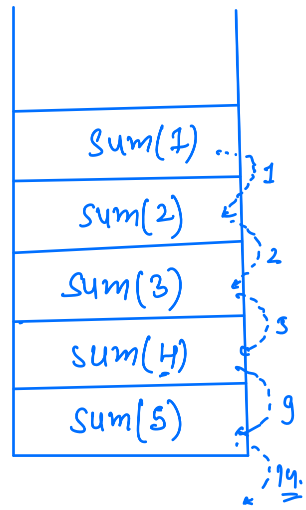
Space Complexity for Recursive Codes

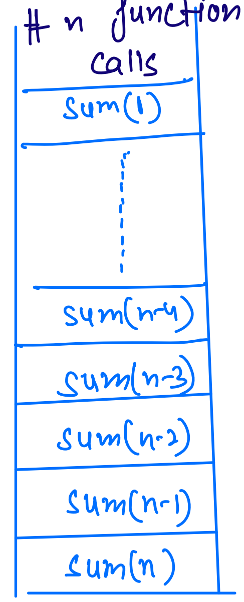
↳ Function calls are stored in stack, that is extra space that you are using.

↳ SC : Max^m size of stack

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

Eg : N=5.



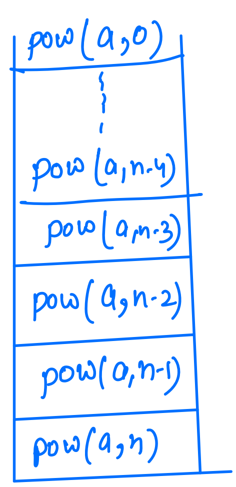


$\boxed{T.C \to O(N), \; S.C \to O(N)}$

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

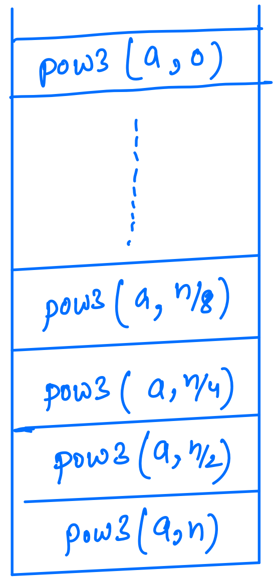
$\boxed{\begin{array}{l} T.C \to O(N) \\ S.C \to O(N) \end{array}}$

```
int pow(a,n){
    if (n==0) return 1
    return (pow(a,n-1)*a)
}
```



$\boxed{\begin{array}{l} T.C \to O(N) \\ S.C \to O(N) \end{array}}$

```
int pow3(a, n){
    if (n==0) return 1
    p = pow3(a, n/2)
    if (n%2 ==0) {return p*p}
    else {return p*p*a}
}
```



T.C → $O(\log_2 N)$

S.C → $O(\log_2 N)$

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

// Time taken to calculate fib(N) = f(N)

