Recursion 1

Demo Vido :- https://www.youtube.com/watch?v=-xMYvVr9fd4

RECURSION :- ① Function calling itself

② A problem is breaked down into smaller problem (Sub problem) & the solution is generated using the sub problems

Ex :- Sum (N) = 1 + 2 + 3 + 4 + ... + N

OR Sum (N) = Sum (N-1) + N

Eg :- Sum (10) = Sum (9) + 10

Question :- How to Write Recursive Code?

↳ We can Solve any Recursive Code using below Magic Steps :-

① Assumption :- Decide what the function does for the given problem

② Main logic :- Break the problem down into smaller sub problems to solve the assumption.

③ Base Case :- Identify the inputs for which we need to stop recursion

FUNCTION CALL TRACING

↳ To understand Recursion i.e how function calls work.

Code :-

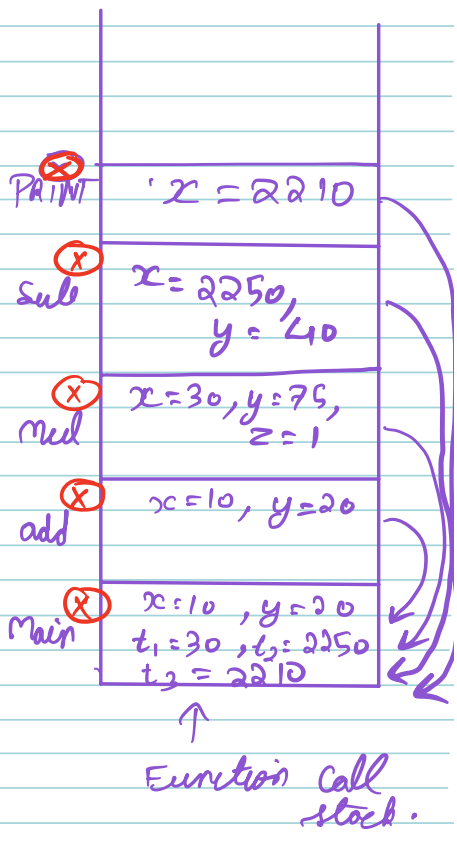
```
int add ( int x, int y) {
    return x + y;
}

int mul (int x , int y , int z) {
    return x * y * z;
}

int Sub ( int x, int y) {
    return x - y;
}

Void print (int x) {
    SOUT ( x );
}

int main () {
    int x = 10;
    int y = 20;
    t1 = add (x, y);
    t2 = mul ( t1, 75, 1);
    t3 = Sub ( t2, 40);
    ~~SOUT~~ PRINT (t3)
}
```



↑ Function Call stack.

OUTPUT

↳ 2210

Question :- Given a positive integer N, find factorial of N

Quiz 1 :- If N = 5, find the factorial of N

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$= 120$

IDEA :- fact (N) = fact (N-1) * N

Eg:- fact (5) = fact (4) * 5

↳ **Recursive Code**

↳ Lets use 3 Magical Steps.

PSEUDO CODE

```
function fact ( N ) {
Base Case ←  if ( N == 0 ) { return 1; }
Main logic {  int temp = fact (N-1);
              return temp * N
}
```

fact (N-1) → Assumption

$3 = 3 \times 2 \times 1 = 6$

DRY RUN

1st Way:- Using Code

N=3

```
function fact ( N : 3 ) {
    if ( N == 0 ) { return 1; }
    int temp = fact ( 2 );   // 2
    return temp * N          // 2 * 3 = 6
}

function fact ( N = 2 ) {
    if ( N == 0 ) { return 1; }
    int temp = fact ( 1 );   // 1
    return temp * N          // 1 * 2 = 2
}

function fact ( N = 1 ) {
    if ( N == 0 ) { return 1; }
    int temp = fact ( 0 );   // 1
    return temp * N          // 1
}

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

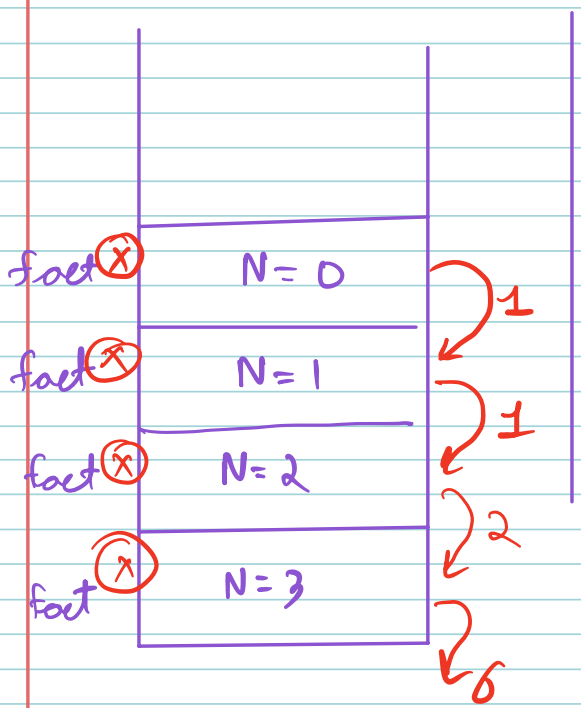
6

1

1

}

Way 2:- Using Call Stack.




