Time & Space Complexity of Recursive Algorithms [Part - 2] - LIVE

Special class

Love Babbar • Sept 18, 2022

① Merge Sort

$$T(n) = K_1 + K_1 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + K_3 n + K_4 n$$

$$= K + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n(K_3 + K_4)$$

$$= K + 2 * T\left(\frac{n}{2}\right) + nK_5$$

$$T(n) = 2 * T\left(\frac{n}{2}\right) + n * K_5$$

Ist → 2* $\quad T(n) = 2T\left(\frac{n}{2}\right) + n * K_s$

IInd → 2^{1} $\quad 2T\left(\frac{n}{2}\right) = 4T\left(\frac{n}{4}\right) + \frac{n}{2} * K_s$

IIIrd → 2^{2} $\quad 4T\left(\frac{n}{4}\right) = 8T\left(\frac{n}{8}\right) + \frac{n}{4} * K_r$

→ 2^{a-1} $\quad 2^{a-1}T(1) = 2^{a}T(0)$

$2^{a-1} \times 2 = 2^{a}$ — a times

$a = \log_2 n$

$$T(n) = 2^{a} * T(0) + a * n * K_s$$

$$T(n) = 2^a \boxed{T(0)} + \underbrace{a * n * k}_{d}$$

$$= 2^a + a * n$$

$$= 2^{\log n} + n * a$$

$$= \underbrace{2^{\log n}}_{n} + n * \log n$$

$$n + n\log n = n\log n$$

$$\therefore \mathcal{O}(n\log n)$$

$2^{\log n} + n \log n$

$n = 2^{10}$

$2^{\log 2^{10}} + 2^{10} \times \log(2^{10})$

$= 2^{10 \times (1/2)} + 2^{10} \times \log(2^{10})$

$= 2^{10} + 2^{10} \times \log(2^{10})$

$= n + n \times \log(n)$

$\longrightarrow n(\log n + 1)$

T.C

$n \log n$

H/W → Master's theorem

T.C

counting → O(n) → expsol

factorial → O(n!)

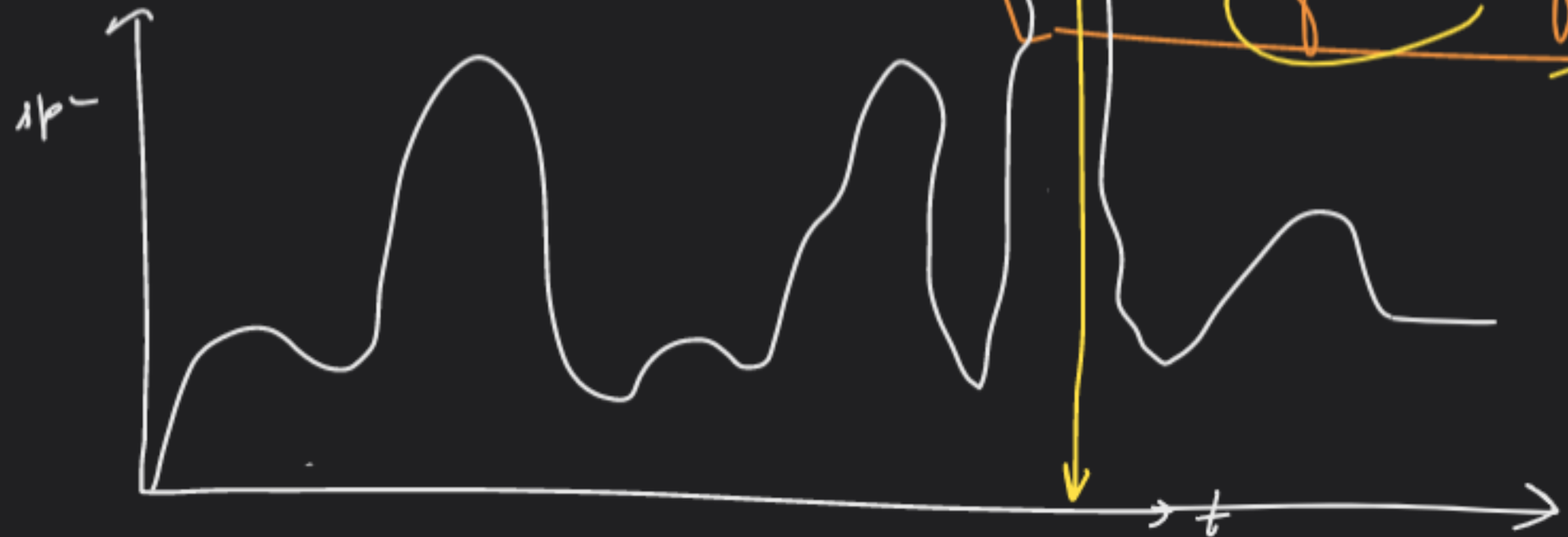
B.S → O(log n) → $\frac{n}{2^x} = 1$

Merge Sort → O(n log n) → 

fibonacci → O(2^n) → Rec tree

Space complexity → what ?

