

T(N) → time taken to calculate factorial of N.

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

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

Recurrence Relation

$$T(0) = c'$$

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

Substitute $N \to N-1$ in recurrence relation

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

$T(N) = [T(N-2) + C] + C$

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

Substitute $N \to N-2$ in recurrence relation

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

$T(N) = [T(N-3) + C] + 2C$

$T(N) = T(N-3) + 3C$

After k substitutions

$T(N) = T(N-k) + kC$

$T(0) = c'$

$N - k = 0 \Rightarrow N = k$

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

$T(N) = T(0) + NC = c' + NC$

$T(N) = O(N)$

Time complexity of Fibonacci

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

T(N) → time taken to calculate Nth fibonacci no.

time taken to calculate (N-1) fibonacci no T(N-1)

time taken to calculate (N-2) Fibonacci no T(N-2)

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

Recurrence Relation

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

Substitute N → N-1 in recurrence rel^n

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

$N \rightarrow N-2$

$$T(N-2) = T(N-3) + T(N-4) + c$$

$$T(N) = [T(N-2) + T(N-3) + c] + [T(N-3) + T(N-4) + c] + c$$

$$T(N) = T(N-2) + 2T(N-3) + T(N-4) + 3c$$

$$T(N-3) = T(N-4) + T(N-5) + c$$

$$T(N-4) = T(N-5) + T(N-6) + c$$

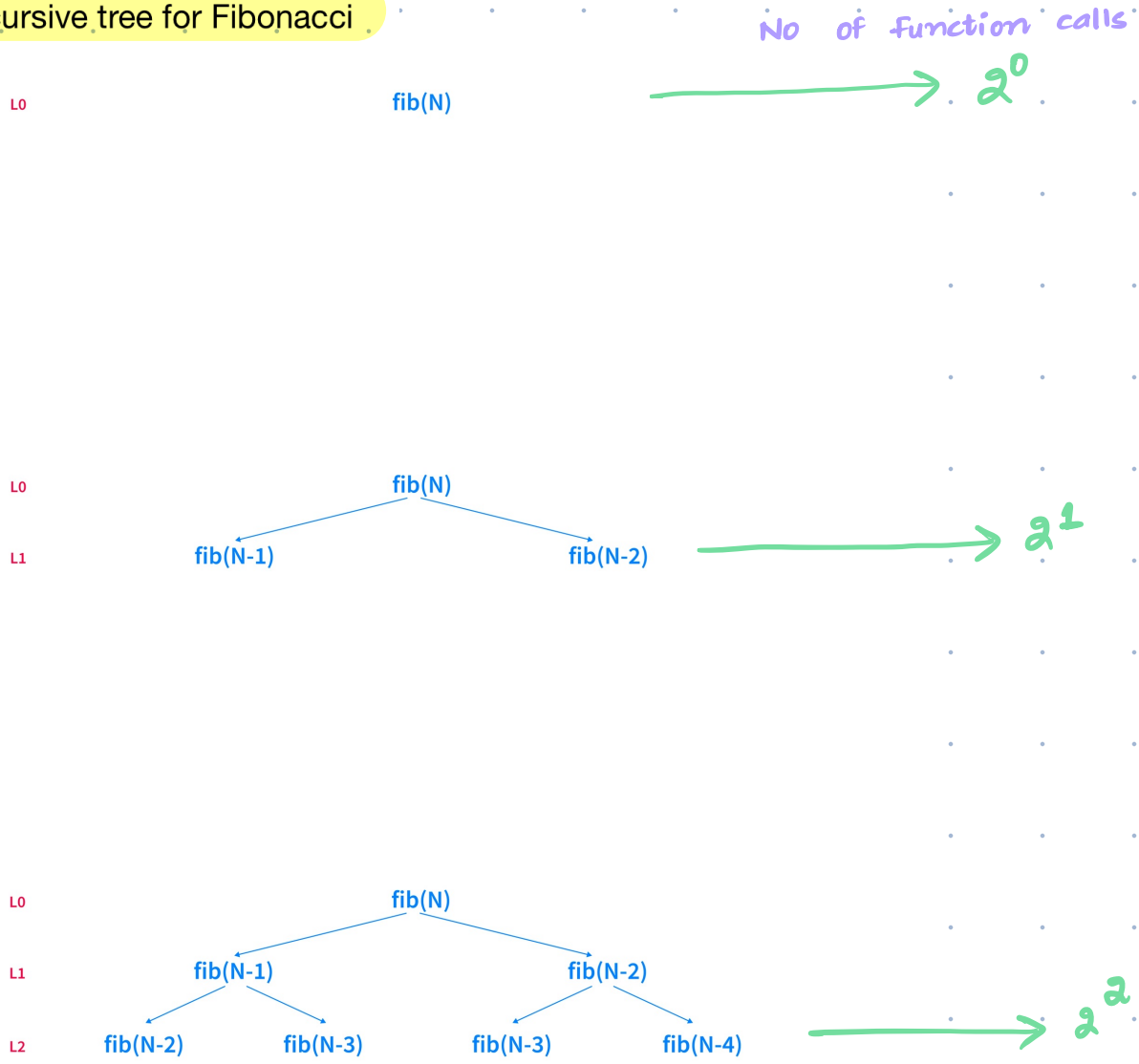
$$T(N) = [T(N-3) + T(N-4) + c] + 2[T(N-4) + T(N-5) + c] + [T(N-5) + T(N-6) + c] + 3c$$

