Recursion :- function calling itself.

↳ technique of solving a problem using $\underbrace{\text{smaller instance}}_{\text{sub problem}}$ of same problem.

Steps :-

1) Assumption :- Assume that our recursive code will work.

2) Main logic :- Solve the problem using subproblem.

3) Base | Exit condition

1) fibonacci Series

N:	0	1	2	3	4	5	6	7	8	.....
	0	1	1	2	3	5	8	13	21	.....

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

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

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

Ⓘ Assumption :-

fib(N) returns N^{th} fibonacci number.

Ⓘ Main logic

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

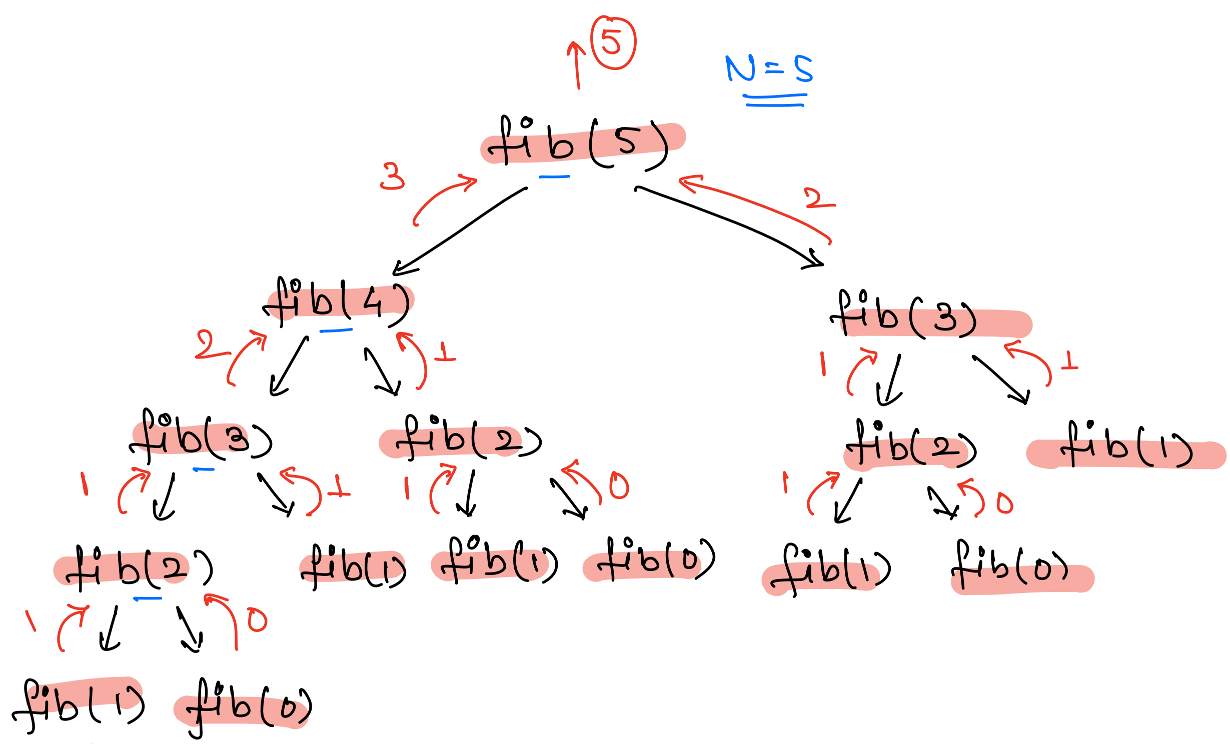
Ⓘ Base | Exit cond^n

× $fib(1) = fib(0) + \cancel{fib(-1)}$

✓ $fib(2) = fib(1) + fib(0)$

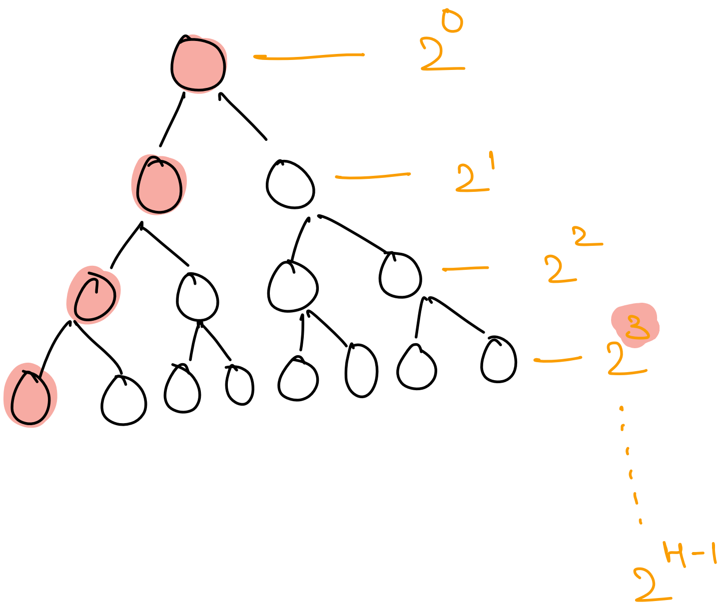
$fib(0) = \cancel{fib(-1)} + \cancel{fib(-2)}$