Space required to run an algo as a function of input

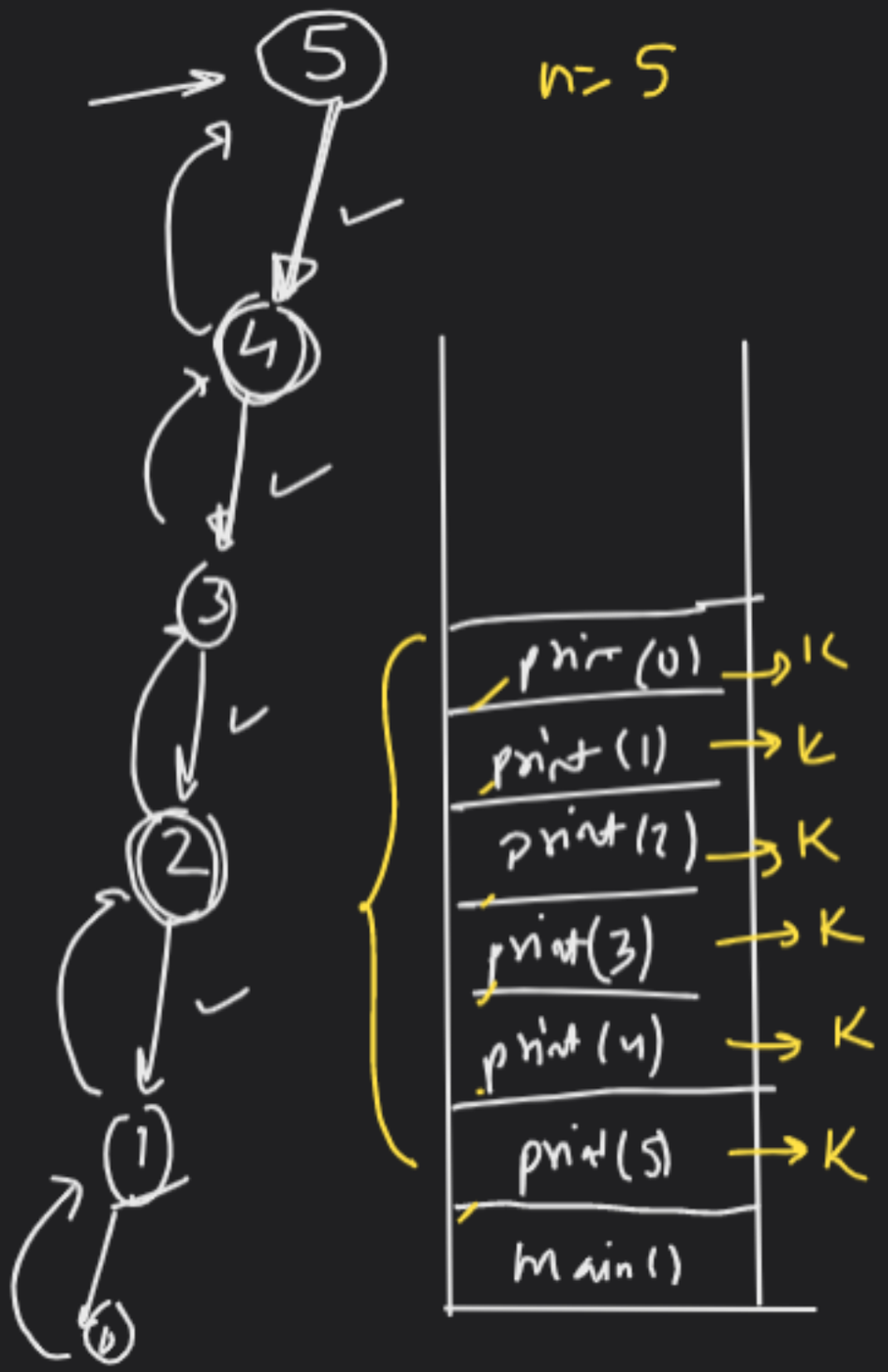


① Counting

```
void print (n)
{
//B.C
   cout << n;
//R.R
   print (n-1);
}
```

O(1)