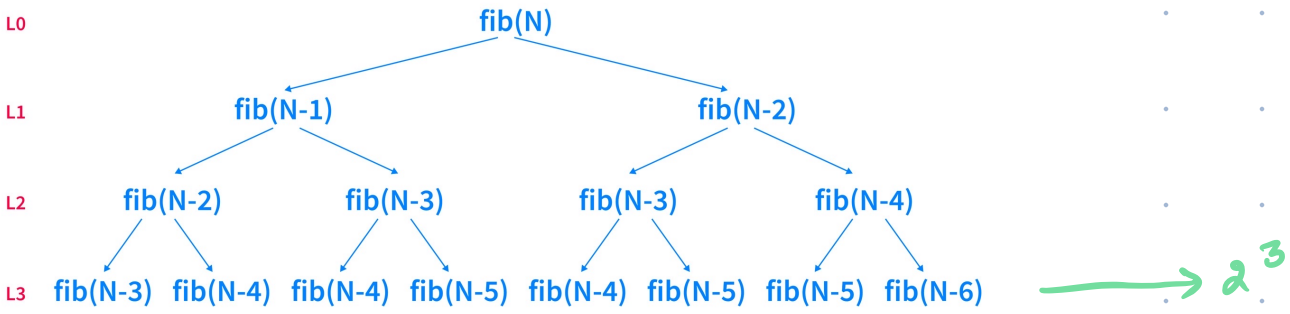
$$T(N) = T(N-3) + 3T(N-4) + 3T(N-5) + T(N-6) + 7c$$

