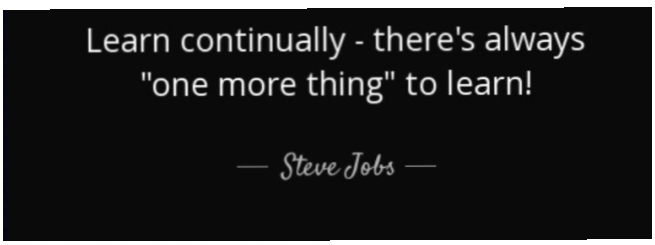
Quote →



Today's Content

- pow (a, n) {
  // three ways.
  }
- pow (a, n, p)
- T.C of recursive codes
- S.C of recursive codes.

Q) Given a, n . Find a^n using recursion.

Eg:

a	n	a^n
2	5	32
3	4	81

//Assumption → Calculate & return a^n

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

$a^n = \underbrace{a*a*a*a*---*a}_{n\text{-times}}$

$a^n = a^{n-1} * a$

$[a^n = pow(a, n-1) * a]$

```
int pow2 (a, n) {
    if (n == 0) { return 1 }

    if (n % 2 == 0) {
        return [ pow2(a, n/2) * pow2(a, n/2) ]
    }
    else {
        return [ pow2(a, n/2) * pow2(a, n/2) * a ]
    }
}
```

$a^{10} = a^9 * a$

$= a^5 * a^5$

$a^{14} = a^7 * a^7$

$a^{11} = a^5 * a^6$

$a^{11} = a^5 * a^5 * a$

$a^{13} = a^6 * a^6 * a$

① N is even.

$a^n \rightarrow a^{n/2} * a^{n/2}$

$pow(a, \frac{n}{2})$

② N is odd.