```
function fact (N = 3) {
    if (N == 0) { return 1; }
    int temp = fact(N-1);
    return temp * N
}
```

Question :- Fibonacci Series

↳ Eg:- 0 1 1 2 3 5 8 13 21 ..... ∞

The fibonacci sequence is a series of Nos. in which each no is the sum of two preceding numbers.

Ex:-

I/P → 6th No	I/P → 8th No.
O/P → 8	O/P → 21

Quiz 2 :- If N=7, find the N^{th} No in Fibonacci Sequence.

↳ Ans = 13

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

Eg:- $fib(7) = fib(6) + fib(5)$

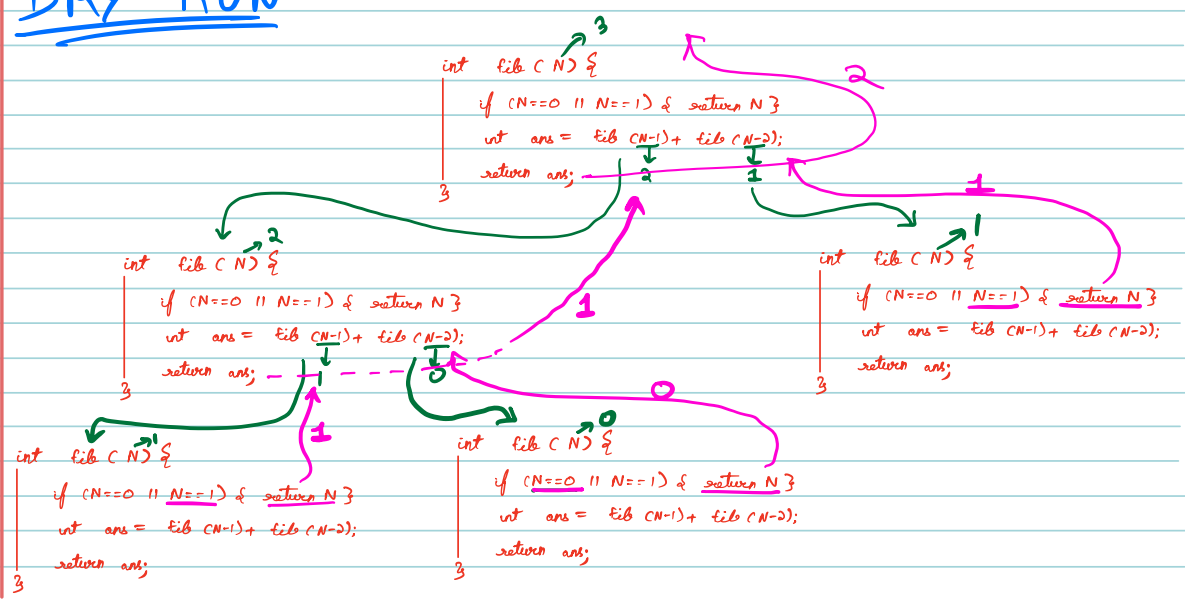
$= 5 + 8$

$= 13$

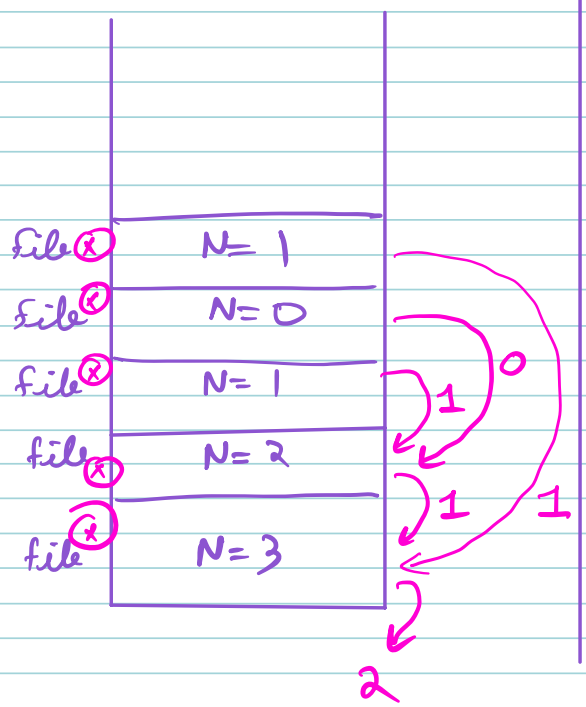
Recursive Code