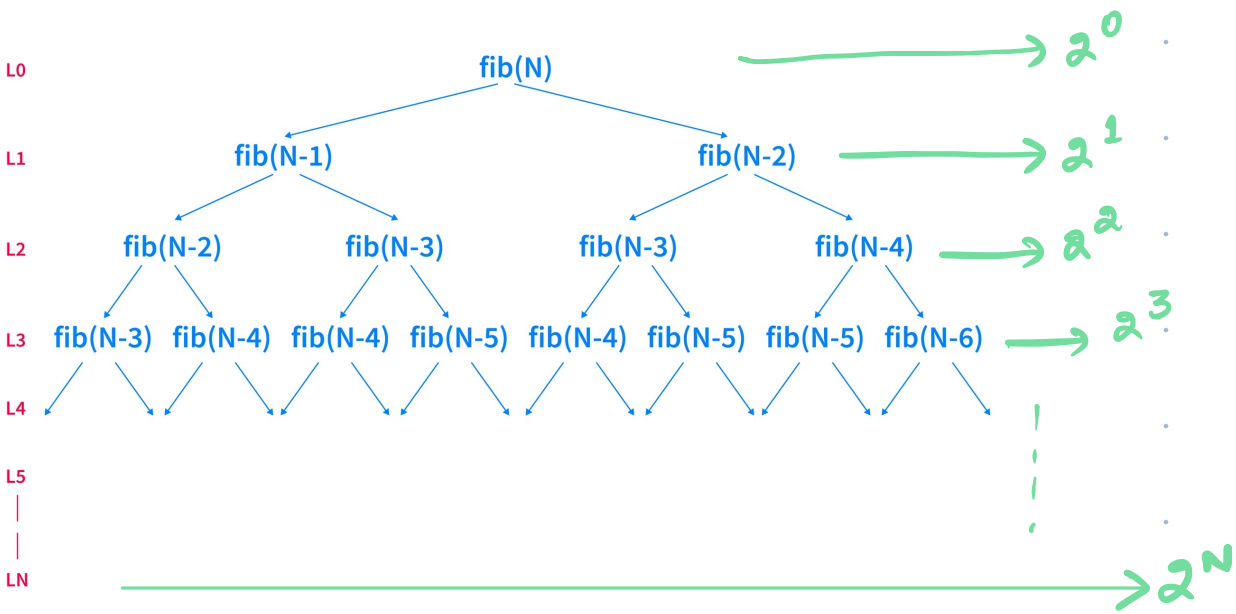
Another definition of Time Complexity

T.C = Total No of function calls * Time taken in every func. call

Recursive tree for Fibonacci







Total function calls: $2^0 + 2^1 + 2^2 + 2^3 - \cdots - 2^N$

Sum of a GP $= a\left[\frac{r^n - 1}{r - 1}\right]$

$a = 2^0 = 1 \quad r = 2^1/2^0 = 2$

0 .... N
N+1 terms

Total function calls: $1\left[\frac{2^{N+1} - 1}{2 - 1}\right] = 2^{N+1} - 1$

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

T.C = Total function calls × Time per call

$= (2^{N+1} - 1) \cdot C$

$= 2^{N+1} \cdot C - C$

$= 2^N \cdot 2 \cdot C - C$

$= 2^N \cdot 2C - C$

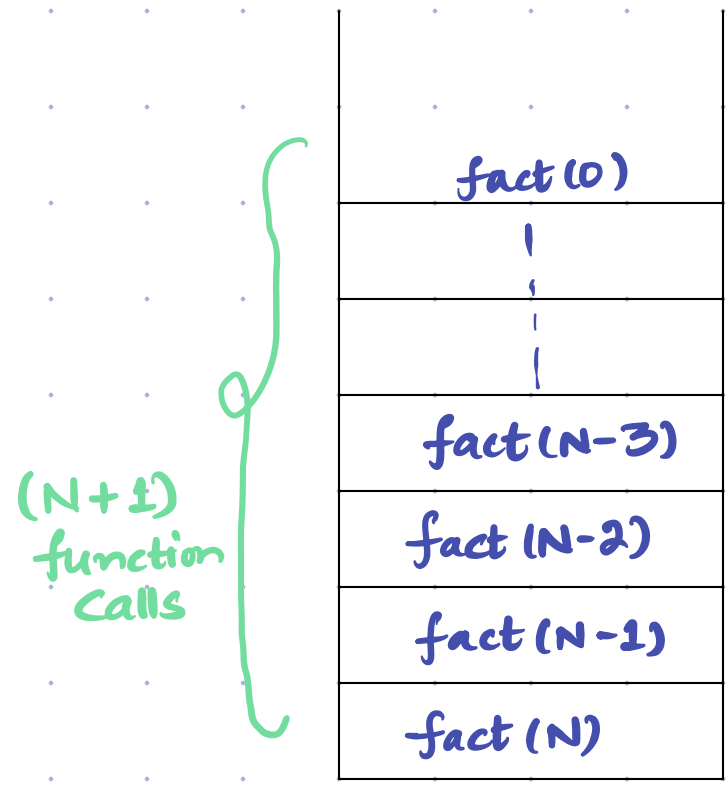
$T.C = O(2^N)$

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

$T.C = (N+1) * C$

$= NC + C$

$T.C = O(N)$



(N+1) func at max

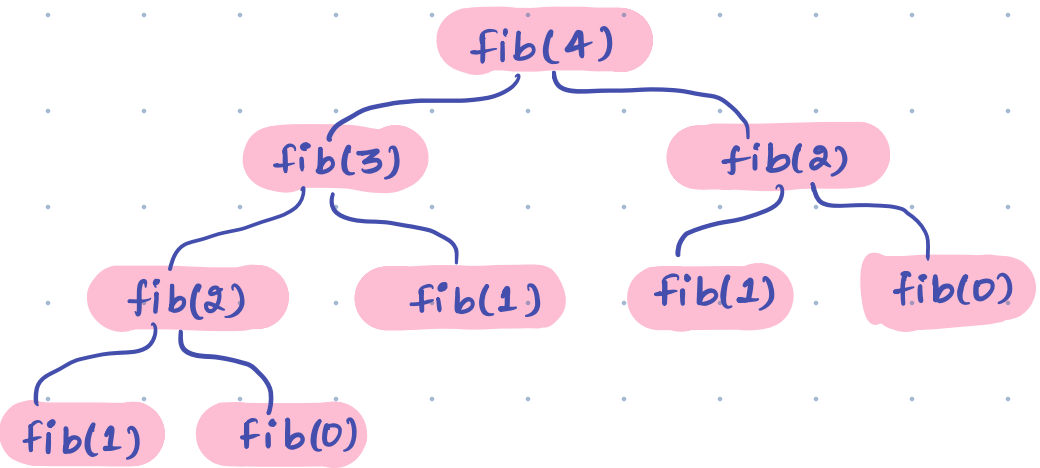
S.C = O(N+1)

