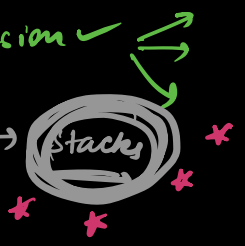
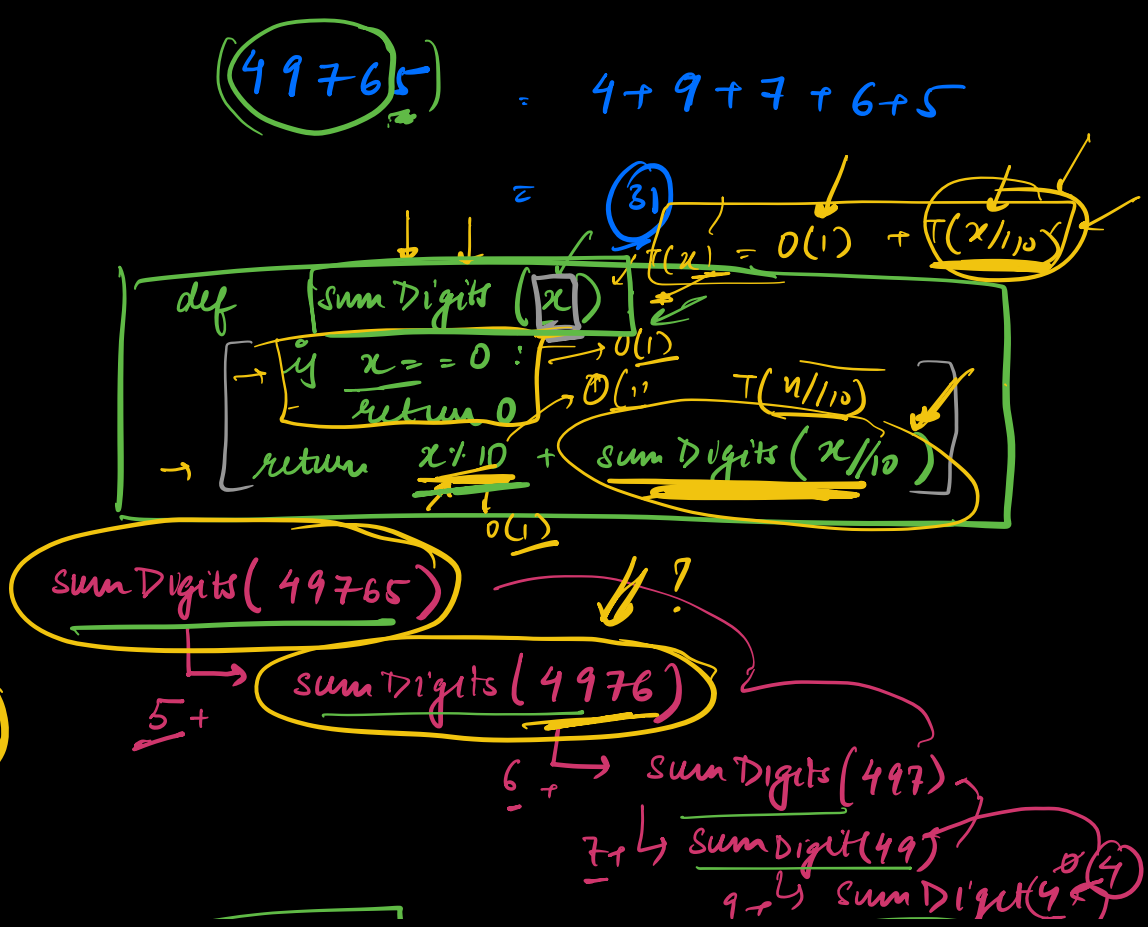
Recap