Space



1 call -> k

n+1 call → (n+1) * k

$S.C \rightarrow nk + k$

$\rightarrow nk$

$\rightarrow n$

$S.C \rightarrow O(n)$

(2) factorial

```
int fact (int n)
{
  // B.C
  if (n == 0 || n == 1)
    return 1;

  return n * fact(n-1);
}
```

control

o(1)

f(5)

5 * f(4)

4 * f(3)

3 * f(2)

2 * f(1)

B.C

n


S.C → n * k
d
= .
= O(n)

③ Binary Search

binSearch(n size) → K

↓

binSearch(n/2 size) → K

↓

binSearch(n/4 size) → K

⋮

binSearch(1 size) → K

(a times)

a = log n

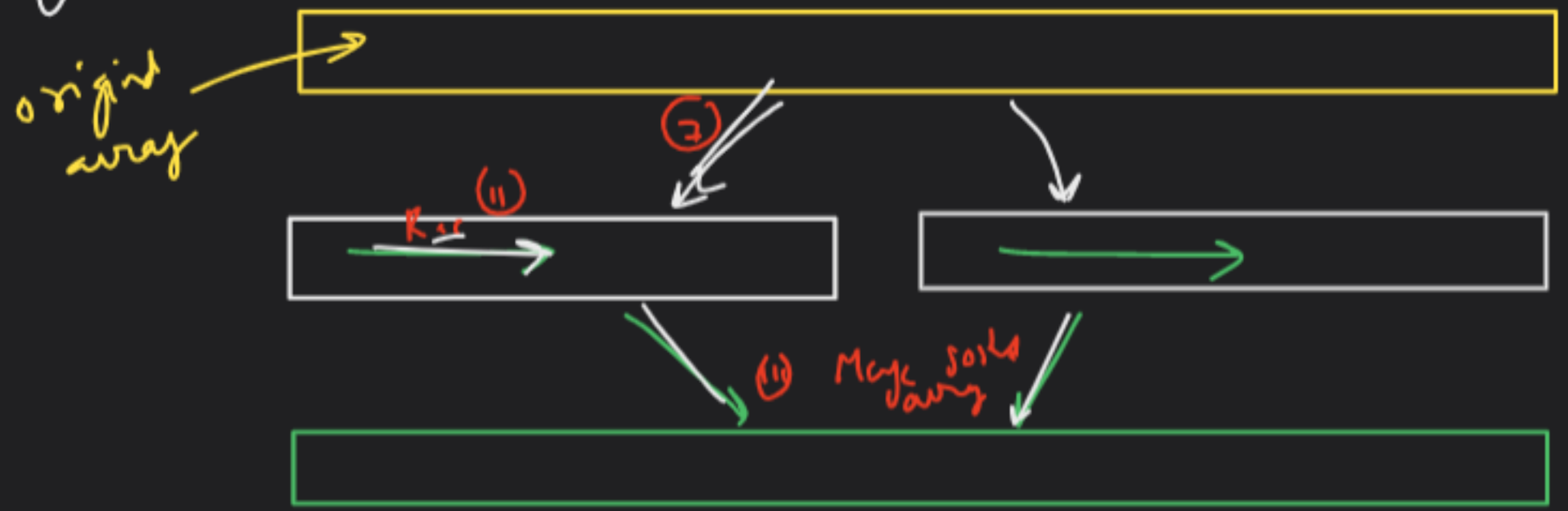
```
bool binSearch(arr, s, e, target)
{
    //B.C
    if (s > e)
        return false;

    int mid = (s+e)/2;

    if (arr[mid] == target) return true;

    if (arr[mid] > target)
        return binSearch(arr, s, mid-1, target);
    else
        return binSearch(arr, mid+1, e, target);
}
```