$$\left[f(N) = f(N-1) + f(N-2) + 1 \right]$$

$f(1) = 1$

$f(0) = 1$

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

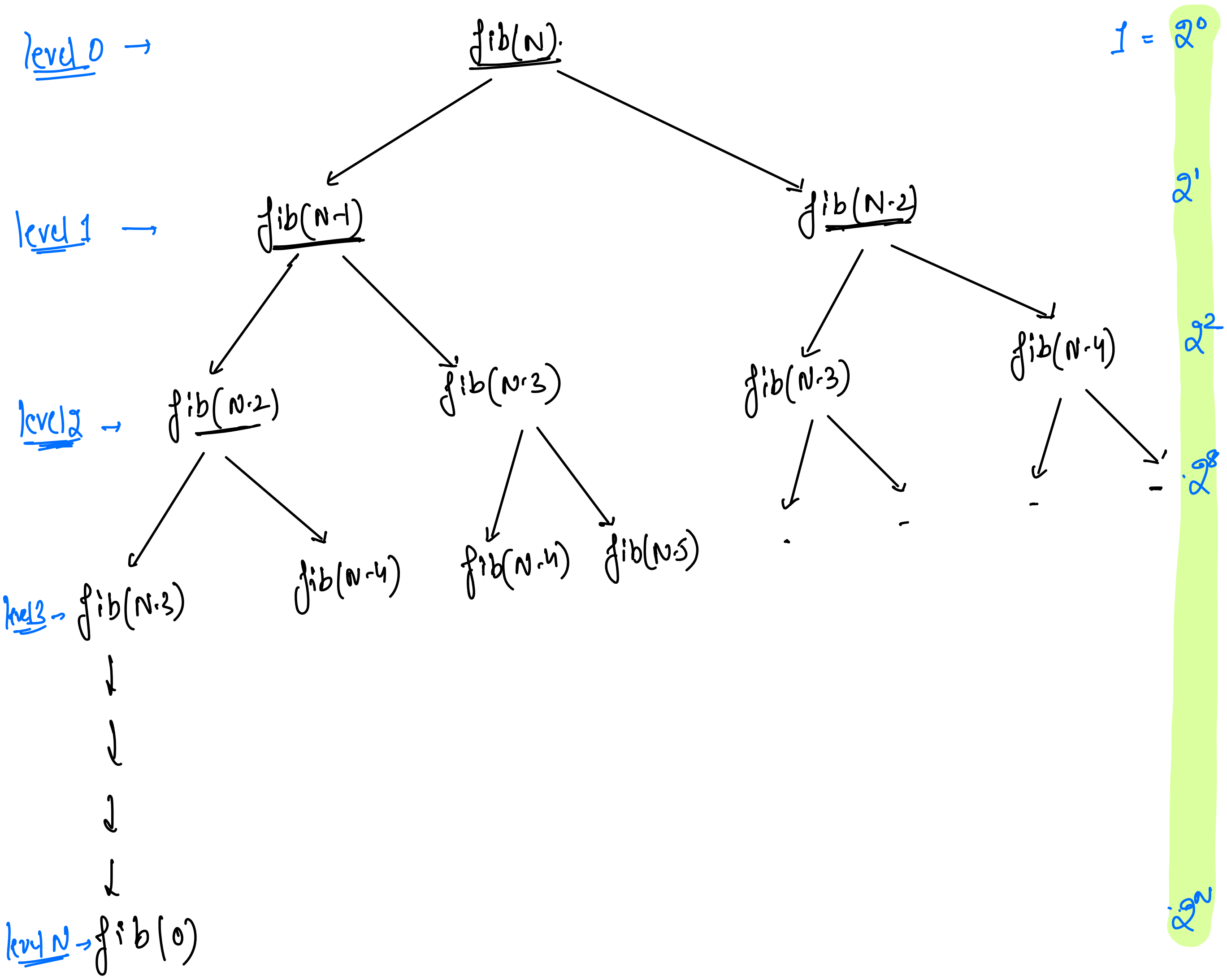
$$f(N-1) = f(N-2) + f(N-3) + 1$$

$$f(N-2) = f(N-3) + f(N-4) + 1$$

$$f(N) = f(N-2) + f(N-3) + 1 + f(N-3) + f(N-4) + 1 + 1$$

$$= f(N-2) + 2f(N-3) + f(N-4) + 3$$

Not a good approach.



Total function calls $= 2^0 + 2^1 + 2^2 + \text{———} \; 2^N$

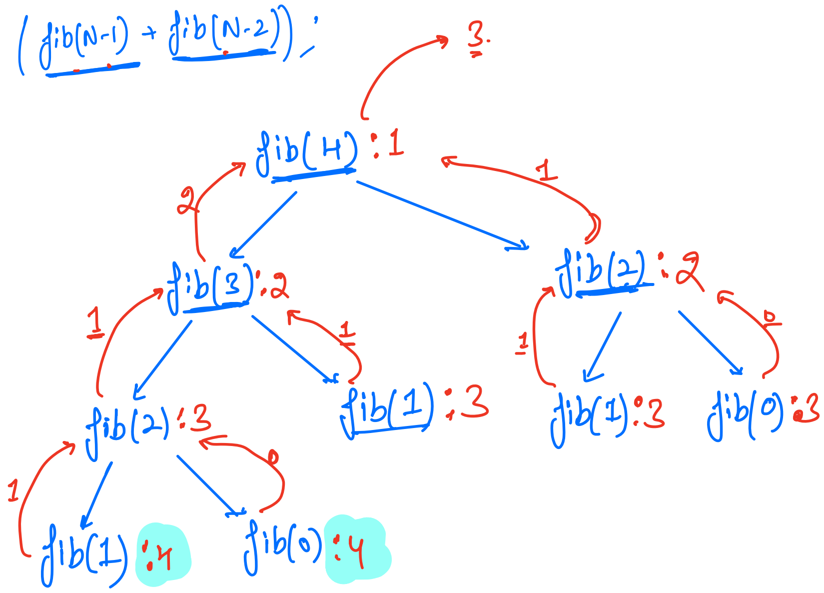
$\left[\underline{G.P},\ a = 1,\ r = 2,\ t = N+1\right]$

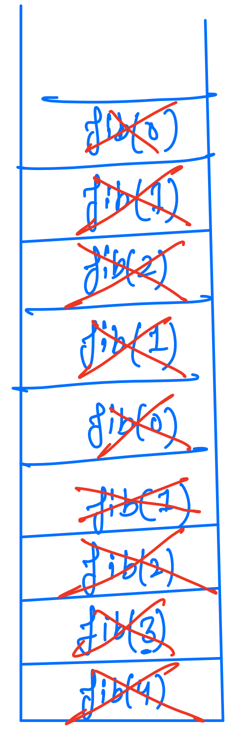
$$= \frac{a\left[r^t - 1\right]}{(r-1)} = \frac{\cancel{1}\left[2^{N+1} - 1\right]}{\cancel{[2-1]}}$$