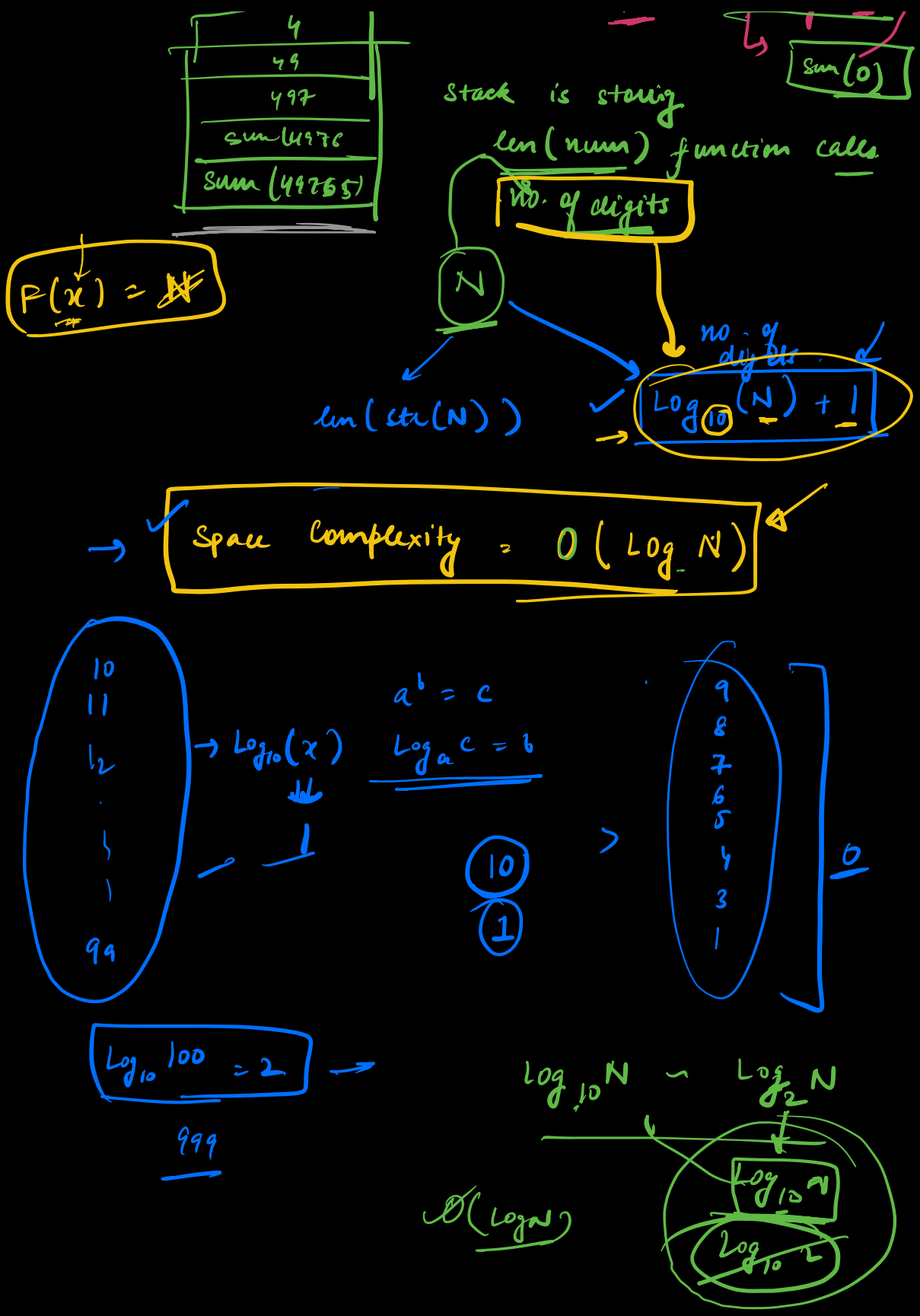
1. 3 step Approach for Recursion ✓
2. How recursion works. → 

---

1. How recursion is better than iteration
2. Time complexity and space complexity for recursion

Q) Given a number. Write a recursive program to calculate sum of digits.




4
49
497
sum(4976
sum(49765)
sum(0)
Stack is storing
len(num) function calls
no. of digits
N
P(x) = N
len(str(N))
no. of digits
Log10(N) + 1
Space Complexity = O(Log N)
10
11
12
99
Log10(x)
a^b = c
Log_a c = b
10
1
9
8
7
6
5
4
3
1
0
Log10 100 = 2
999
Log10 N ~ Log2 N
Log10 N
Log10 2
O(LogN)

I assume time taken for solving the problem for no. N is

$T(N) \Rightarrow T(N//10)$

$T(x)$ → that gives the time taken to execute the fn for a value x

Recursive Relation

$$T(N) = T(N/10) + O(1)$$

$$T(N/10) = T(N/100) + O(1)$$

$$T(N) = T(N/100) + 2 * O(1)$$

$$T(N) = T(N/1000) + 3 * O(1)$$

$$T(N) = T\left(\frac{N}{10^K}\right) + K * O(1)$$

$O(1)$

9/4

5/2

5/16

T(6)