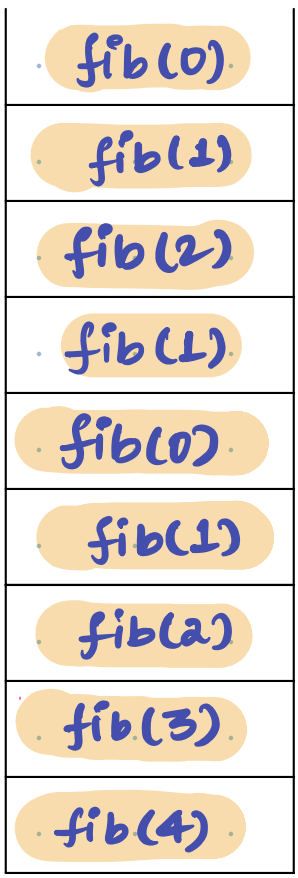
S.C = O(N)

Space Complexity

Max amount of stack space used by our algorithm.

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





N Max calls in stack

S.C = O(N)

max : 4

N = 4

Print numbers in increasing order

Given N, print all numbers from 1 to N in increasing order.

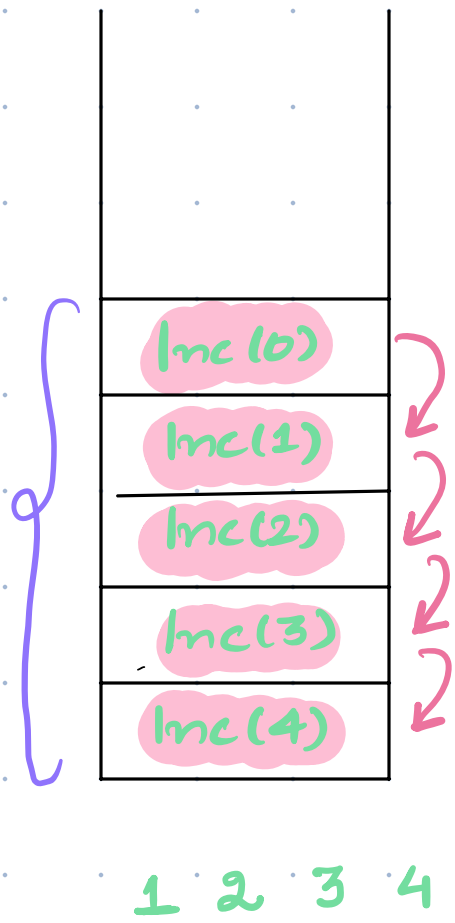
N = 5 o/p : 1 2 3 4 5

Assumption : Inc(N) → Print no from 1 → N.

Main logic : Inc(N-1)
print(N)

Base Case : N = 0 return

```
void Inc(int N){
    if(N==0) return;
    Inc(N-1);
    print(N);
}
```



N = 4

Inc(4)

(N+1) calls

T.C = (N+1)*c

T.C = O(N)

S.C = N+1

S.C. = O(N)

HW

Print numbers in decreasing order

Given N, print all numbers from N to 1 in decreasing order.

N = 5 O/p : 5 4 3 2 1

Order of printing & calling recursive func. will change in above code for dec. order.

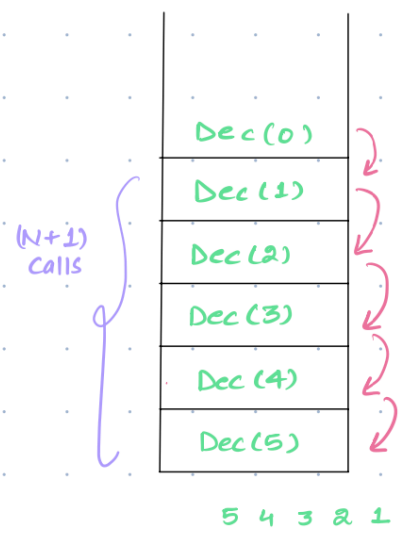
N = 5 O/p : 5 4 3 2 1

Assumption : Dec(N) : Prints no from N to 1 in dec order

Main logic : print(N)
Dec(N-1)

Base Case :

```
void printDec(int N) {
    if (N == 0) return;
    print(N);
    Dec(N-1);
}
```



T.C = (N+1) * 1
= O(N)

S.C = (N+1)
= O(N)