→ More problems on Recursion.

→ TC / SC analysis for recursion.

Q. Given a number, Write the recursive function to find the sum of digits of number.

1234 ⇒ 10

```
int sumDigit(int N){
    if(N<10){
        return N;
    }
    return N%10 + sumDigit(N/10);
}
```

* `if(N==0) return 0;`

(i) sumDigit(N) ✓

(ii) Main logic

N = 1234

1+2+3+4

N%10 → last digit

N/10 → remove last digit

(iii) Base Condition

0 → 0

N<10 ⇒ 'N'

N is a positive ?

Q. Implement "pow" function.

pow(a, n) ⇒ a^n

a = 3
n = 3

$a^n = 3^3 = 3 \times 3 \times 3 = 27$.

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

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

$a^0 = 1$

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

a^5

↳ $a \times a^4$

↳ $a \times a^3$

↳ $a \times a^2$

↳ $a \times a^1$

↳ $a \times a^0$ ↓ Base condition.

5 multiplications.

$a^N \Rightarrow$ N multiplications.

↳ O(N) : TC

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

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

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

$= a^5 \times a^5$

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

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

a^N {

- $a^{N/2} \times a^{N/2}$ if $N\%2 == 0$
- $a^{N/2} \times a^{N/2} \times a$ if $N\%2 == 1$.

$a^{10} \Rightarrow a \times a^9 \Rightarrow$ 10 multiplication.

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

↳ $a^5 = a^2 \times a^2 \times a$

↳ $a \times a$.

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

↓

$\log N$

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

$a^{64} \rightarrow a^{32} * a^{32}$

↳ $a^{16} \times a^{16}$

↳ $a^8 \times a^8$

↳ $a^4 \times a^4$

↳ $a^2 \times a^2$

↳ $a \times a$

6 multiplications.

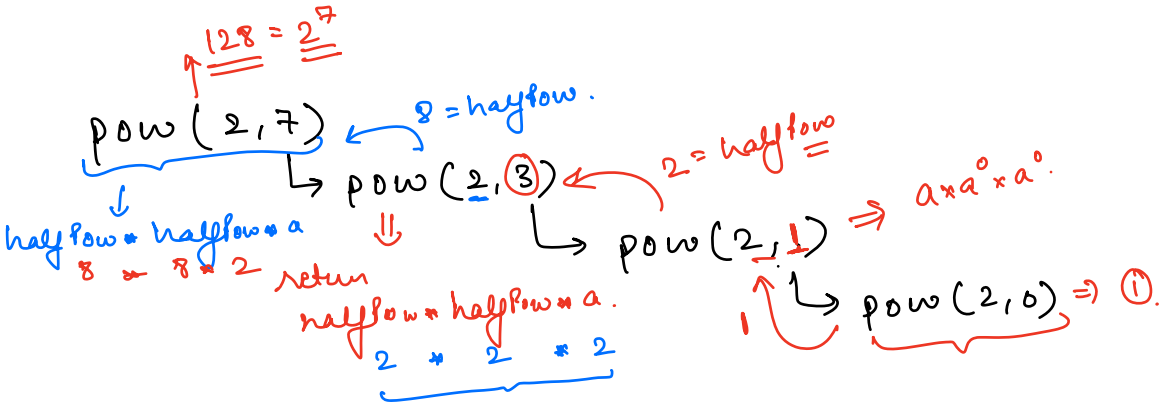
How many multiplications ?

$\Rightarrow \log_2 N$.

TC : $O(\log N)$

HW: Analyse the pow(a,n) function of your respective language.

→



pow(2,1) ⇒ 2 * pow(2,0) * pow(2,0).
1 1

⇒ 2

pow(a, n, d) ⇒ $a^n \%. d$ ≈ 10^9

≤ d-1

$a \%. d \Rightarrow [0, d-1]$.

$(a * b) \%. M = (a \%. M * b \%. M) \%. M$

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

```
int halfAns = (halfPow % d * halfPow % d) % d;
```

halfPow % d ≤ d-1 ≃ $10^9 - 1$

halfPow % d ≤ d-1 ≃ $10^9 - 1$

$(10^9 - 1) * (10^9 - 1) \simeq 10^{18}$

Typecast to long.