```
int fib ( N ) {
    if ( N==0 || N==1 ) { return N; }        // Base Case
    int ans = fib(N-1) + fib(N-2);           // Main logic / Assumption
    return ans;
}
```

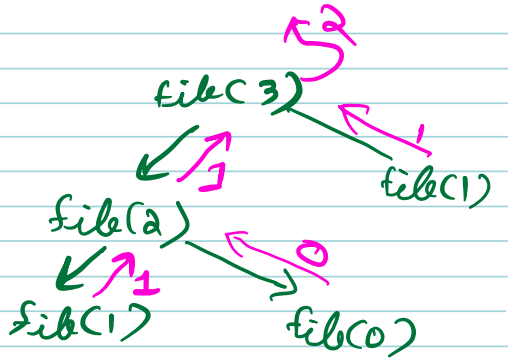
DRY RUN



↳ Using Call Stack




```
int fib(N) {                    // 3
    if (N==0 || N==1) { return N }
    int ans = fib(N-1) + fib(N-2);
    return ans;
}
```



Question :- Given two integers a & n. Find a^n using recursion.

Ex:- a = 2 , n = 3

$2^3 \rightarrow 8$ Ans

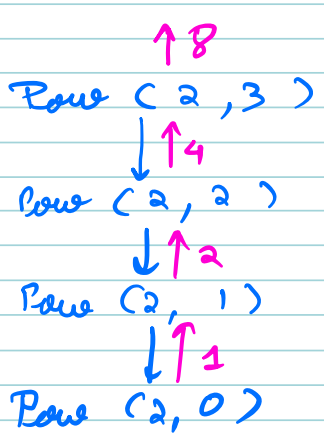
IDEA :- Pow (a, n) = Pow (a, n-1) * a

Eg:- $2^5 = 2^4 * 2$

PSEUDO CODE

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

DRY RUN



↳ OPTIMIZATION

Current Way :-

$2^4 = 2^3 * 2$

$2^{10} = 2^9 * 2$

Better Way :-

$2^4 = 2^2 * 2^2$

$2^{10} = 2^5 * 2^5$

IDEA

if n is even,

$Pow(a, N) = Pow(a, N/2) * Pow(a, N/2)$

if n is odd,

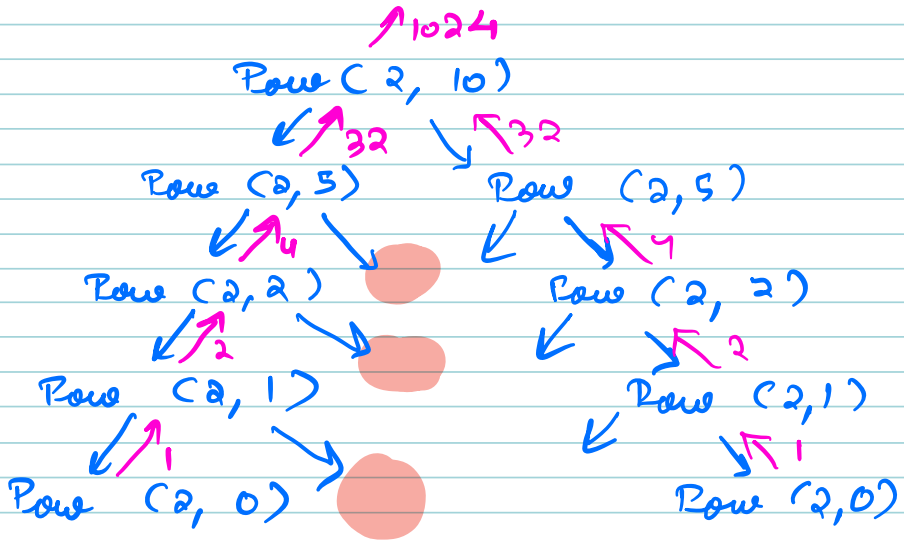
$Pow(N) = Pow(a, N/2) * Pow(a, N/2) * a$

↳ $2^{11} = 2^5 * 2^5 * 2$

PSEUDO CODE