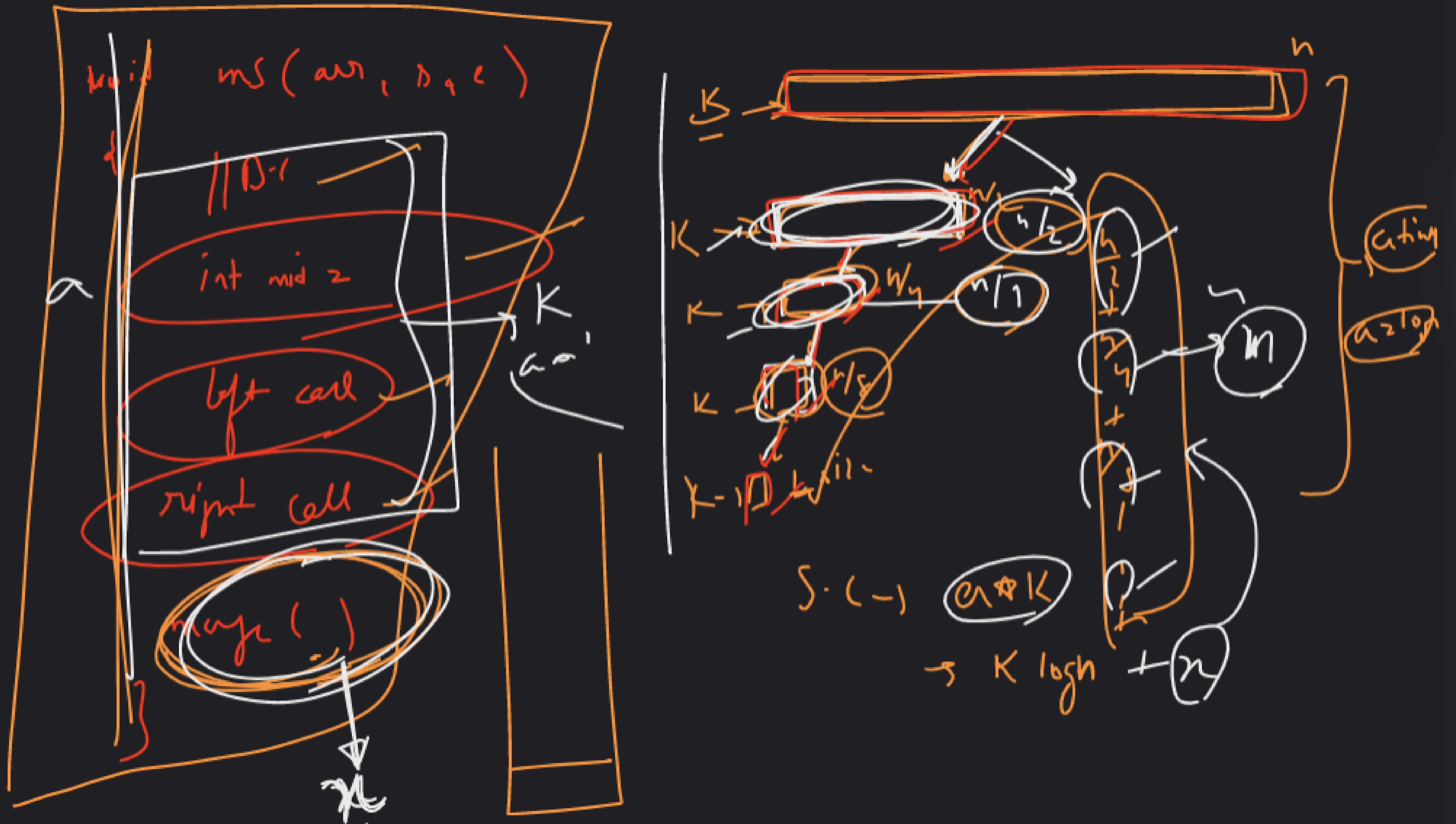
S.C → a * K

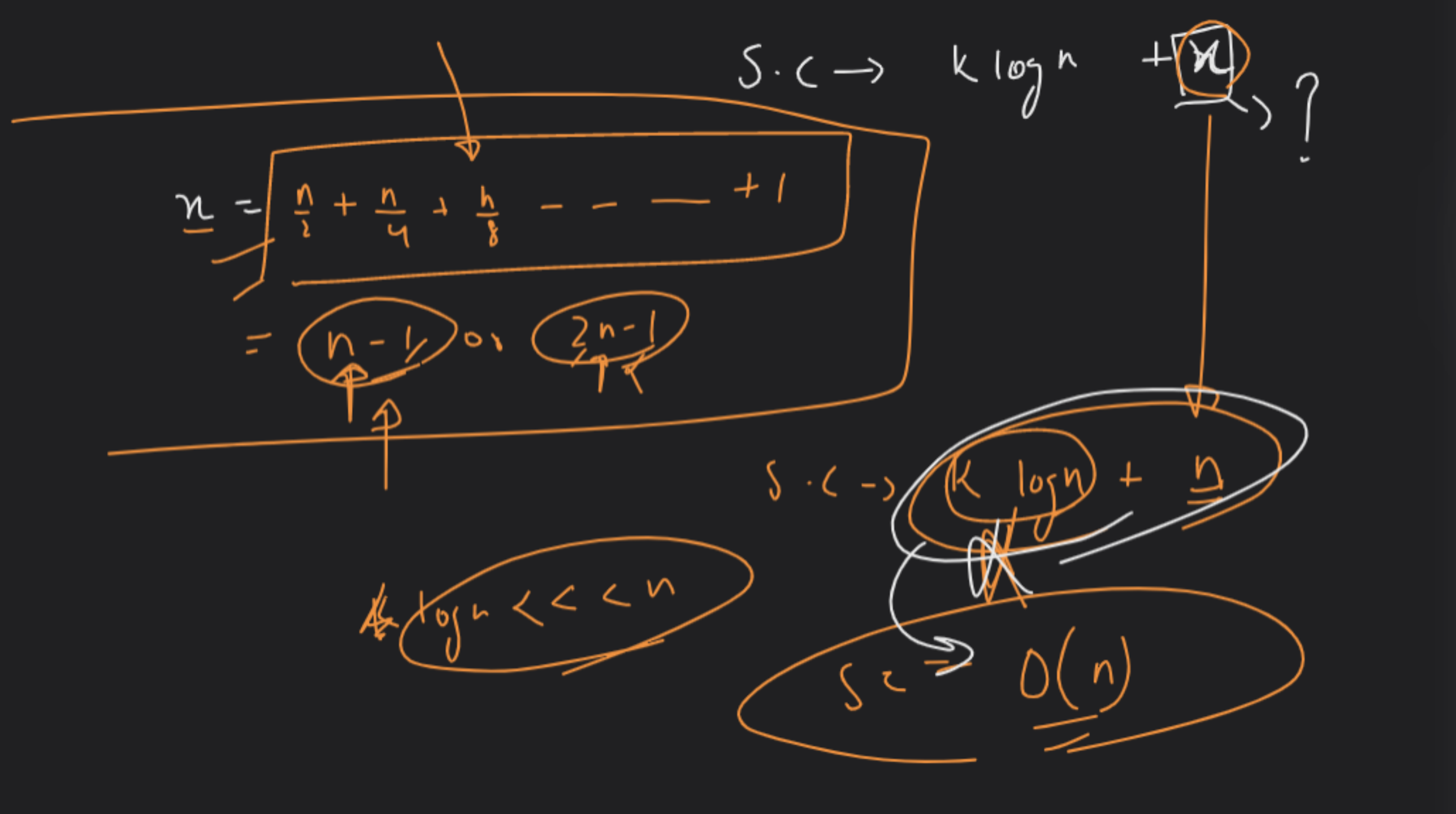
