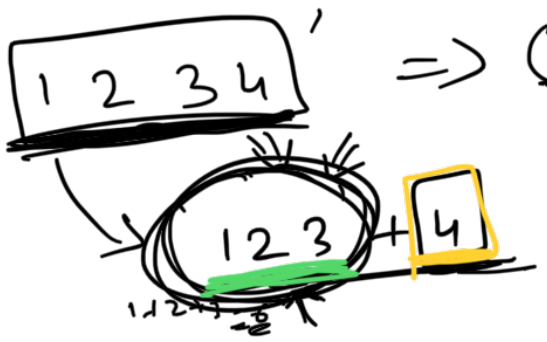
Recursion - 2

↳ More problems on recursion
↳ T.C. / S.C analysis

Q.1 Given a number, write a recursive code to find the 'sum of the digits'

eg. '1 2 3 4' => 1 + 2 + 3 + 4 = 10.



=> Last digit => % 10
remove last digit => / 10

1234 % 10 = 4

1234 / 10 = 123

123 ✓

=> correct ans.

// Assumption → int Sum(N){

// Base Case → if (N==0) return 0;

// main logic. → return Sum(N/10) + (N%10);

remove last

last digit

S

Sum(1234)

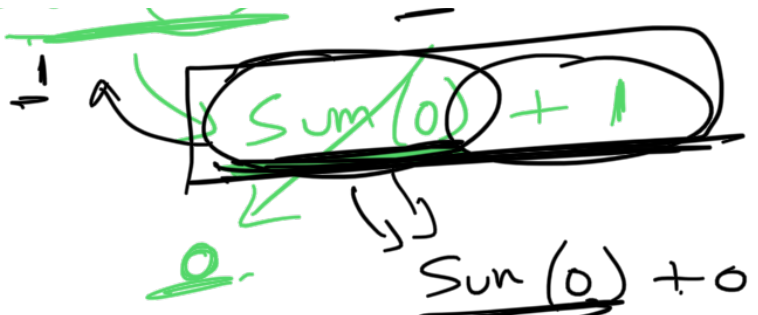
10

6 Sum(123) + 4

6 Sum(12) + 3

3 Sum(1) + 2

if (N < 10)
return N



Q.2 Implement the power function (recursively)

$$\boxed{Pow(a,n)} = a^N$$

eg. $a=3, n=3 \Rightarrow a^N = 3^3 = 3.3.3 = \boxed{27}$.

$$a^N = \underbrace{\overbrace{a * a * a * \ldots\ldots * a * a}^{(N-1)\text{ times}}}_{N\text{-times}}.$$