```
func Pow ( a, n ) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        return Pow(a, n/2) * Pow(a, n/2);
    } else {
        return Pow(a, n/2) * Pow(a, n/2) * a;
    }
}
```

DRY RUN



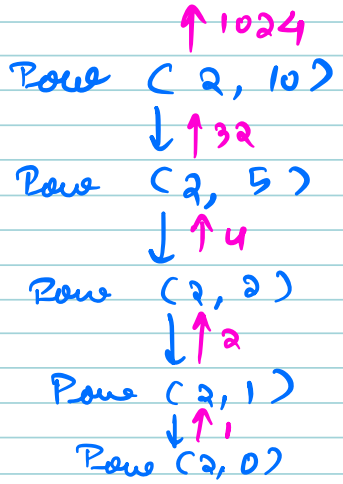
Stupidity :- Redundant call for the value which is already computed.

PSEUDO CODE [Fast Exponentiation / Fast Power]

```
func Pow(a, n) {
    if (n == 0) return 1;

    int temp = Pow(a, n/2);
    if (n % 2 == 0) {
        return temp * temp;
    } else {
        return temp * temp * a
    }
}
```

DRY RUN



TIME COMPLEXITY

(1) PSEUDO CODE

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

Recurrence Relation

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

↑ For some extra done on assumption

Base Case

$$T(0) = 1$$

↳ Solving Recurrence Relation

(Maths)

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

↳ $T(N-1) = T(N-2) + 1$

$$\Rightarrow T(N) = [T(N-2) + 1] + 1$$

$$\Rightarrow T(N) = T(N-2) + 2$$

$\Rightarrow$

↳ $T(N-2) = T(N-3) + 1$

$$\Rightarrow T(N) = T(N-3) + 3$$

Generic

$$\boxed{T(N) = T(N-k) + k}$$

Goal, $T(N-k) = T(0)$

↳ $N - k = 0$

↳ $\boxed{N = k}$

$$\Rightarrow T(N) = T(N-N) + N$$

$$= T(0) + N$$

$$\boxed{T(N) = 1 + N}$$ — (Base Case)

$$\boxed{T.C. = O(N)}$$

(2)

PSEUDO CODE