= K * a = ~~K~~ * logn

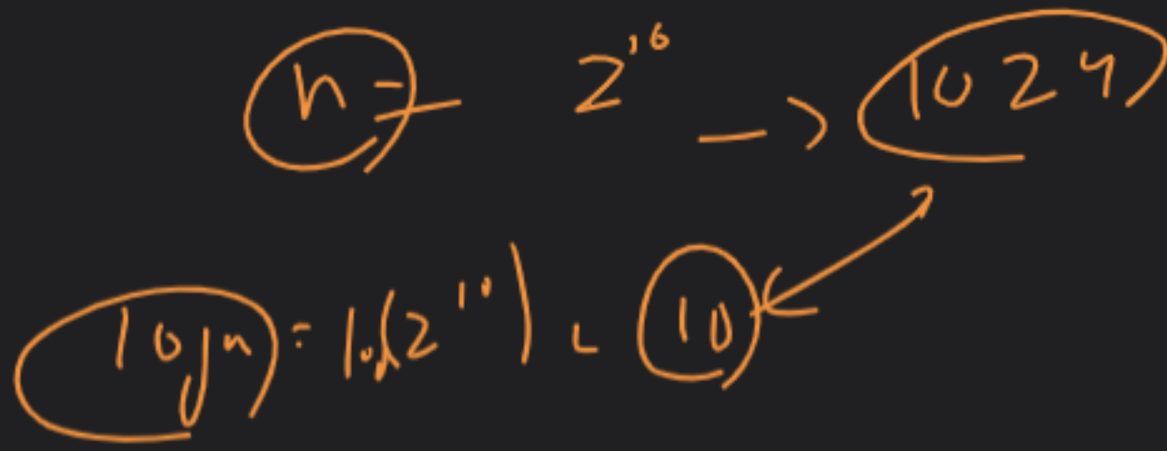
S.C → O(logn)