$N//10^K = 0$

$\frac{N}{10^K} = 0 \Rightarrow 10^K > N$

$\log_{10} 10^K > \log_{10} N$

$10^k > N$

$k > \log_{10} N$

$k = \log_{10} N + 1$

$= O(\log_{10} N)$

$T(N) = \log_{10} N \times O(1)$

$T(N) = O(\log_{10} N)$

$O(\log N)$

N

$10^k > N$

N

$10^{\log_{10} N} \times 10$

N
N × 10

$= O$

Power fn

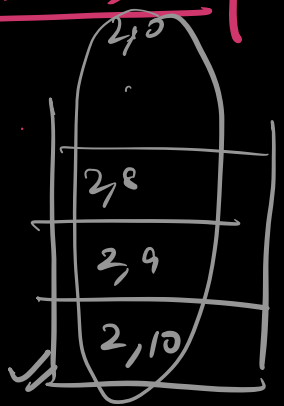
$$a^n = \underbrace{a \times a \times a \cdots}_{n \text{ times}}$$

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

Recursive relation to calculate a_n

$$\text{Power}(a, n) = \text{Power}(a, n-1) * a$$

```
def power(a, n)
    if n == 0
        return 1
    return a * power(a, n-1)
```



$$T(a, n) = O(1) + T(a, n-1)$$

$$\rightarrow T(a, n) = O(1) + T(a, n-2) + O(1)$$

$$= T(a, n-2) + 2 * O(1)$$

$$T(a, n) = T(a, n-k) + k * O(1)$$

$n - k = 0$

$n == k$

$$T(a, n) = T(a, 0) + N * O(1)$$

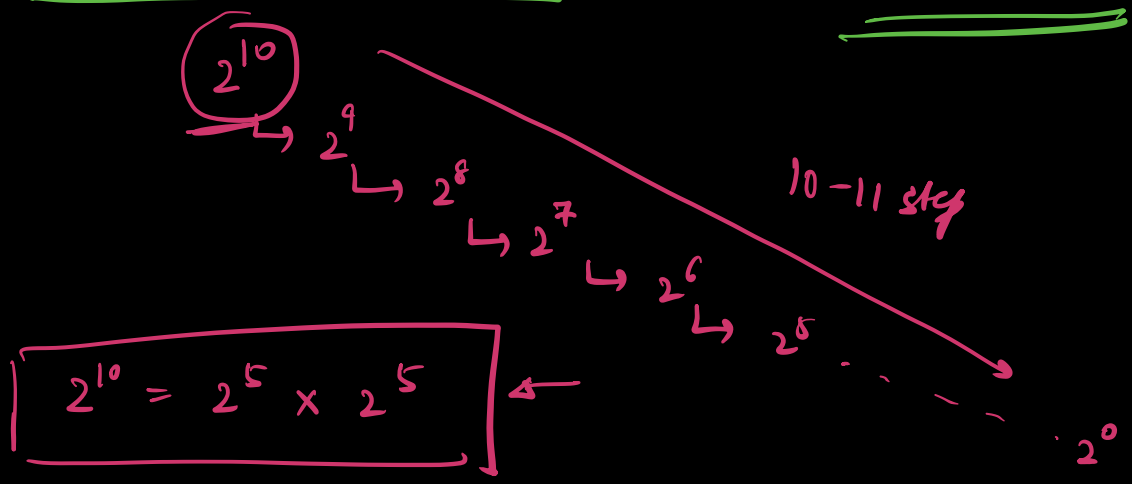
$T(a, 0) \rightarrow O(1)$

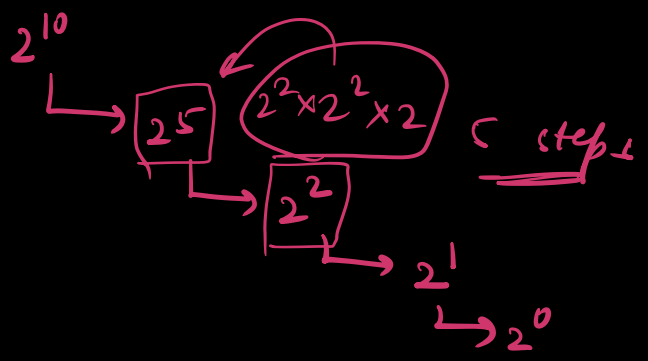
$T(a, n) = O(N)$

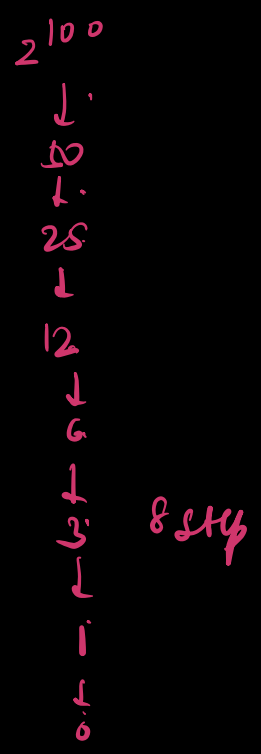
$SC = O(N)$

Optimised power fn :