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

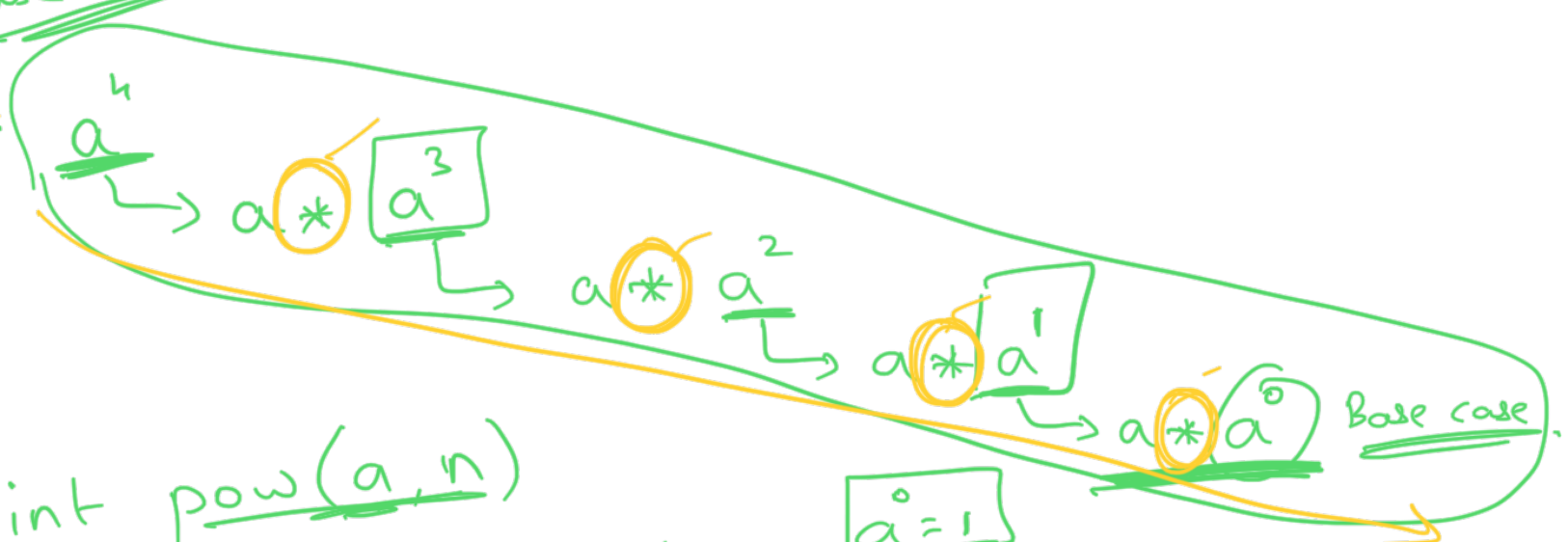
// main logic

$$pow(a,n) = pow(a, n-1) * a$$

Assumption

Base case, when $N=0$, $\boxed{a^0 = 1}$

$N=4$.




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

N → N multiplications

T.C => O(N) !!

pow(3,3) → 3 * pow(3,2) → 3 * pow(3,1) → 3 * pow(3,0) (Base Case)

27 = 9 · 3 = 3 · 3 · 3

$a^6 = a * a^5$

$= \frac{a^3 * a^3}{(a^3)^2}$

$a^7 = a * a^6$

$= a * a^3 * a^3$

$a^{2x} = a^x * a^x$

$a^x * a^y = a^{x+y}$

N = 16

$a^{16} \rightarrow a^8 * a^8 \rightarrow a^4 * a^4 \rightarrow a^2 * a^2 \rightarrow a^1 * a^1$