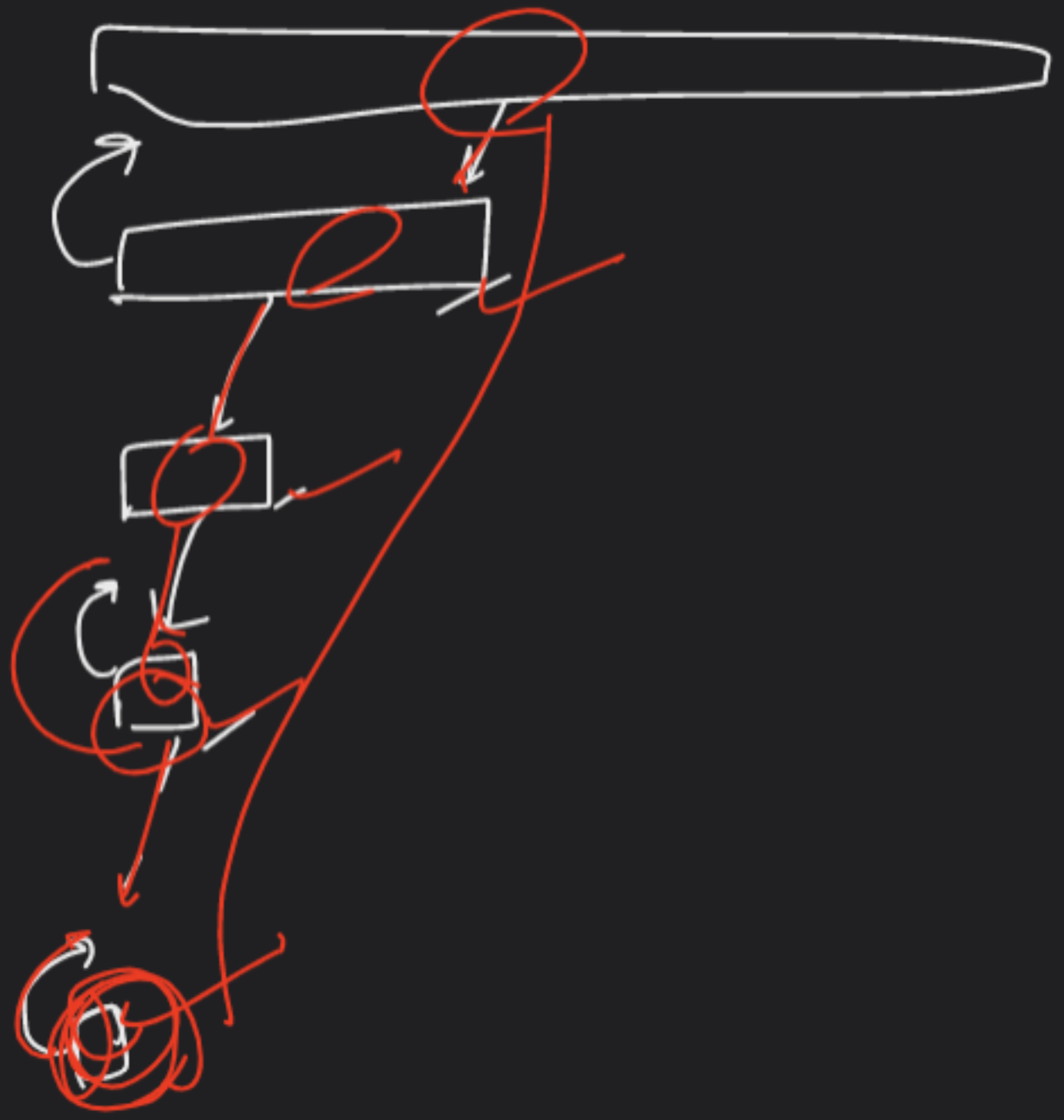
(4) Merge Sort :-

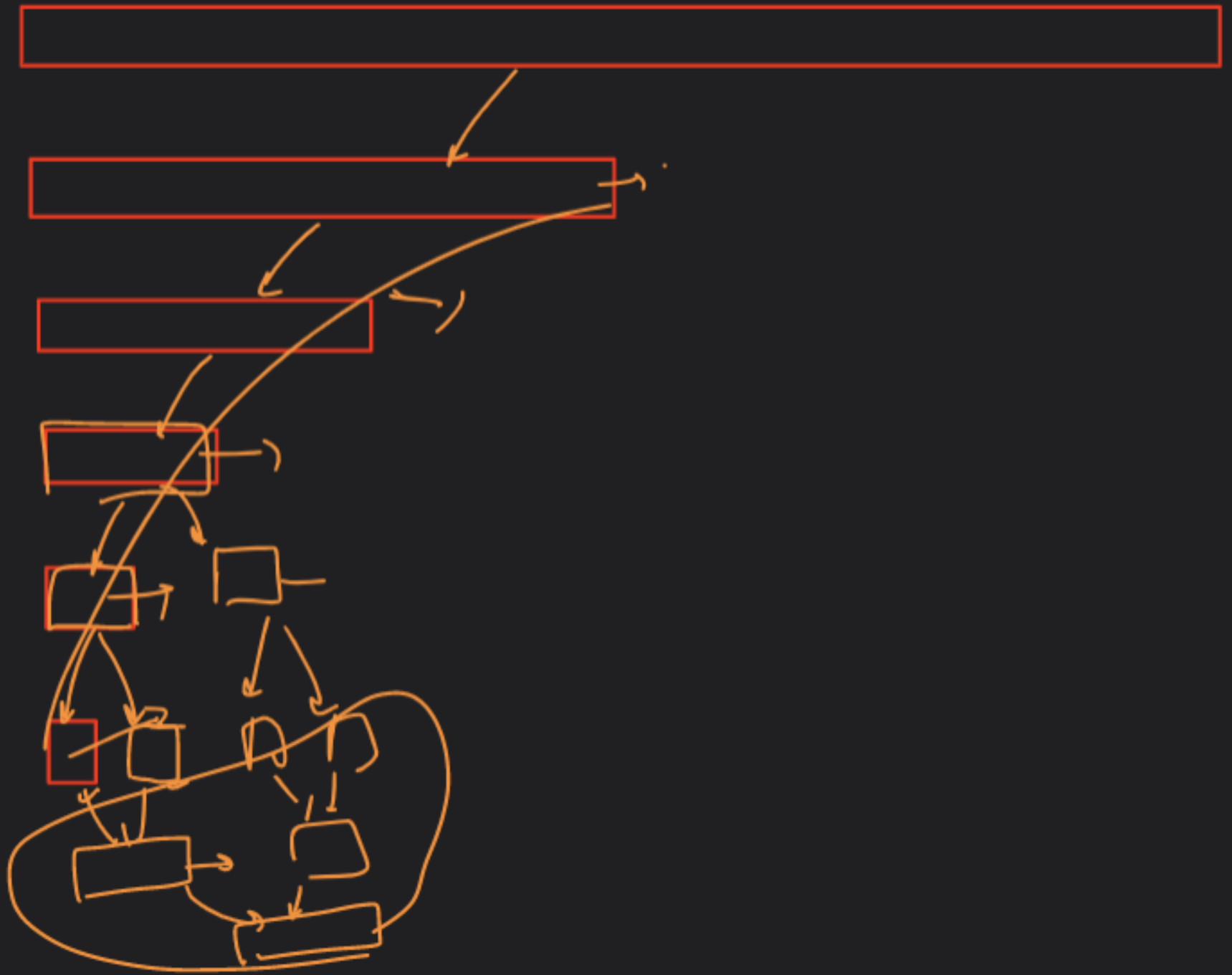



void ms(arr, s, e)
{
int mid =
left call
right call
merge( )
}
K
n
n/2
n/4
n/8
K log n + n


S.C → k log n + n → ?
n = n/2 + n/4 + n/8 - - - - +1
= n-1 or 2n-1
log n << < n
S.C -> k log n + n
S.C => O(n)