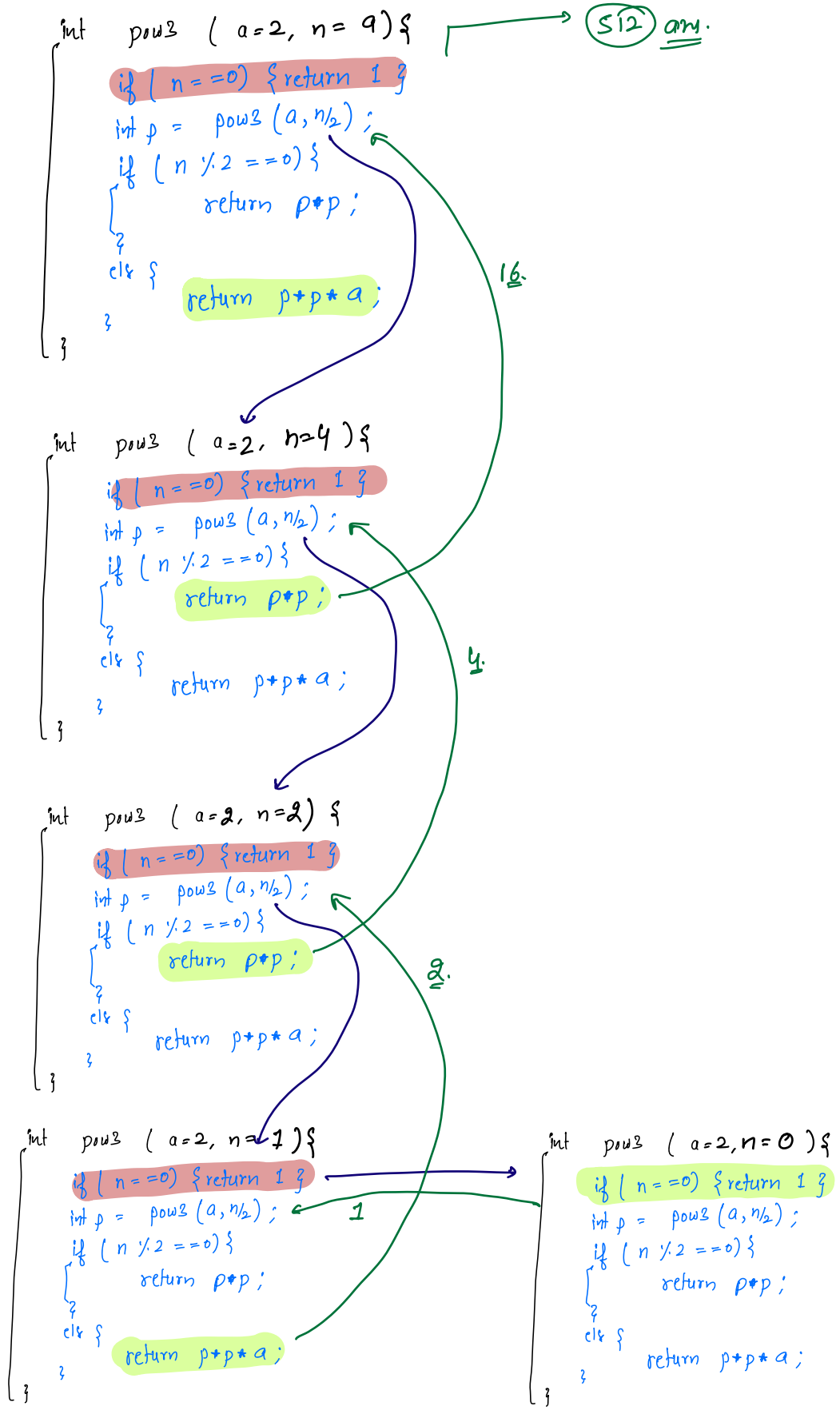
$a^n \rightarrow a^{n/2} * a^{n/2} * a$

$pow(a, n/2)$

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

// fast exponentiation

Tracing


int pow3 (a=2, n= 9){
if (n==0) {return 1 }
int p = pow3 (a, n/2) ;
if (n %2 ==0) {
return p*p ;
}
else {
return p*p* a ;
}
}
512 ans.
16.
int pow3 (a=2, n=4){
if (n==0) {return 1 }
int p = pow3 (a, n/2) ;
if (n %2 ==0) {
return p*p ;
}
else {
return p*p* a ;
}
}
4.
int pow3 (a=2, n=2) {
if (n==0) {return 1 }
int p = pow3 (a, n/2) ;
if (n %2 ==0) {
return p*p ;
}
else {
return p*p* a ;
}
}
2.
int pow3 (a=2, n=1){
if (n==0) {return 1 }
int p = pow3 (a, n/2) ;
if (n %2 ==0) {
return p*p ;
}
else {
return p*p* a ;
}
}
1
int pow3 (a=2, n=0){
if (n==0) {return 1 }
int p = pow3 (a, n/2) ;
if (n %2 ==0) {
return p*p ;
}
else {
return p*p* a ;
}
}

Q: Given a, n, m. Calculate $a^n \% m$.

Note → Take care of overflows.

Constraints → $1 <= a <= 10^9$, $1 <= n <= 10^9$, $2 <= m <= 10^9$

Ass^m: Calculate & return $a^n \% m$.

$a^n = a^{n/2} * a^{n/2}$

$a^n \% m = (a^{n/2} * a^{n/2}) \% m$

$= ((a^{n/2} \% m) * (a^{n/2} \% m)) \% m$

powmod(a, n/2, m)

```
powmod ( int a , int n, int m){
    if (n==0) { return 1 }

    long p = powmod(a, n/2, m)   //p = a^(n/2) % m
                                 At max p <= m = 10^9
    if (n%2 == 0){
        return (p * p) % m
    }
    else {
        return (p * p * a) % m
    }
}
```

$10^9 * 10^9 * 10^9$ ✗

```
    return (p%m * p%m * a%m) % m
```

$10^9 * 10^9 * 10^9$ ✗

```
    return ((p*p)%m * a) % m
```

$(10^9 * 10^9) \% m$; 10^9 ; 10^9 ✓

```
int powmod ( a , n , m ){
    if (n==0) { return 1 }
    long p = powmod ( a , n/2 , m)
    if ( n%2 == 0){
        return (p*p) % m ;
    }
    return ((p*p)%m * a) % m ;
}
```

T.C for Recursive Code Using Recursive Relation :-

①
```
int sum(N){ // Time taken to calculate sum(N) = f(N)
    if (N == 1){return 1}
    return (sum(N-1) + N)
}
```
f(N-1)

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

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

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

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

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

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

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

// After K steps.