$a^N = \log_2 N$ multiplications

16 → 8 → 4 → 2 → 1

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

K steps

$\left(\frac{N}{2^K} = 1\right)$

$2^K = N$

$\Rightarrow \log_2(2^K) = \log_2(N)$

$\Rightarrow K \log_2(2) = \log_2 N$

$k = \log N$

N → N/2 → N/4 → N/8 → N/16

23 → 11 → 5 → 2 → 1

4 steps

$\log_2 N$

$a^{15} = a * a^{14}$ → $a^7 * a^7$

$a^6 = a^3 * a^3$

Pow (a, n) => a^N

a^N =

- if (N is even): $a^{N/2} * a^{N/2}$
- if (N is odd): $a * a^{N/2} * a^{N/2}$

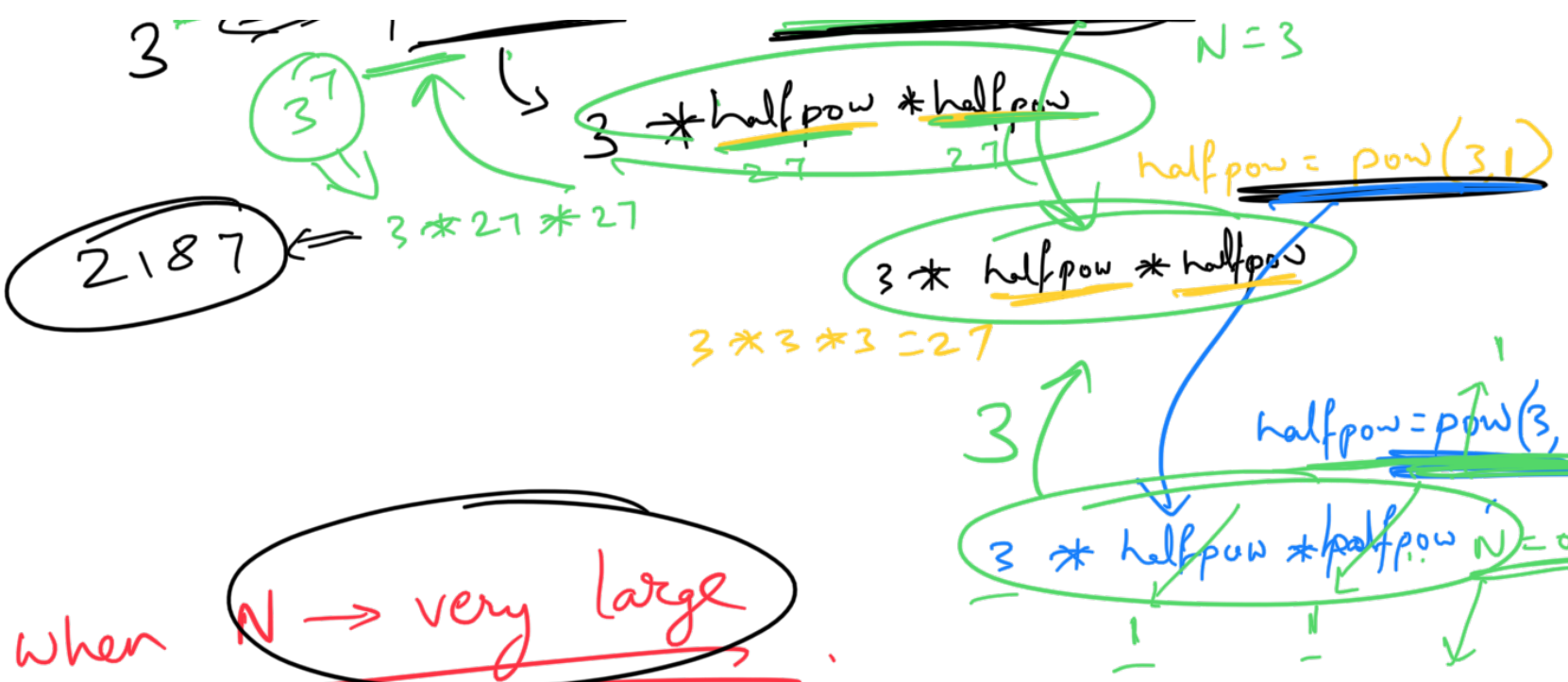
$(N-1)/2 = N/2$

a^{17} → $a * a^{16}$ → $a^8 * a^8$

```
int Pow(a, n)
{
    if (N==0) return 1          // Base Case  => 1 operation

    int halfpow = pow(a, n/2);

    // N is even
    if (N%2==0)
        return (halfpow * halfpow);      // main logic

    // N is odd
    else
        return (a * (halfpow * halfpow));

}
```

step = $\log_2 N$

pow(3, 7) → N=7 → halfpow = pow(3,3)



when N → very large .

(27) $\Rightarrow 10^9$ steps .

$\log_2(10^9) \Rightarrow$ 32 step .

MLE

Overflow .

pow(0,0) = 1

pow(a, n, d) . $\Rightarrow (a^N)\%d$

- prime number
- 10^9+7

mod

N = 10

fermat little theorem

pow(a, n, d) .

$a^N \% d$

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

$\log_2 N$ multiplication $\Rightarrow$ T.C $\Rightarrow O(\log_2 N)$

Time Complexity for recursive code

eg. sum(N) = N + sum(N-1)

↳ N-1 + sum(N-2)

sum(4) = 4 + (3+2+1)

sum(3)

Sum(N) ⇒ N additions

⇒ O(N).