Dry Run



T.C Analysis

TC : (No. of funcⁿ calls) × (Work done in each funcⁿ call)



$$\# \text{ of fun}^n \text{ calls} = \underbrace{2^0 + 2^1 + 2^2 + \cdots\cdots 2^{H-1}}_{G.P}$$

$a = 2^0 = 1$

$r = 2$

$\#$ of terms $= x = H$

$$\# \text{ of fun}^n \text{ calls} = \frac{2^0(1-2^H)}{(1-2)} = \frac{1(1-2^H)}{(-1)}$$

$$\# \text{ of fun}^n \text{ calls} = 2^H - 1$$

Height of tree = $H = N$

# of funⁿ calls = $2^N - 1$

TC : (# of funⁿ calls) × (1)

: $2^N - 1$

$\boxed{TC : O(2^N)}$

Recurrence Relation

$T(N)$: TC funⁿ to fib N^{th} fibonacci no.

$T(N-1)$: TC funⁿ to fib $(N-1)^{st}$ fibonacci no.

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

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

$$T(N) = T(N-1) + T(N-1) + 1$$ } Upper Bound.

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

$$\begin{cases} T(0) = 1 \\ T(1) = 1 \end{cases}$$

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

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

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

$$= 4[2T(N-3) + 1] + 3$$

$$T(N) = 8T(N-3) + 7$$

$$= 8[2T(N-4) + 1] + 7$$

$$T(N) = 16T(N-4) + 15 \quad \text{4 steps}$$

After k steps :-

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

$$N-k = 0$$

$$\boxed{k = N}$$

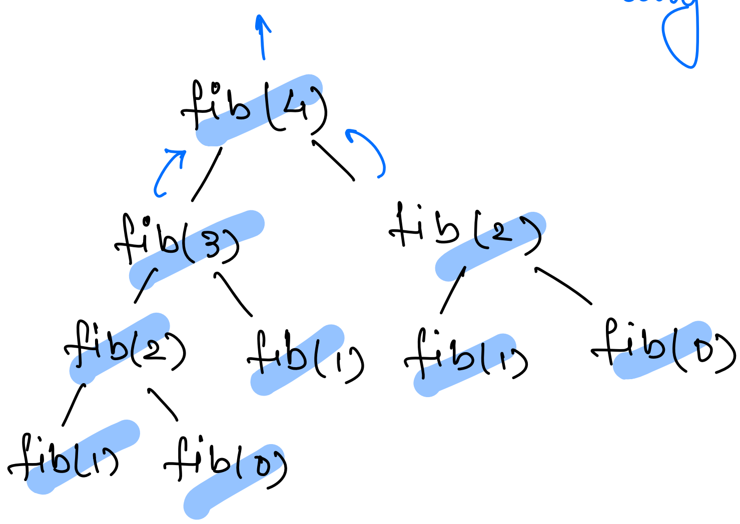
$$T(N) = 2^N \cdot \cancel{T(0)}^{\,1} + 2^N - 1$$

$$= 2 \cdot 2^N - 1$$

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

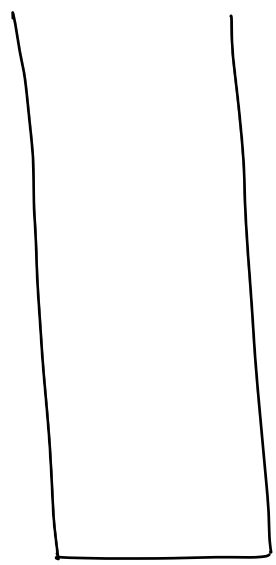
$$\boxed{TC : O(2^N)}$$

Space Complexity :- Maximum amount of memory used by our recursive funⁿ at any point of time.



SC : 4

$\boxed{SC : O(N)}$



Call Stack

Power function

$pow(a, n) \Rightarrow a^n$

$$a^N = \underbrace{a \times a \times a \times \cdots\cdots \, a \times a}_{N \text{ times.}}$$

$$\boxed{a^N = a^{N-1} \times a}$$

$x^a \cdot x^b = x^{a+b}$

$pow(a, n) = pow(a, n-1) \times a$

$\boxed{a^0 = 1}$

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

TC : $O(N)$

a^5
↳ $a^4 \times a$
↳ $a^3 \times a$
↳ $a^2 \times a$
↳ $a^1 \times a$
↳ $a^0 \times a$

a^N : N multiplications

$$\left.\begin{array}{c} 5 \\ \downarrow \\ 4 \\ \downarrow \\ 3 \\ \downarrow \\ 2 \\ \downarrow \\ 1 \\ \downarrow \\ 0 \end{array}\right\} H = N$$