TC → $O(\log N)$







Courses!

```
def power(a, n):
    if n == 0:
        return 1
    if n % 2 == 0:
        return power(a, n//2) * power(a, n//2)
    else
        return a * power(a, n//2) * power(a, n//2)
```

$$\Rightarrow T(a,n) = 2 * T(a, n//2) + O(1)$$

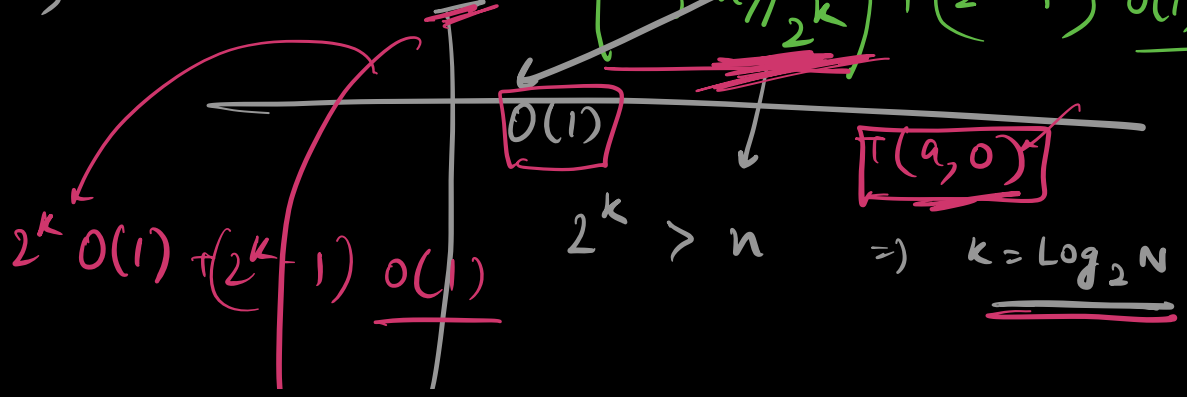
$$T(a, n//2) = 2 * T(a, n//4) + O(1)$$

$$T(a,n) = 2\left(2T\left(a, n//4\right) + O(1)\right) + O(1)$$

$$= 4T\left(a, n//4\right) + 2O(1) + O(1)$$

$$T(a,n) = 8T\left(a, n//8\right) + 4O(1) + 2O(1) + O(1)$$

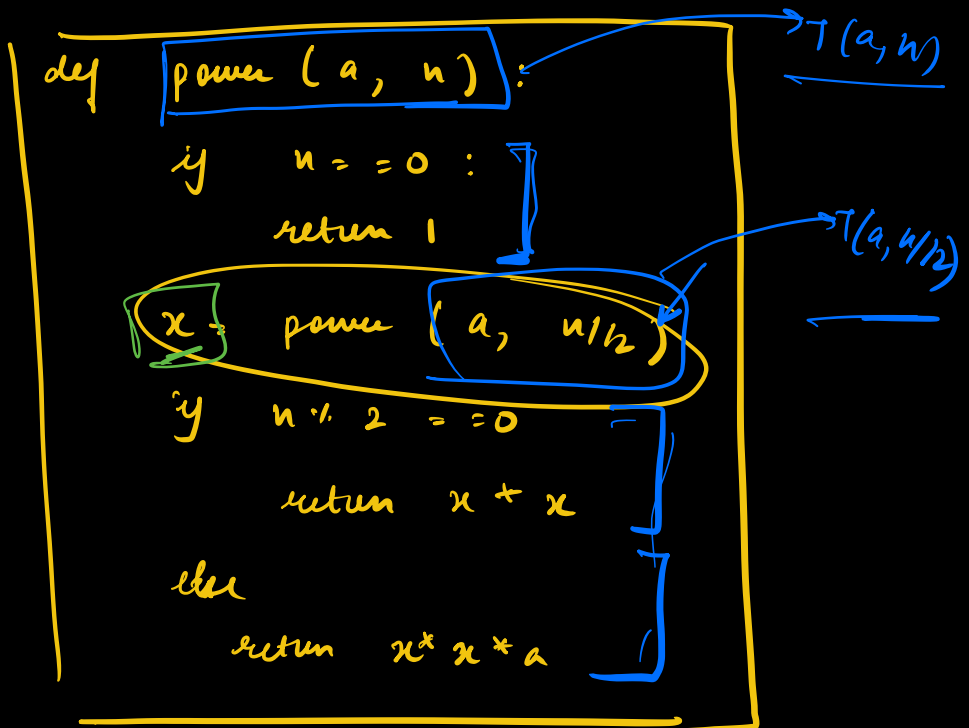
$$\Rightarrow T(a,n) = 2^k T\left(a, n//2^k\right) + \left(2^k - 1\right) O(1)$$