N-2 + Sum(N-1)

Sum(0)

How to find T.C. for any recursion fn?

↳ Using recurrence relation

Logic ⇒ Sum(N) = N + Sum(N-1)

>1 operation

1+2+3+....+N

T(N).

Sum(0) = 0

N+1 numbers.

T(0) = 1

1+1+1+.. N-1

Time ⇒ T(N)

0(1) operations.

T(N-1)

T(N) ⇒ T.C. of sum of N numbers

Time.

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

operation

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

2

O(1)

$= 2 + T(N-2)$ → $1 + T(N-3)$

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

$= 3 + T(N-3)$

a+b, a×b, a-b, a/b → O(1).

0≠0

sum(0)

T(0) → 0.

Sum(0) ⇒ 0.

After K steps,

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

operation

After N steps,

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

$= N + T(0)$ → Base case.

$= N + 1$.

$T(N) = O(N)$

e.g. 2.

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

O(?)

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

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

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

$T(0) = 1$

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

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

Use substitution method

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

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

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

Substi $\Rightarrow 2^2 T(N-2) + 2^2 - 1$

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

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

$\Rightarrow 8(2T(N-4)+1)+7$

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

$\Rightarrow 2^4 \cdot T(N-4) + 2^4 - 1$

$1 + 2 + 4 + 8 + 16 + \cdots + 2^{K-1}$

$= 2^K - 1$

After K steps, $T(N) = 2^K \cdot T(N-K) + (2^K - 1)$

