Q. Implement power function.

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

$pow(2, 6) = 2^6 = 64$

$pow(3, 3) = 27$

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

$= a \times a^{n-1}$

$a^5 = a \times a^4$

$\hookrightarrow a \times a^3$

$\hookrightarrow a \times a^2$

$\hookrightarrow a \times a^1$

$\hookrightarrow a \times a^0$

No. of multiplications = 5

$a^N \Rightarrow N$

TC: $O(N)$

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

$$a^{10} = a \times a^{9}$$

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

$$a^{14} = a^{7} \times a^{7}$$

$$a^{15} = a \times a^{7} \times a^{7}$$

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

$$a^{64} = a^{32} \times a^{32}$$

$$\hookrightarrow a^{16} \times a^{16}$$

$$\hookrightarrow a^{8} \times a^{8}$$

$$\hookrightarrow a^{4} \times a^{4}$$

$$\hookrightarrow a^{2} \times a^{2}$$

$$\hookrightarrow a^{1} \times a^{1}$$

$$a^{N} = \begin{cases} a^{N/2} * a^{N/2} & \text{if } N\%2 == 0 \\ a * a^{N/2} * a^{N/2} & \text{else} \end{cases}$$

```
int pow(a, n) {
    if(n == 0)
        return 1;
    int halfPow = pow(a, n/2); // a^(N/2)
    int halfAns = halfPow * halfPow;
    if(n % 2 == 0) {
        return halfAns;
    }
    else {
        return a * halfAns;
    }
}
```

Ex

pow(2,7) { → 128
// N = 7
hp = pow(2,3)
ha = 64 ← 8
rt a * ha
}

pow(2,3) {
// N = 3
hp = pow(2,1)
ha = 4 ← 2
rt a * ha
}

pow(2,1) {
// N = 1
hp = pow(2,0) = 1
ha = 1 ← 1
rt a * ha
}

pow(2,0) {
// N = 0
base case
}

Note Explore library pow(a,n) fun^

pow(a,n) = $a^N$ ⇒ Overflow for larger values of N.

Google

pow(a,n,d) ⇒ $a^N \% d$

```
int pow(a, n, d) {
    if(n == 0)
        return 1;
    int halfPow = pow(a, n/2, d); // a^(N/2)
    int halfAns = (halfPow % d * halfPow % d) % d;
                  (long)          (long)
    if (n % 2 == 0) {
        return halfAns;
    }
    else {
        return (a % d * halfAns % d) % d;
    }
}
```

```
int halfAns = (halfPow % d * halfPow % d) % d;
               10^9          10^9
```

> int range.

Ex: $d = 10^9 + 1$

halfPow % d ⇒ [0, d-1]

⇒ $[0, 10^9]$

$N \Rightarrow \frac{N}{2} \Rightarrow \frac{N}{4} \Rightarrow \frac{N}{8} \Rightarrow \cdots\cdots \Rightarrow 1|0$

$O(\log_2 N)$

# Time Complexity Analysis

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

↳ $(N-1) + Sum(N-2)$

↳ $(N-2) + Sum(N-3)$

↳ ...

$N \rightarrow N-1 \rightarrow N-2 \rightarrow N-3 \rightarrow \cdots\cdots 1$

N steps

TC: $O(N)$

⇒ Recurrence relation

$T(N)$ : TC of $Sum(N)$

$T(N-1)$ : TC of $Sum(N-1)$

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

$T(N) \quad 1 + T(N-1)$

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

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

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

$= 2 + T(N-2)$

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

$= 3 + T(N-3)$

$= 4 + T(N-4)$

⋮

After K steps $T(N) = K + T(N-K)$

$K = N$ $T(N) = N + T(0)$

$T(0) = 1$

$T(N) = N + 1$

TC: $O(N)$

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

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

$T(N) = T(N-1) + \underbrace{T(N-2)}_{< T(N-1)} + 1$

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

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

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

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

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

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

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

$= 16T(N-4) + 15$

$\vdots$

After K steps. $= 2^K T(N-K) + (2^K - 1)$

N steps $\qquad K = N.$

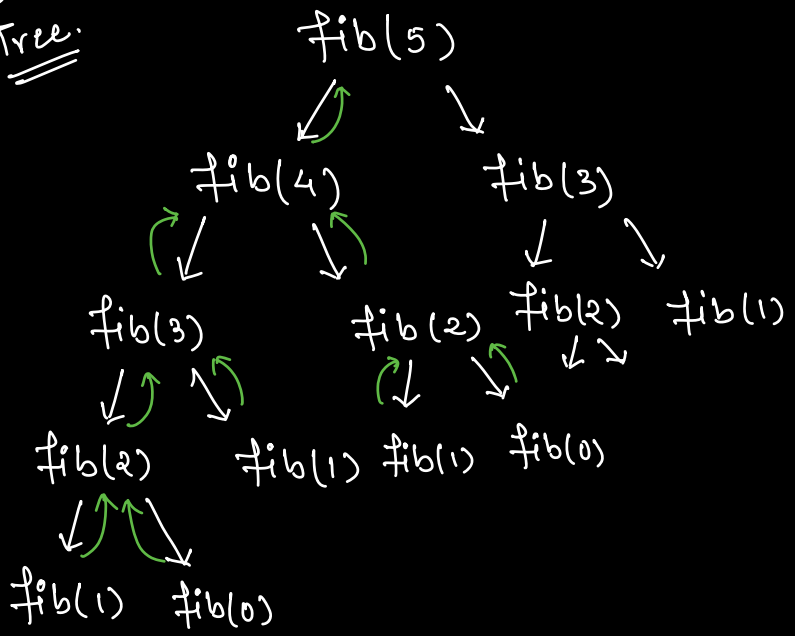
$T(N) = 2^N T(0) + 2^N - 1$

$T(0) \to 1$

$= 2(2^N) - 1$

TC : $O(2^N)$

Recursion Tree:




Quiz

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

↳ Power fun
↳ Binary search.