→ Try all the possible variations with modulo.

$(a \times b) \% d = (a \% d * b \% d) \% d$

overflow

$10^9 + 7$? Prime No.

Time Complexity of Recursive code :-

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

Sum(N) ⇓ T(N)

↳ (N-1) + Sum(N-2)

↳ (N-2) + Sum(N-3)

↳ ...

⇒ In every iteration, one addition operation is there.

N → (N-1) → (N-2) → (N-3) → — — — — — → 1

Calls : N.

Total no. of add operation = N.

TC : O(N)

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

⇓ TC function → T(N)

T(N-1)

Add operation.

$T(N) = 1 + T(N-1)$ ← Put.

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

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

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

$= 2 + T(N-2)$

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

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

After N steps $T(N) = N + T(0)$ → (1)

$$T(N) = N$$

TC of sum(N) fun^{n} : $O(N)$

Ex :- $T(N) = 2T(N-1) + 1$ → fib(n) = fib(n-1) + fib(n-2)

$T(0) = 1$.

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

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

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

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

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

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

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

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

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

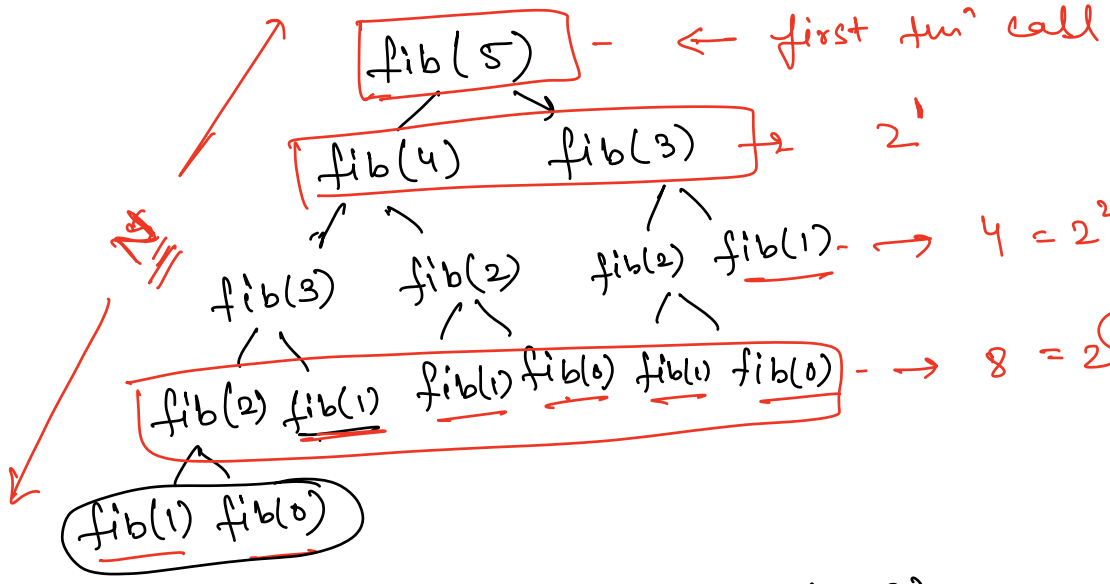
put $K = N$, $T(N) = 2^N * T(0) + (2^N - 1)$

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

$TC = O(2^n)$ Exponential TC.

$a^x \times a^y = a^{x+y}$.

$N+1$ (1)



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

HW: $T(N) = T(N-1) + T(N-2)$ → TC: $O(2^N)$

Ex: $T(N) = T(N/2) + 1$ ⇒ Power function, Binary search.