After N steps, $T(N) = 2^N \cdot T(N-N) + (2^N - 1)$

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

$= 2^N + 2^N - 1$

$= 2 \cdot 2^N - 1$

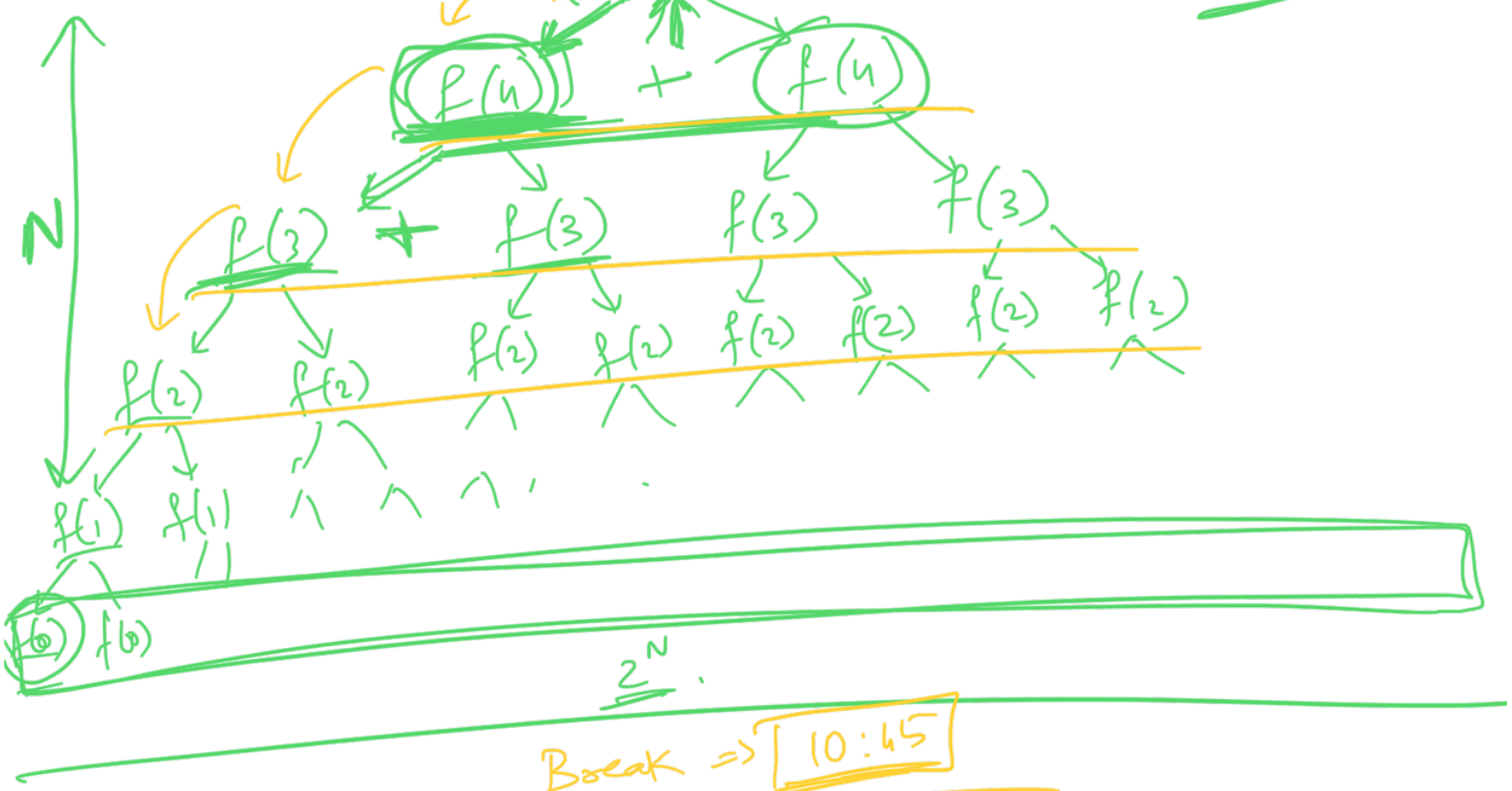
$$T(N) = O(2^N)$$

exponential.

$f(N) = f(N-1) + f(N-1)$ $\Rightarrow O(N)$

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

Recursion Tree



Break => 10:45

Quiz (3). $T(N) = T(N/2) + 1$

$T(0) = 1$

$T(1) = 1$

Use substitution

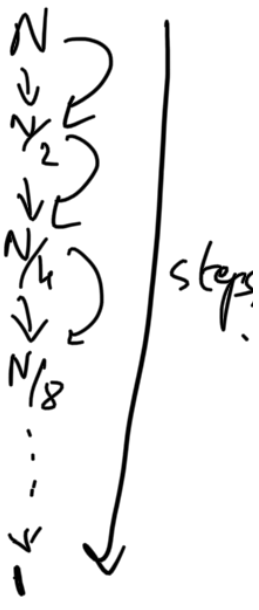
Power fn / Binary Search

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

$= T(N/4) + 2$

$= T(N/8) + 1 + 2$

$= T(N/8) + 3$



After K steps

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

After $\log_2 N$ steps

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

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

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

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

$2^K = N$

$K = \log N$

$\log N = \log 2^K$

operation

$$T(N) = O(\log_2 N)$$

Quiz ⑩. $T(N) = 2T(N/2) + 1$

$a^x * a^y = a^{x+y}$

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

$T(N) = \text{pow}(a, N/2) + \text{pow}(a, N/2) + 1$

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

use substitution.

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

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

$= 4T(N/4) + 3 \Leftarrow 2^2 T(N/2^2) + 2^2 - 1$

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

$= 8T(N/8) + 7 \Leftarrow 2^3 (N/2^3) + 2^3 - 1$

After K steps: $T(N) = 2^K T\left(\frac{N}{2^K}\right) + 2^K - 1$

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

After $\log_2 N$ steps: $T(N) = 2^{\log N} T\left(\frac{N}{2^{\log N}}\right) + 2^{\log N} - 1$