$2^{\log_2 N}\,(O(1) + (2^{\log_2 N} - 1)\,O(1))$

$= N * O(1) + (N-1)\,O(1)$

$\hookrightarrow O(N)$



$\Rightarrow T(a, n) = T(a, n/2) + O(1)$

$T(a, n/2) = T(a, n/4) + O(1)$

$T(a, n) = T(a, n/4) + 2 * O(1)$

$$T(a, n) = T(a, n/2^k) + k * O(1)$$

$O(1)$

$k = \log N$

$\log(N)$

$$O(\log N)$$

5 mins Break

power (a, n)

↓

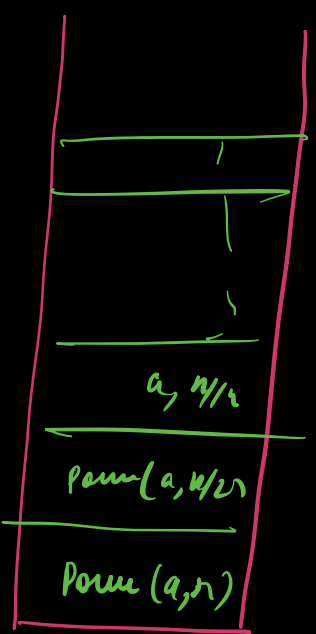
power $(a, n/2)$

↓

power $(a, n/4)$

⋮

power $(a, 0)$



$\Rightarrow \log_2 N$ $O(\log N)$

```
def mod_power(a, n, M):
    if n == 0
        return 1
    x = mod_power(a, n//2, M)
    if n % 2 == 0
        return (x * x) % M
    else
        return (a * ((x * x) % M)) % M
```

Assumption ⇓ Returns $a^n \% M$

$a^n \% M$

↳ $a^{n/2} \% M$

$\rightarrow (a * x * x) \% M$

Calculating TC for recursive relation

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

$T(n-1) = 2T(n-2) + O(1)$

$T(n) = 2(2T(n-2) + O(1)) + O(1)$

$= 4T(n-2) + 2O(1) + O(1)$

$= 8T(n-3) + 4O(1) + 2O(1) + O(1)$

K^{th} Term $2^k(n-k) + 2^{k-1}O(1) + 2^{k-2}O(1)$

$2^k T(n-k) + (2^k - 1)O(1)$

$2^n T(0) + (2^n - 1)O(1)$

$= O(2^n)$

① $T(n) = 2T(n/2) + O(1)$

$\Downarrow$

$O(n)$ ✓

② $T(n) = T(n/2) + O(1)$

$O(\log n)$ ~

merge sort

③ $T(n) = 2T(n/2) + O(n)$

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

$T(n) = 4T(n/4) + 2^2 O(n/2) + O(n)$

$T(n/4) = 2T(n/8) + 4O(n/4)$

$T(n) = 8T(n/8) + O(n) + 2^2 O(n/2) + 4^2 O(n/4)$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + O(n) + O(n/2) \cdots O\left(\frac{n}{2^{k-1}}\right)$$