$$= 2^{N+1} - 1 = 2 \cdot 2^N - 1 \Rightarrow \underline{O(2^N)}$$

$\boxed{T.C \rightarrow O(2^N)}$

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



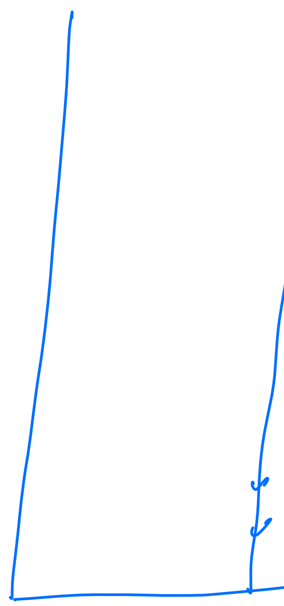


observation → Max stack size was 4

$\boxed{S.C \rightarrow O(N)}$

[At given point of time → there were N function calls in the stack.]

4.



Q.1 K^{th} Symbol

start from 0 & in every subsequent row, $\left\{ \begin{matrix} 0 \rightarrow 01 \\ 1 \rightarrow 10 \end{matrix} \right\}$

Given two no's A & B, return value present at B^{th} index of A^{th} row.

1 → [0]

×2

2 → [0 1]

×2

3 → [0 1 1 0]

×2

4 → [0 1 1 0 1 0 0 1]

×2

5 → [0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0]

eg A = 5, B = 6.

eg A = 5, B = 13.

// Return element present in A^{th} row at B^{th} idx.

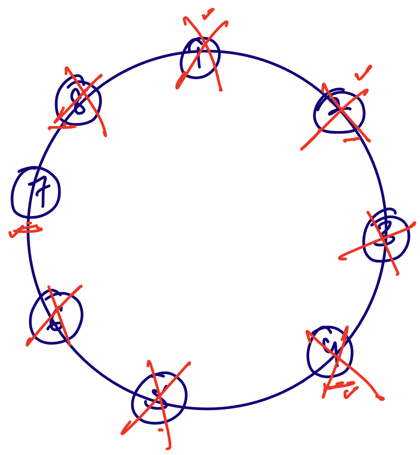
```
int solve ( A, B) {
    //✓
    if B is lying in first half of A^th row
        return { solve(A-1, B) }
    if B is lying in second half of A^th row.
        // consider the toggled value.
}
```

Josephus Problem

N = no. of persons in circle.

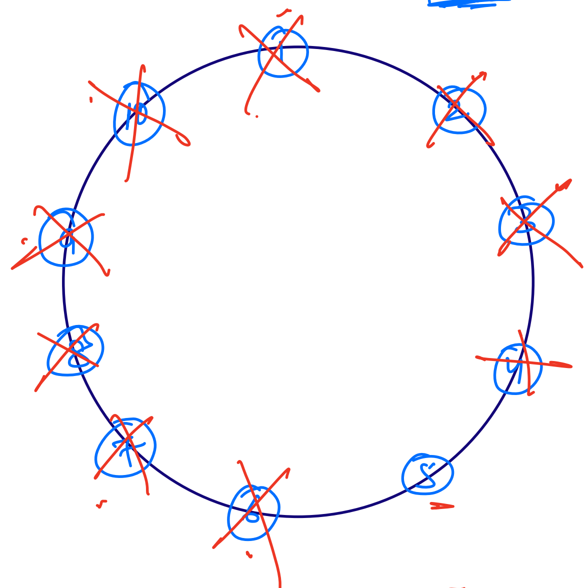
K → skip next (K-1) persons & K^{th} person will be Killed.

K=3.



[ans = 7]

K=4.



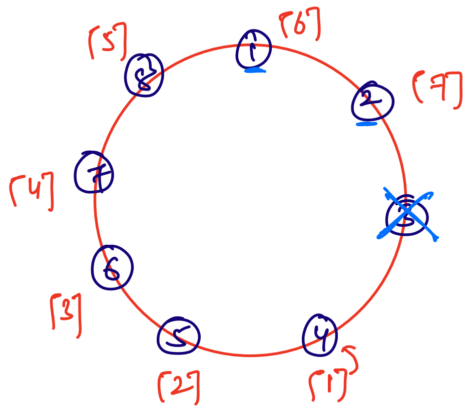
[ans = 5]

//Assm → find & return the posⁿ of last man standing if n person & killing every K^{th} person.

```
int josephus( n, K){

        josephus(n-1, K)

}
```



K=3.
N=8.

take care of mapping.

1	2	~~3~~	4	5	6	7	8	1	2
			↑+K	↑+K	↑+K	↑+K	↑+K	↑+K	↑
			1	2	3	4	5	6	7

→ Trace every problem using stack.

→ Arrays. [8 lectures]. → Arrays

→ Hashing.

→ Recursion

[B.T
modular arithmetic]

$f(N) \longrightarrow N^2$

$+$

$f(N-1) \longrightarrow N^2$

$+$

$\longrightarrow N^2$

$+$

$\longrightarrow N^2$

$+$

$f(1) \longrightarrow N^2$

$f(0) \longrightarrow$

S.C $\rightarrow$ $\underline{O(N^3)}$