Recursion Tree:

$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \to \cdots \to \frac{N}{2^k} \to \cdots \to 1$

$\log_2 N$

TC : $O(\log N)$

$a^N \to a^{N/2} * a^{N/2}$

$\hookrightarrow a^{N/4} * a^{N/4}$

$a^{N/4}$ → ht

Ex:- * $T(N) = 2T(N/2) + 1$ ←

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

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

$= 4T(N/4) + 3$ ←

$= 4[2T(N/8) + 1] + 3$ ←

8nd step. $\boxed{T(N) = 8T(N/8) + 7}$

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

$T(1) \rightarrow 1$

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

$\log_2 N = K$

$T(N) = 2^{\log_2 N} \cdot T(1) + (2^{\log_2 N} - 1)$

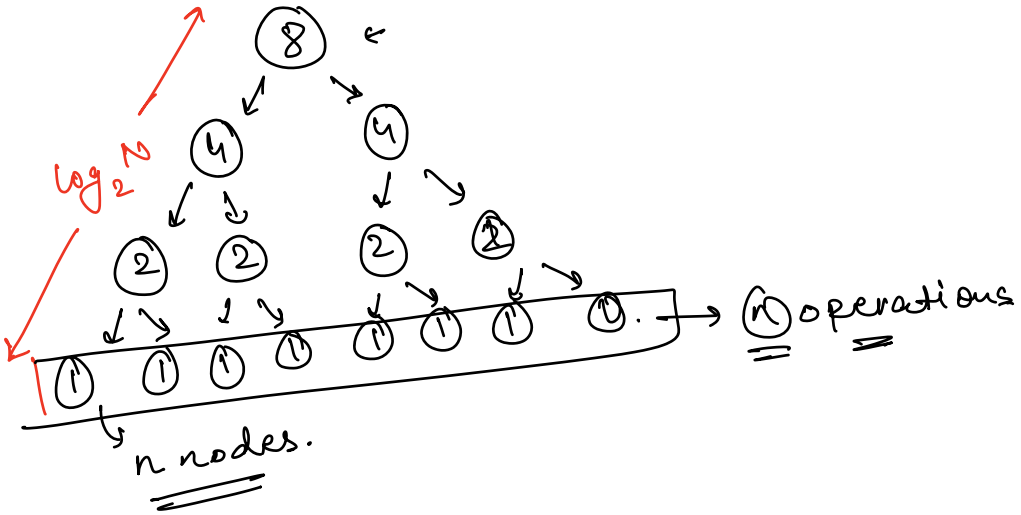
$= N \cdot 1 + N - 1$

$a^{\log_a x} = x$ ✓

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

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

Recursion Tree :-



Ex :- $T(N) = 2T(N/2) + N$ ← $O(N/2) \Rightarrow O(N)$

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

$$T(N) = 2\left[2T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N.$$

$$= 4T(N/4) + 2N.$$

$$= 4\left[2T\left(\frac{N}{8}\right) + \frac{N}{4}\right] + 2.N. \leftarrow$$

$$T(N) = 8T(N/8) + 3N. \leftarrow$$

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

Number of steps. ← $K = \log_2 N$

✓ $T(N) = 2^{\log_2 N} \cdot T(1) + \log_2 N \cdot N$.

$T(N) = N \cdot 1 + N \log N$.