$f(N) = f(N-K) + K$, $f(1) = 1$

$N - K = 1 \Rightarrow K = N-1.$

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

$= f(1) + N - 1$

$= 1 + N - 1 = N$

{Time complexity = O(N)}

②
```
int fact (N){
    if (N==1) {return 1}
    return ( fact (N-1) * N)
}
```

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

f(N-1)

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

[T.C → O(N)]

Task 1 → 10 min

Task 2 → 20 min

task 1 & task 2. = 10 + 20 = 30.

③ int pow1 (a, n){ // Time taken to calculate pow1(a,n) = f(n)

```
    if (N==0){return 1 }
    return pow1(a,n-1) * a
}
```

pow1(a,n-1) → f(n-1)

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

{ T.C → O(N) }

⑤
```
int pow3(a, n){          // Time taken to calculate pow3(a,n) = f(n)
    if(n==0){return 1}
    p = pow3(a, n/2)
    if(n%2==0){return p*p}
    else {return p*p*a}
}
```

$f(1)=1$, $f(0)=1$

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

$f(\frac{N}{2}) = f(N/4) + 1$

$f(N) = f(N/4) + 2$

$f(N/4) = f(N/8) + 1$

$f(N) = f(N/8) + 3$

$f(N/8) = f(N/16) + 1$

$f(N) = f(N/16) + 4$

T.C → $O(\log_2 N)$

After K steps →

$f(N) = f\left(\frac{N}{2^K}\right) + K$, $f(0)=1 \rightarrow f(1)=1$.

// $\frac{N}{2^K} = 0$ ✗ $\quad \frac{N}{2^K} = 1 \Rightarrow [N = 2^K] \Rightarrow [K = \log_2 N]$

$f(N) = f(N/N) + \log_2 N = f(1) + \log_2 N = 1 + \log_2 N \approx \log_2 N$

```
int powmod(a, n, m){          // Time taken to powmod(a,n,m) = f(n)
    if(n==0){return 1}
    long p = powmod(a, n/2, m)
    if(n%2==0){return (p*p)%m}
    else {return [(p*p)%m * a]%m}
}
```

$f(n) = f(n/2) + 1$

$f(n) = \log_2 n$

(4)

```
int pow2 (a, n) {                    // Time taken to calculate pow2(a, n) = f(n)
    if (n == 0) return 1
    if (n % 2 == 0) {
        return {pow2(a, n/2) * pow2(a, n/2)}
    }
    else {
        return {pow2(a, n/2) * pow2(a, n/2) * a}
    }
}
```

$f(n) = 2f(n/2) + 1$ ($1 = 2^1 - 1$)

$f(1) = 1$, $f(0) = 1$

$f(n/2) = 2f(n/4) + 1$

$f(n) = 2[2f(n/4) + 1] + 1$

$f(n) = 4f(n/4) + (2^2 - 1)$

$f(n/4) = 2f(n/8) + 1$

$f(n) = 4[2f(n/8) + 1] + 3$

$f(n) = 8f(n/8) + (2^3 - 1)$

After K steps:

$f(n) = 2^K f\left(\frac{n}{2^K}\right) + (2^K - 1)$, $f(1) = 1$

// $\frac{n}{2^K} = 0$ ✗ $\frac{n}{2^K} = 1 \Rightarrow [n = 2^K] \Rightarrow k = \log_2 n$.

$f(n) = n \cdot f(1) + n - 1$

$= 2n - 1$

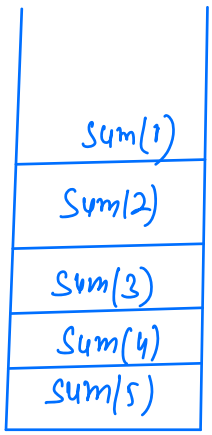
{T.C → O(N)}

Space Complexity for Recursive Codes ?

→ function calls are stored in stack. It is considered as extra space.