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

SC : $O(N)$

$\Rightarrow$

$$a^8 = a^4 \times a^4$$

$$a^{10} = a^5 \times a^5$$

$$a^{11} = a \times a^5 \times a^5$$

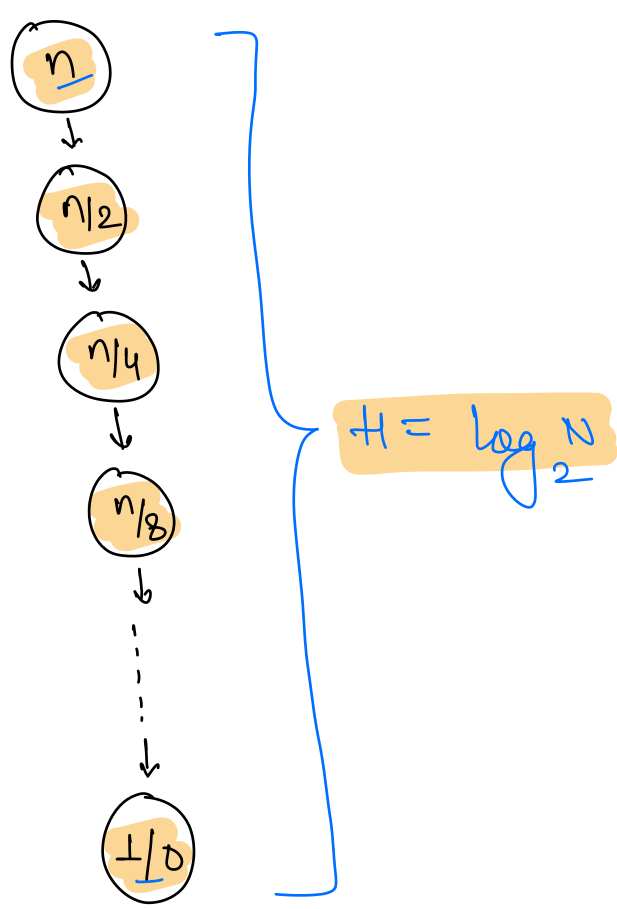
$$a^9 = a \times a^4 \times a^4$$

$$a^N = \begin{cases} a^{N/2} \times a^{N/2} & \text{if } N \text{ is even} \\ a \times a^{N/2} \times a^{N/2} & \text{if } N \text{ is odd.} \end{cases}$$

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

    hp = pow(a, n/2);
    ans = hp x hp;
    if(N % 2 == 0){
        return ans;
    }
    return a x ans;
}
```

Recursion Tree :-



$$\boxed{\begin{aligned} TC &: O(\log_2 N) \\ SC &: O(\log_2 N) \end{aligned}}$$

Recurrence Relation

HW $\boxed{T(N) = T(N/2) + 1} \Rightarrow TC : O(\log_2 N)$

Q. Gray Code

Binary sequence in which consecutive no's differs by single bit. Given N, generate gray code sequence of N bits.

N = 0 : { }

N = 1 :
0
1

N = 2 :
0 0
0 1
1 1
1 0

$\Rightarrow$ { "00", "01", "11", "10" }

$\boxed{N \Rightarrow 2^N \text{ gray code no's}}$

N = 3
0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

reverse order

$N = 3$

$= 2^3$

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

$N = 4$

$2^4 = 16$

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

```
List<String> grayCode(N) {
    if(N == 0) return {};
    if(N == 1) return {'0', '1'};
    List<String> codes = grayCode(N-1);
    int x = codes.size(); // x = 2^(N-1)
    List<String> ans;
    for(i = 0; i < x; i++){
        ans.add('0' + codes[i]);
    }
    for(i = x-1; i >= 0; i--) {
        ans.add('1' + codes[i]);
    }
    return ans;
}
```

$O(x)$, $x = 2^{N-1}$ (first loop)

$O(x)$, $x = 2^{N-1}$ (second loop)

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

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

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

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

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

$$= T(N-3) + 2^{N-2} + 2^{N-1} + 2^N$$

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

<u>K steps</u> :-

$$T(N) = T(N-K) + 2^{N-K+1} + \cdots\cdots 2^N$$

$$N-K = 0 \Rightarrow K = N.$$

$$T(N) = T(\cancel{0}) + 2^1 + 2^2 + 2^3 + \cdots\cdots 2^N$$

1

$$T(N) = \underbrace{2^1 + 2^2 + 2^3 + \cdots\cdots + 2^N}_{GP}$$

$a = 2^1$

$r = 2$

\# of terms $= N$

$$T(N) = \frac{2^1(1-2^N)}{(1-2)} = 2(2^N-1)$$

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

$$\boxed{TC : O(2^N)}$$

$$\# \text{ of iterations} = 2^N + 2^{N-1} + 2^{N-2} + \cdots + 2^0$$

$$\boxed{\begin{array}{l} TC : O(2^N) \\ SC : O(2^N) \end{array}}$$