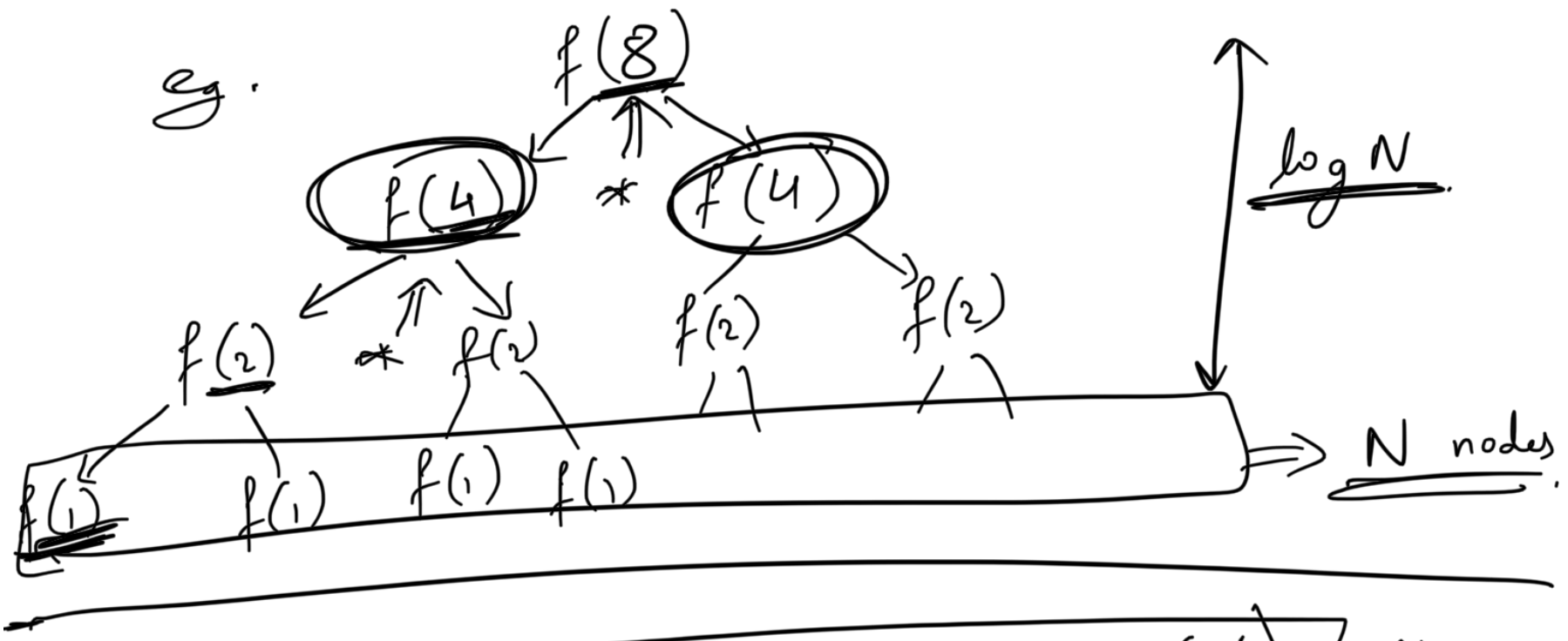
$K = \log N$

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

$= N + N - 1$

$= 2N - 1$

$$T(N) = O(N)$$



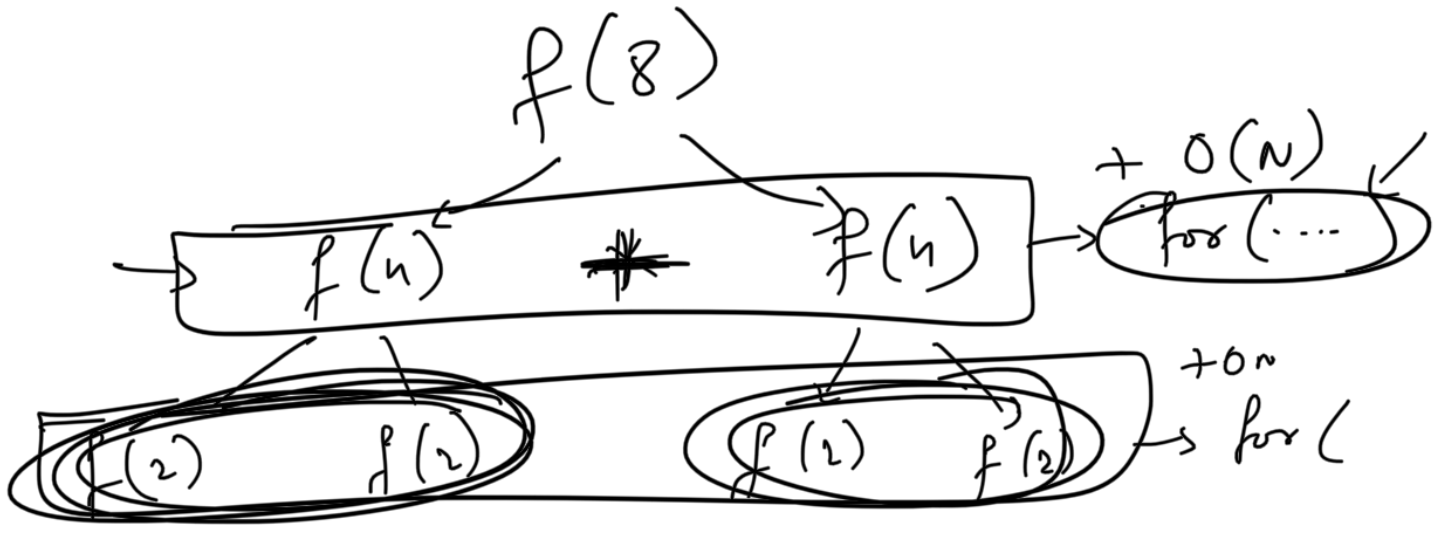
Quiz (5)

$$T(N) = 2\,T(N/2) + O(N) \Rightarrow ?$$

$T(1) = 1$

```
f(N)
{
  -> f(N/2)  -> T(N/2)
  -> f(N/2)  -> T(N/2)
  for (i = 0 ..... N)   } -> O(N)
  {                       N operations
  }
}
```

MergeSort
QuickSort



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

Substitute $N/2$.

$$= 2\left(2T(N/4) + N/2\right) + N$$

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

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

$$= 4\left(2T(N/8) + N/4\right) + 2N$$

$$= 8T(N/8) + N + 2N$$

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

After k steps $\quad T(N) = 2^k \cdot T\left(N/2^k\right) + k \cdot N$

After $\log N$ steps $\quad T(N) = 2^{\log N} \cdot T\left(\frac{N}{2^{\log N}}\right) + (\log N)N$