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

{T.C → O(N) , S.C → O(N)}




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

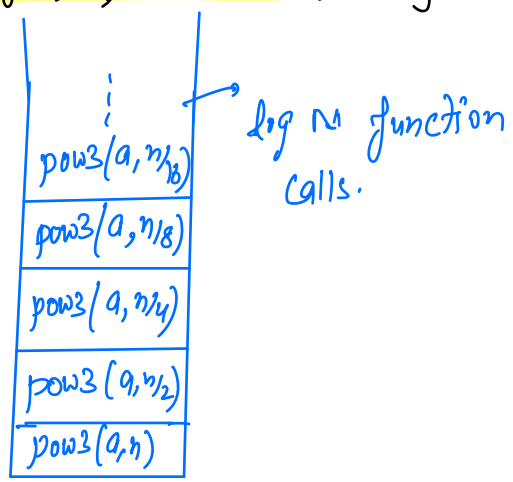
{T.C → O(N) , S.C → O(N)}

```
int pow1 (a, n) {
    if (n==0) { return 1 }
    return pow1 (a, n-1) * n
}
```

{T.C → O(N), S.C → O(N)}

```
int pow3(int a, int 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 N), S.C → O(log N)}




```
int fib(N){   // Time taken to calculate fib(N) = f(N)
    if(N<=1){return N}
    return {fib(N-1) + fib(N-2)}
}
```

f(N-1) f(N-2)

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

$f(0) = 1$

$f(1) = 1$

$f(N) = f(N-1) + f(N-2) + 1$ ✗ not a good approach

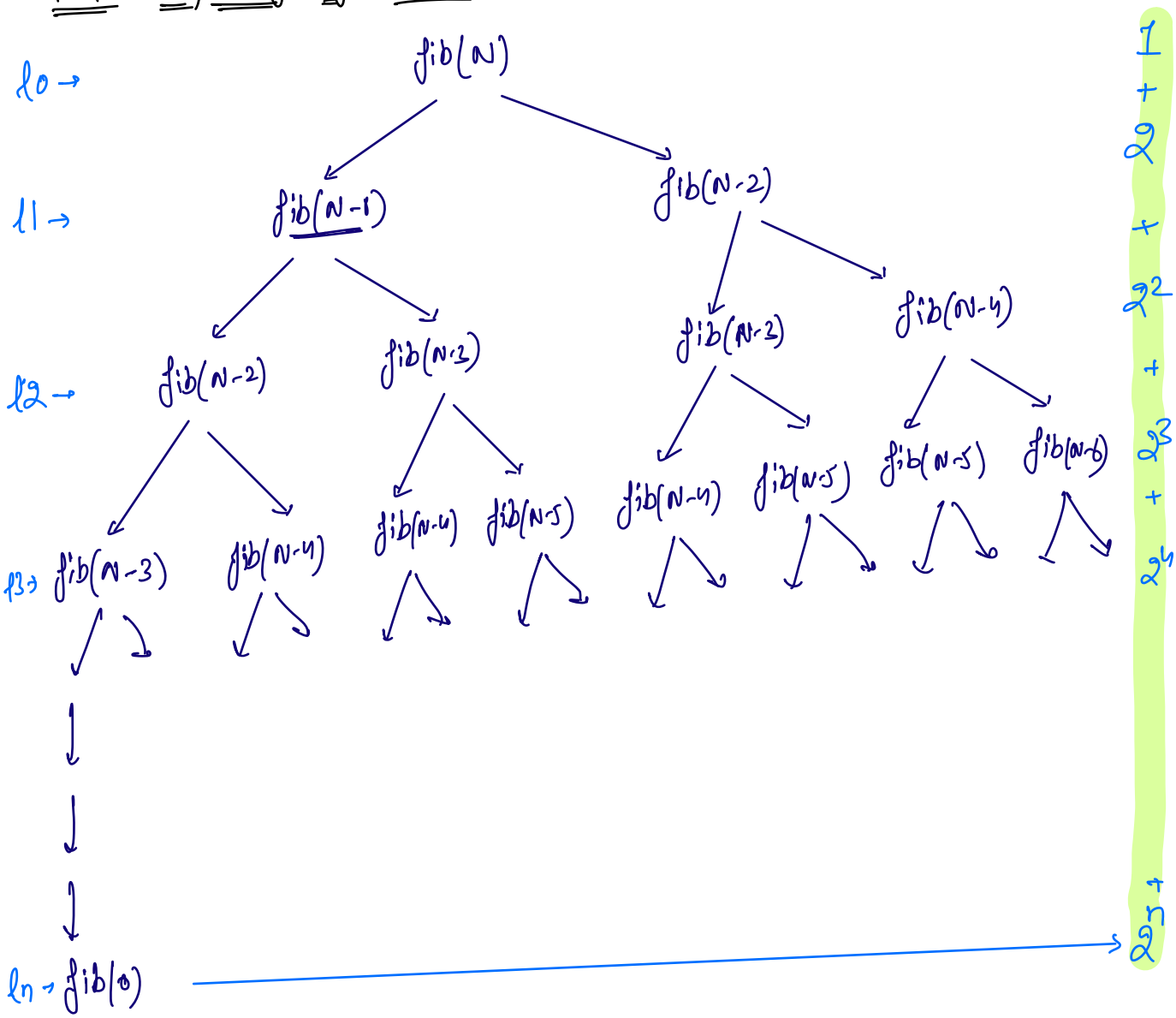
$f(N-1) = f(N-2) + f(N-3) + 1$

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