```
func Pow ( a, n ) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        return Pow(a, n/2) * Pow(a, n/2);
    } else {
        return Pow(a, n/2) * Pow(a, n/2) * a;
    }
}
```

Pow(a, n/2) → $T(\frac{N}{2})$

Recurrence Relation

$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + 1$$

$$\Rightarrow T(N) = 2T\left(\frac{N}{2}\right) + 1$$

$$T(0) = 1 \quad , \quad T(1) = 1$$

Maths :-

$$T(N) = 2T\left(\frac{N}{2}\right) + 1$$

$$T\left(\frac{N}{2}\right) = 2T\left(\frac{N}{4}\right) + 1$$

$$\Rightarrow T(N) = 4T\left(\frac{N}{4}\right) + 3$$

$$T\left(\frac{N}{4}\right) = 2T\left(\frac{N}{8}\right) + 1$$

$$=$$

$$\Rightarrow T(N) = 4\left[2T\left(\frac{N}{8}\right)+1\right] + 3$$

$$\Rightarrow T(N) = 8T\left(\frac{N}{8}\right) + 7$$

$$\hookrightarrow T\left(\frac{N}{8}\right) = 2T\left(\frac{N}{16}\right) + 1$$

$$\Rightarrow T(N) = 16\,T\left(\frac{N}{16}\right) + 15$$

Generic

$$T(N) = \left[2^k\,T\left(\frac{N}{2^k}\right)\right] + (2^k - 1)$$

AIM:- Remove T from right

(X) $\frac{N}{2^k} = 0 \Rightarrow N = 0$

$\Rightarrow \frac{N}{2^k} = 1 \Rightarrow N = 2^k \Rightarrow k = \log_2 N$

$$\Rightarrow T(N) = 2^{\log_2 N}\,T\left(\frac{N}{2^{\log_2 N}}\right) + \left(2^{\log_2 N} - 1\right)$$

$$\Rightarrow T(N) = N\,T\left(\frac{N}{N}\right) + (N-1) \qquad \left[a^{\log_a b} = b\right]$$

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

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

$$\boxed{\Rightarrow T(N) = 2N - 1}$$

T.C. $\Rightarrow$ $O(N)$

(3) PSEUDO CODE [FAST POWER]

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

Recurrence Relation

$$T(N) = T\left(\frac{N}{2}\right) + 1$$

Base Case $T(1) = 1$

Maths :- $T(N) = T\left(\frac{N}{2}\right) + 1$

$T\left(\frac{N}{2}\right) = T\left(\frac{N}{4}\right) + 1$

$\Rightarrow T(N) = T\left(\frac{N}{4}\right) + 2$

$T\left(\frac{N}{4}\right) = T\left(\frac{N}{8}\right) + 1$

$$\Rightarrow T(N) = T\left(\frac{N}{8}\right) + 3$$

$$\vdots$$

Generic

$$\Rightarrow T(N) = T\left(\frac{N}{2^K}\right) + K$$

$$\hookrightarrow \frac{N}{2^K} = 1$$

$$\Rightarrow N = 2^K \quad \Rightarrow \quad K = \log_2 N$$

$$\Rightarrow T(N) = T\left(\frac{N}{2^{\log_2 N}}\right) + \log_2 N$$

$$= T\left(\frac{N}{N}\right) + \log_2 N$$

$$= T(1) + \log_2 N$$

$$\Rightarrow \boxed{T(N) = 1 + \log_2 N}$$

$$\boxed{TC \longrightarrow O(\log_2 N)}$$

④ Fibonacci

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

Recurrence Relation

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

↳ Directly this will not simplify to a generic relation

$$\therefore T(N-1) > T(N-2)$$

So we can take $T(N-1)$ instead of $T(N-2)$ because it's something extra not less.

$$\Rightarrow \boxed{T(N) = 2T(N-1) + 1}$$

↳ H.W.

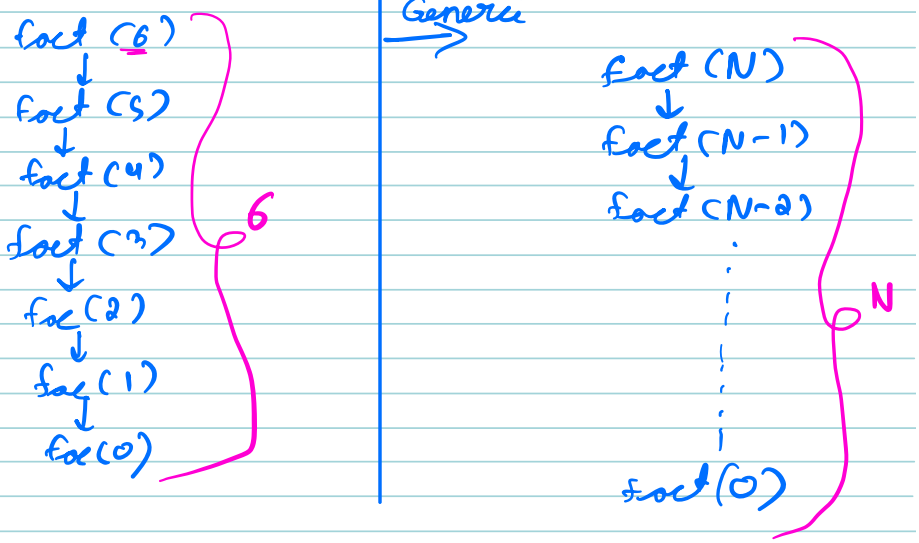
$$\boxed{T.C \rightarrow O(2^n)}$$

Another Way Of Calculating T.C.

Time complexity :- Time taken by one function call * No. of function calls

Quiz 3 :- How many recursive calls in the factorial (6)?

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



TC → 1 * N ← Function call

↑ Time taken by function

TC → O(N)

Quiz 4 How many recursive call in the FAST pow(2,5)