n = 2^16 -> 1024
log n = log(2^10) = 10
10



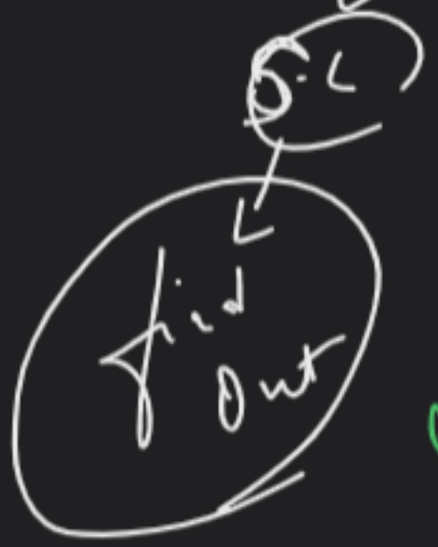


6) Fib Series

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

    return fib(n-1)+ f(n-2);
}
```

H/W

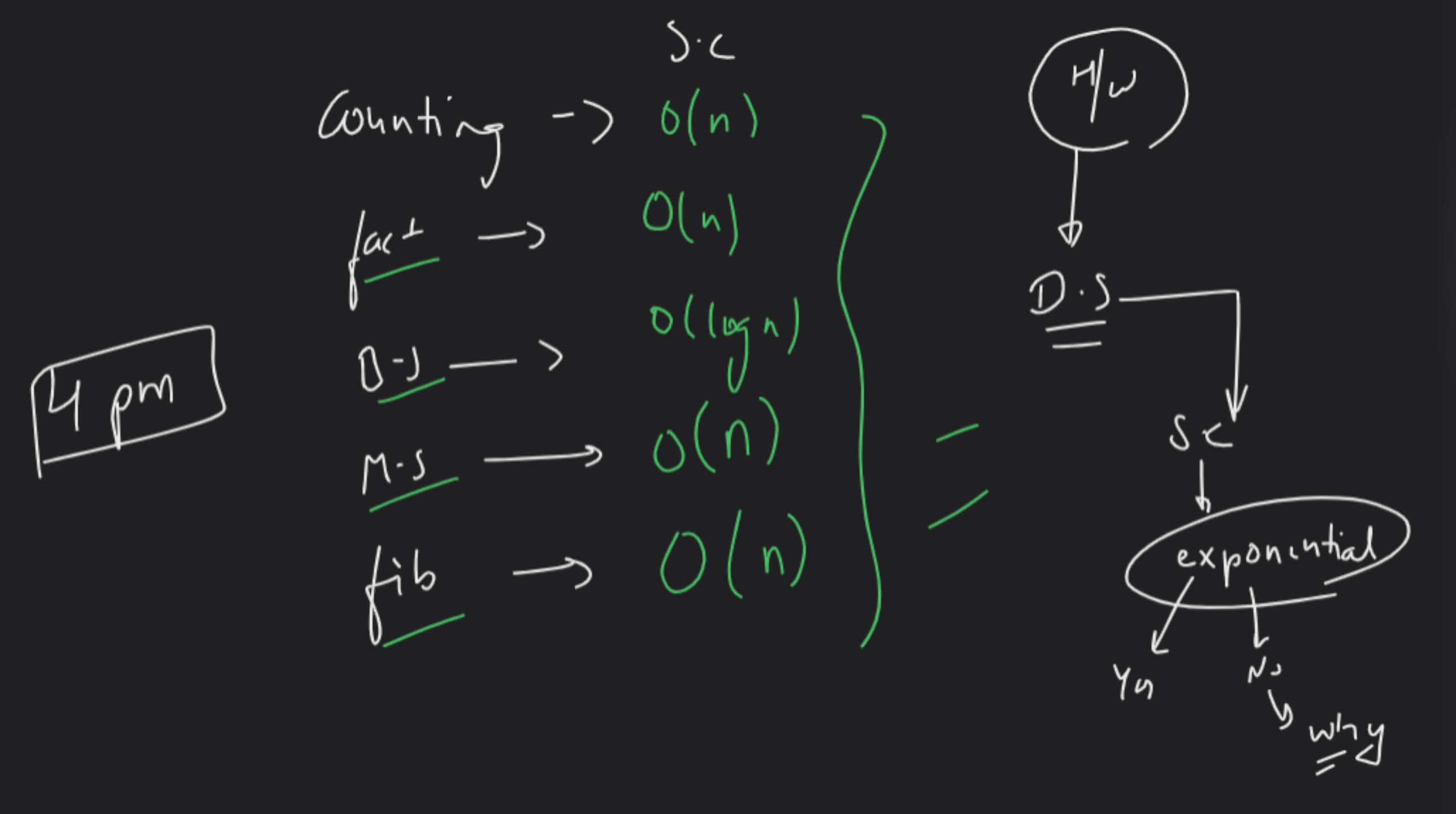
Codus



S.C → not k

= O(h)

O(2^n)


S.C
Counting -> O(n)
fact -> O(n)
O-J -> O(log n)
M-S -> O(n)
fib -> O(n)
4 pm
H/w
D.S
S.C
exponential
Yes
No
why