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

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

Time complexity of fib(N)



Total no. of function calls = $\underbrace{2^0 + 2^1 + 2^2 + 2^3 + \cdots 2^n}_{G.P}$

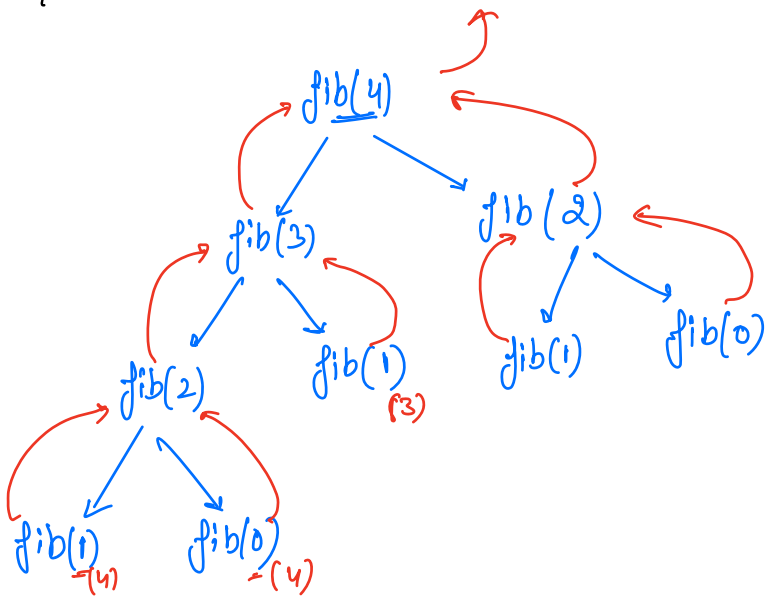
G.P, $a = 1$, $r = 2$, no. of terms $= n+1$

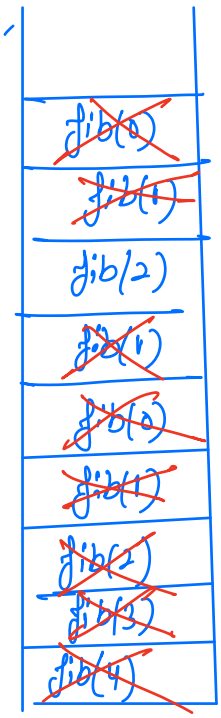
$$= \frac{1\left[2^{n+1} - 1\right]}{[2-1]} = \frac{2^{n+1} - 1}{1} \approx 2 \cdot 2^n - 1$$

{ T.C → $O(2^n)$
S.C → $O(N)$ }

Space Complixity of fib(N)

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





4

Max space used in stack at any given point of time → N.

that's why S.C → O(N).

{ check the base condition. }

↳ "there will be no error."

["stack overflow" → recursion is not the optimal solution]