TC : $O(\log N)$

Quiz $T(N) = 2T(N/2) + 1$

↳ $O(N)$

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

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

$$= 4T(N/4) + 3$$

$$= 4[2T(N/8) + 1] + 3$$

$$= 8T(N/8) + 7$$

$$= 16T(N/16) + 15$$

K steps

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

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

$$N = 2^K$$

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

$$\log_a a^x = x$$

$$K = \log N$$

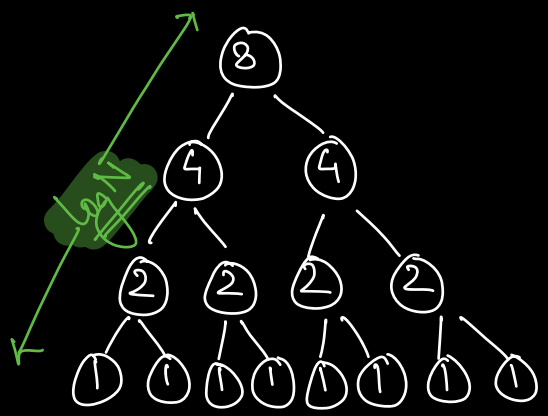
$$T(N) = 2^{\log N} \cdot T(1) + 2^{\log N} - 1$$

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

$T(1) = 1$

$$= 2N - 1$$

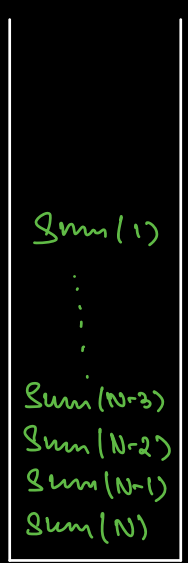
**TC: O(N)**




HW $T(N) = 2T(N/2) + O(N)$

## Space Complexity

```
Sum(N)
  └→ Sum(N-1)
       └→ Sum(N-2)
            └→ Sum(N-3)
                 └→ Sum(N-4)
                      └→ ....
                           Sum(0)
```

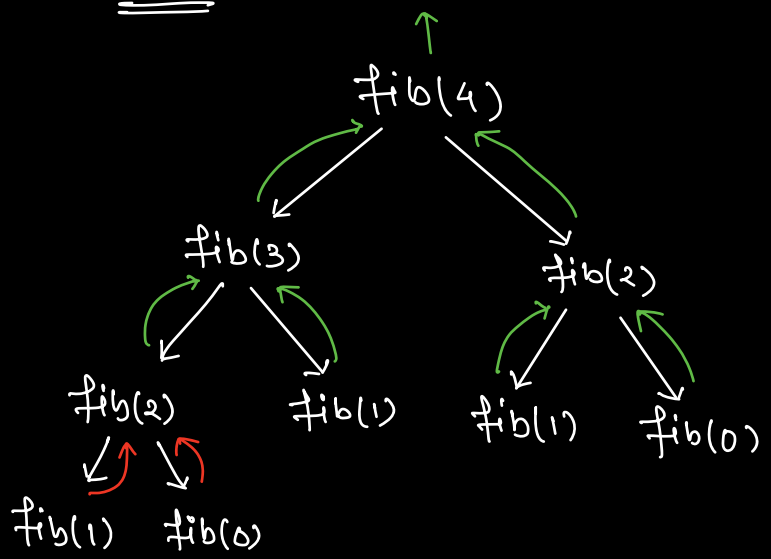


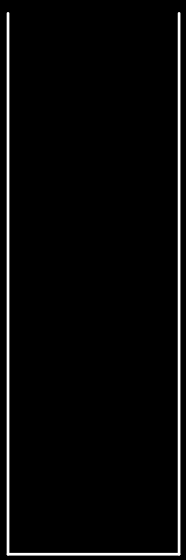

Function stack.

N fun calls are present in the stack.

SC: $O(N)$

* Fibonacci



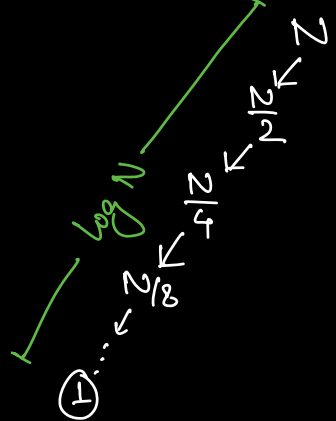
4

SC: Max. no. of fun calls stored in the stack at any point of time.

fib(5) = fib(4) + fib(3)

SC: O(N)

Ex SC of pow() fun




SC: O(log N)

---