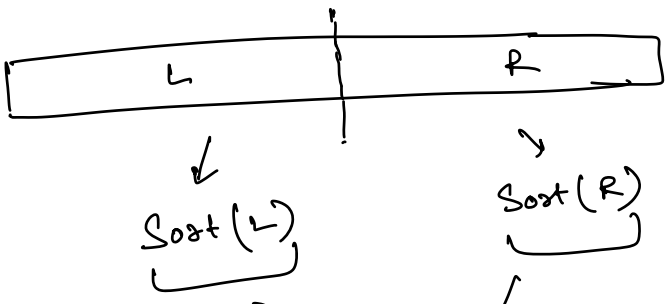
TC : $O(N \log N)$

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

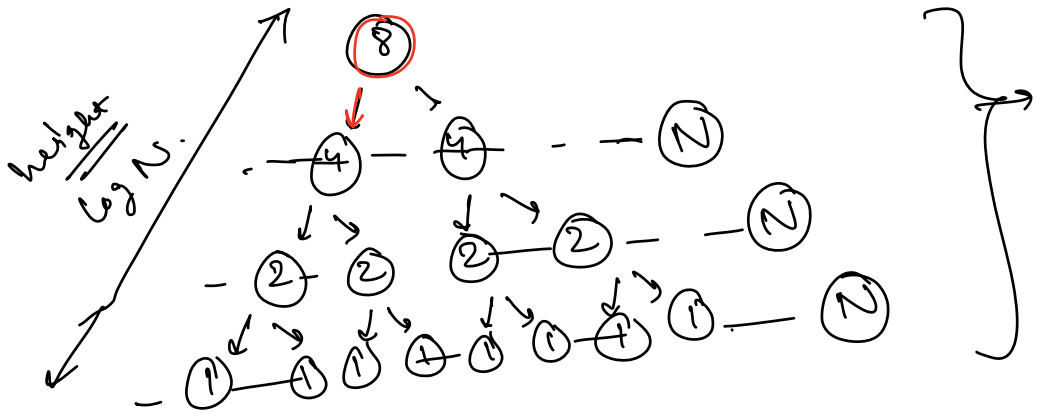
→ Merge Sort

→ Quick Sort.

```
f(n) {
  f(n/2)
  f(n/2)
  for(i=0 to N) {
  }
}
```



Merge → N operations.

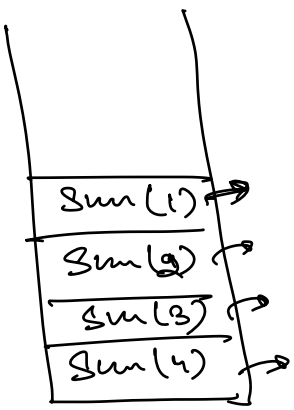


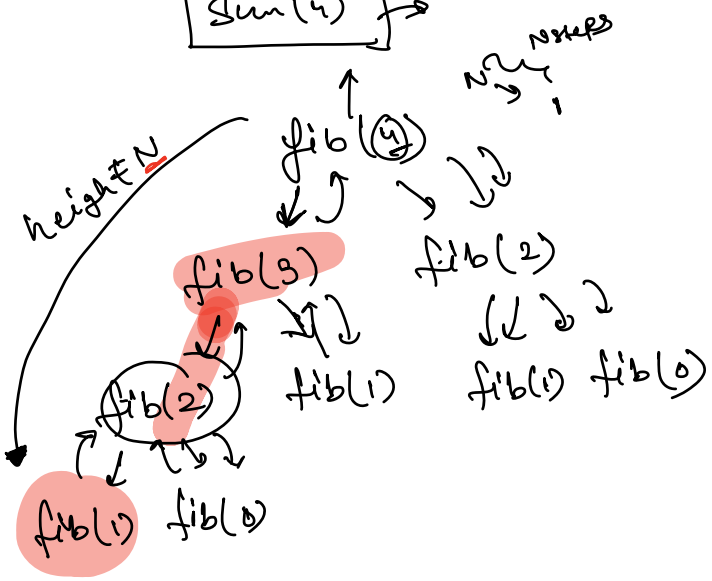
N log N

Space Complexity

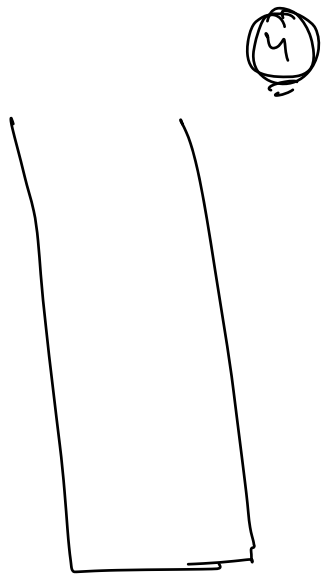
Sum(N) → Sum(N-1) → Sum(N-2) → - - - -

→ SC is the max. amount of stack used at any point of time.





(9) functions calls.



(4) → Depth of the tree
↳ (N)

SC : O(N)