$$k = \log_2 n$$

$$n + \left(n + n/2 + n/4 + \cdots n/2^{k-1}\right)$$

$$T(n) = 2T(n/2) + O(n)$$

$$T(n/2) = 2T(n/4) + O(n/2)$$

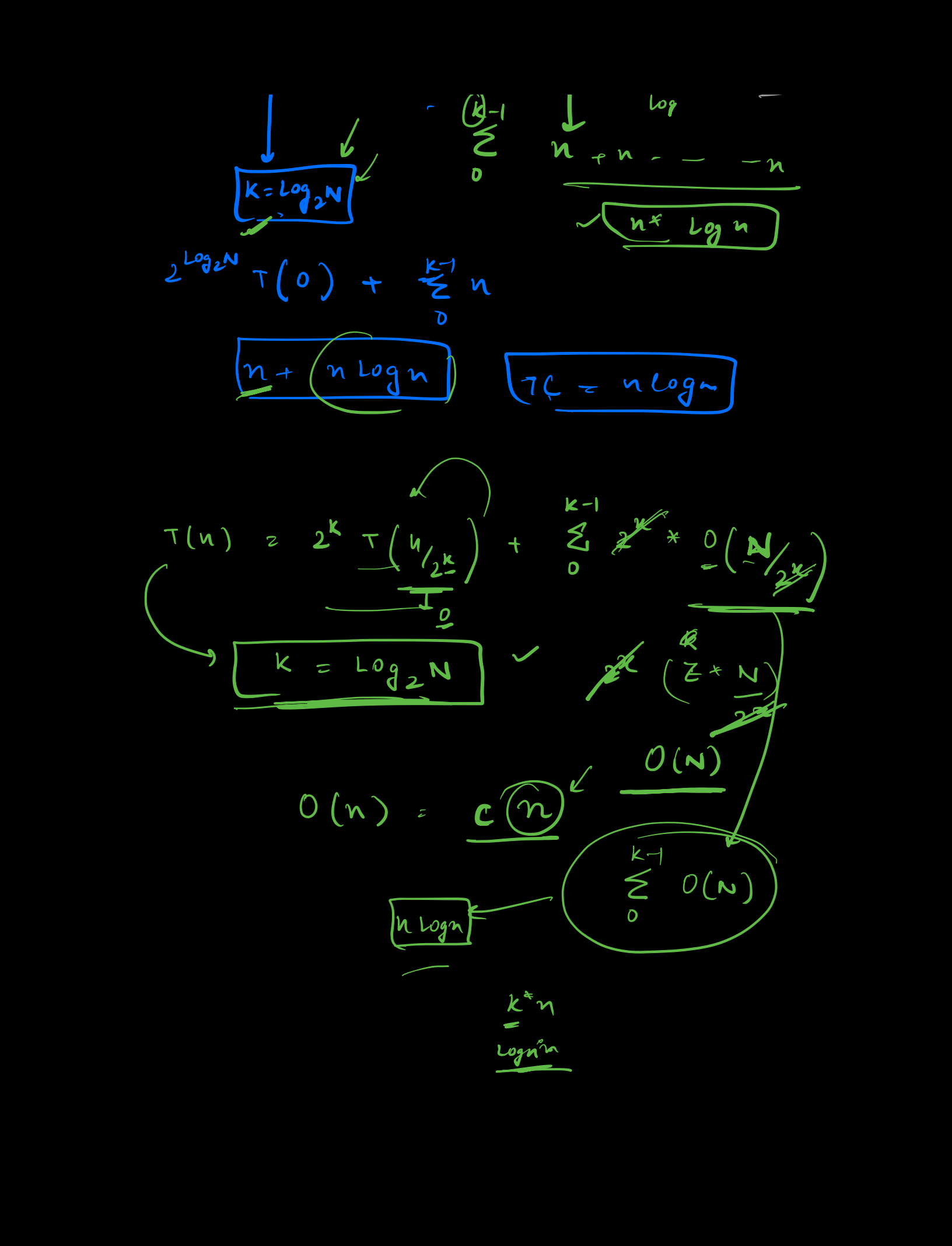
$$T(n) = 2\left(2T(n/4) + O(n/2)\right) + O(n)$$

$$= 4T(n/4) + 2O(n/2) + O(n)$$

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

$$= 8T(n/8) + 4O(n/4) + 2O(n/2) + O(n)$$

$$= 2^k T\left(\frac{n}{2^k}\right) + \sum_{0}^{k-1} 2^i O\left(\frac{n}{2^i}\right) \cdots O(n)$$


K = Log2N
n + n - - - n
n* Log n
2^Log2N T(0) + Σ n
n + n Log n
TC = n Log n
T(n) = 2^k T(n/2^k) + Σ 2^k * O(N/2^k)
k = Log2N
(k * N)
O(N)
O(n) = c n
Σ O(N)
n Logn
k*n
Logn^n

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

$T(n) = T(n-1) + T(n-2) + O(1)$

$T(n) <= 2 * T(n-1) + O(1)$

TC $O(2^n)$

H.W $fib(n)$

$fib(5)$

$O(N)$