```
Func Pow2 ( a, n ) {
    if (n==0) { return 1 }
    temp = Pow2 ( a, n/2 );
    if ( n%2 == 0){
        return temp * temp;
    } else {
        return temp * temp * a;
    }
}
```

Pow (2,5)
↓
Pow (2,2)
↓
Pow (2,1)
↓
Pow (2,0)

Generic

Pow (a, N)
↓
Pow (a, N/2)
↓
Pow (a, N/4)
⋮
Pow (a, 0)

$\log_2 N$

TC → $1 * \log_2 N$

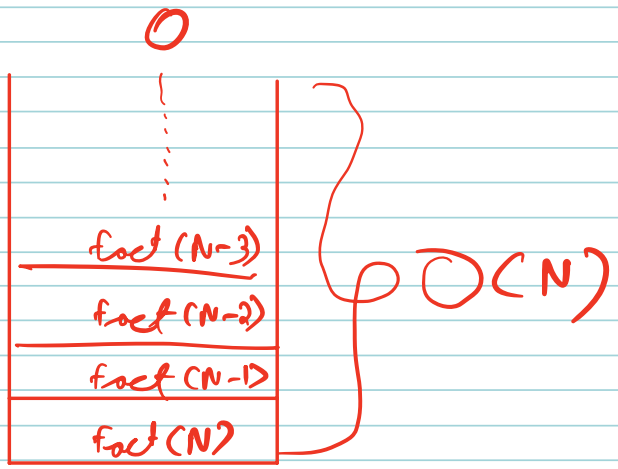
$\log_2 N$

T.C. → $O(\log_2 N)$

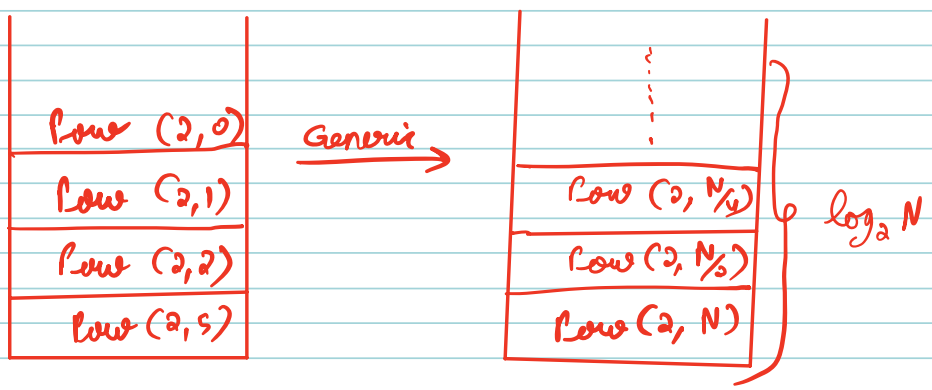
SPACE COMPLEXITY of Recursive call

↳ maximum function in call stack at one time.

① For Factorial



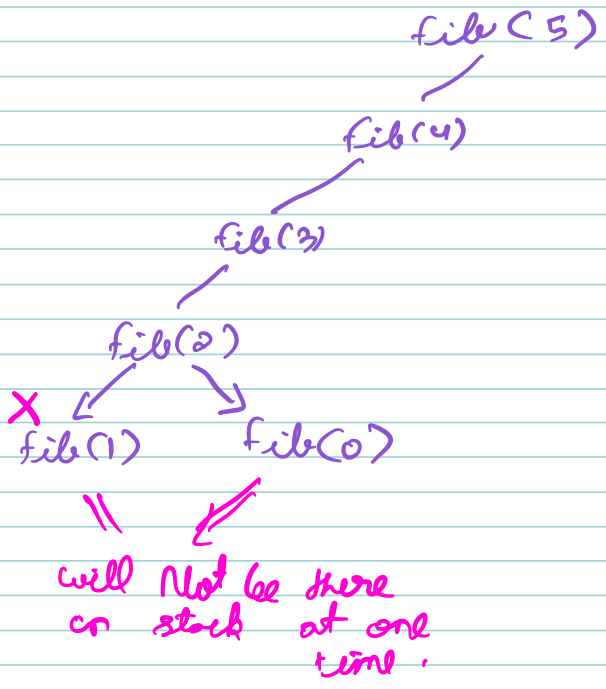
② Fast Power



SC → O ($\log_2 N$)

(3) Fibonacci

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



will Not be there on stack at one time.

SC → O(N)