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

$2^k = N$

$k = \log N$

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

$$T(N) = O(N \log N)$$

Time Complexity

~~Space~~

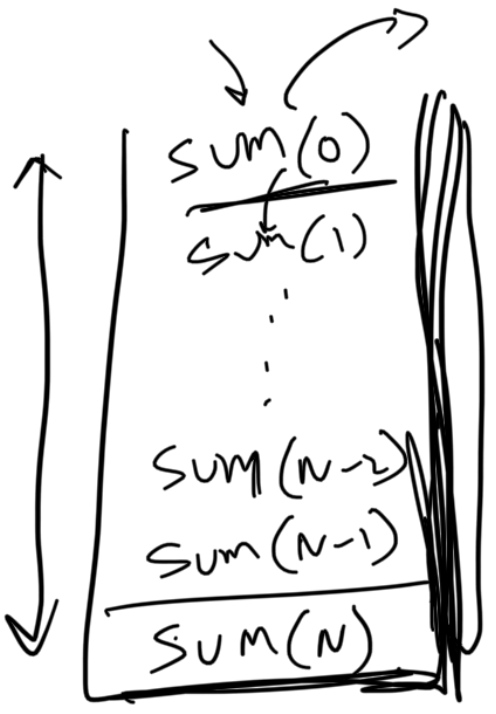
⇒ Maximum size of stack at any moment

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

$$N-1 + SUM(N-2)$$

$$\vdots$$

$$SUM(0)$$

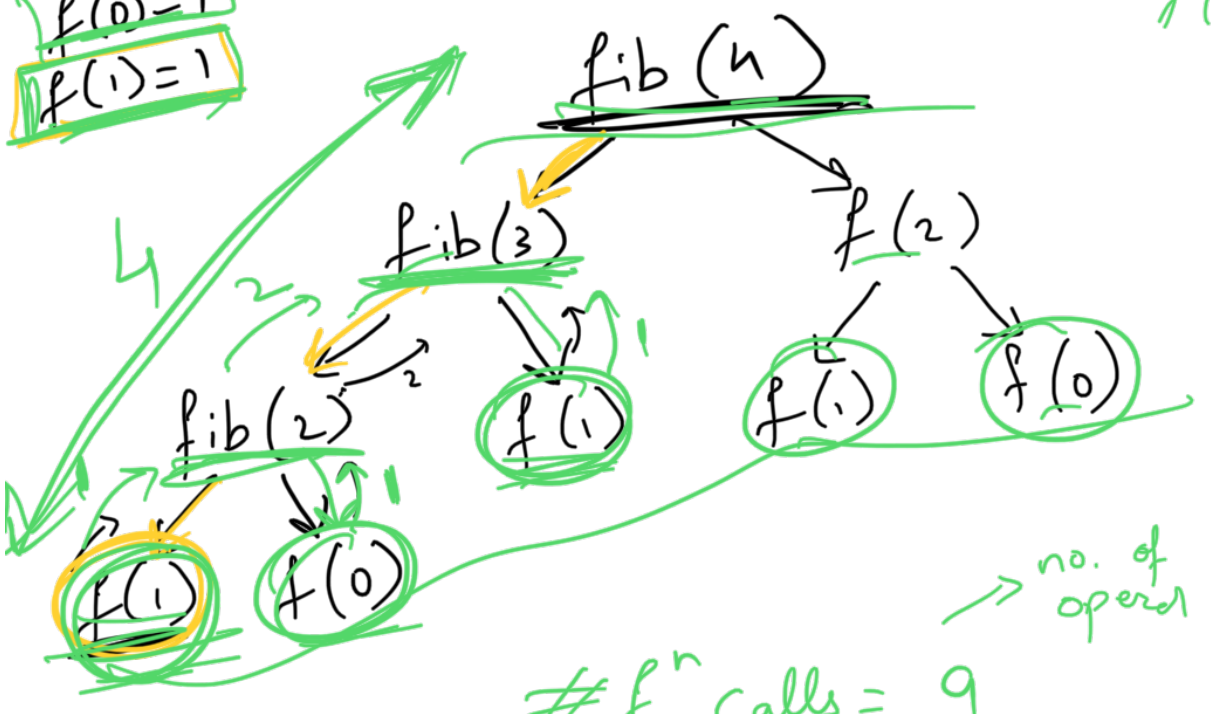


O(N) calls function ⇒ S.C ⇒ O(N)

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

$f(0) = 1$

$f(1) = 1$



no. of operations

f^n calls = 9.

max # f^n stack at any ⇒ 4 (space used)

S.C ⇒ O(N)

T.C ⇒

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

$a^{n/4} \quad a^{n/4} \rightarrow a^{n/8}$

$\log_2 N$.

1

$a^{n/4}$
$a^{n/2}$
a^n

S.C. for power $f^n \Rightarrow O(\log N)$.

$a^n \rightarrow (a^{n/2})^2 \rightarrow (a^{n/4})^2 \rightarrow (a^{n/8})^2 \rightarrow \cdots \rightarrow (1)^2$

$\log_2 N$.

pow(3,0)
pow(3,1)
pow(3,3)
pow(3,7)

$\log N$

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

3^7

$2^{\log_2 N} = N$

K

$\log_2^x = x \cdot \log 2$

$\log\left(2^{\log N}\right) = \log K$

$\log N = \log K$