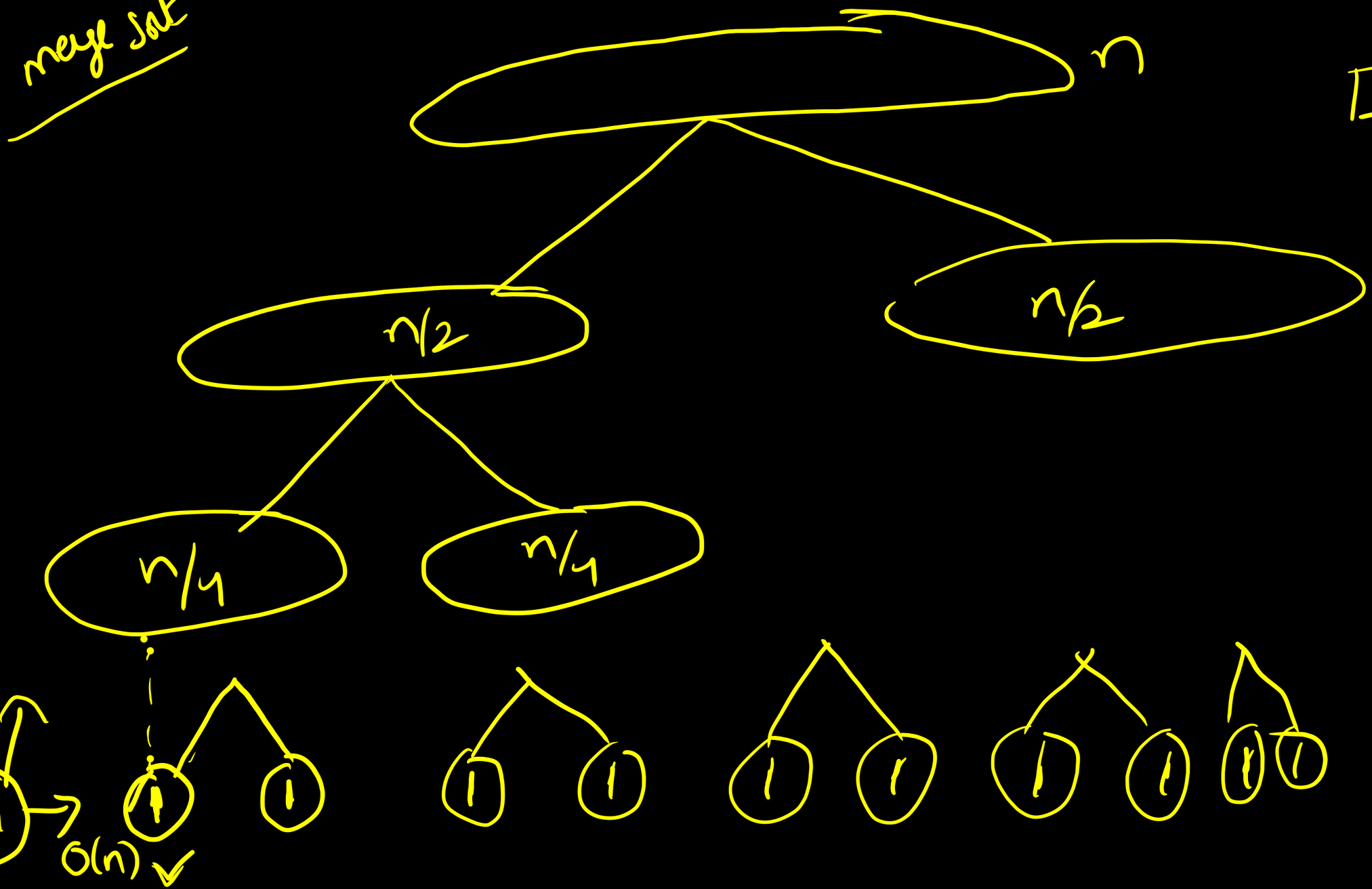
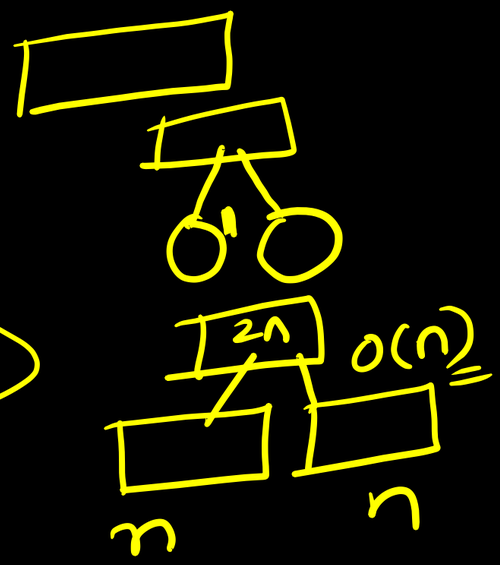
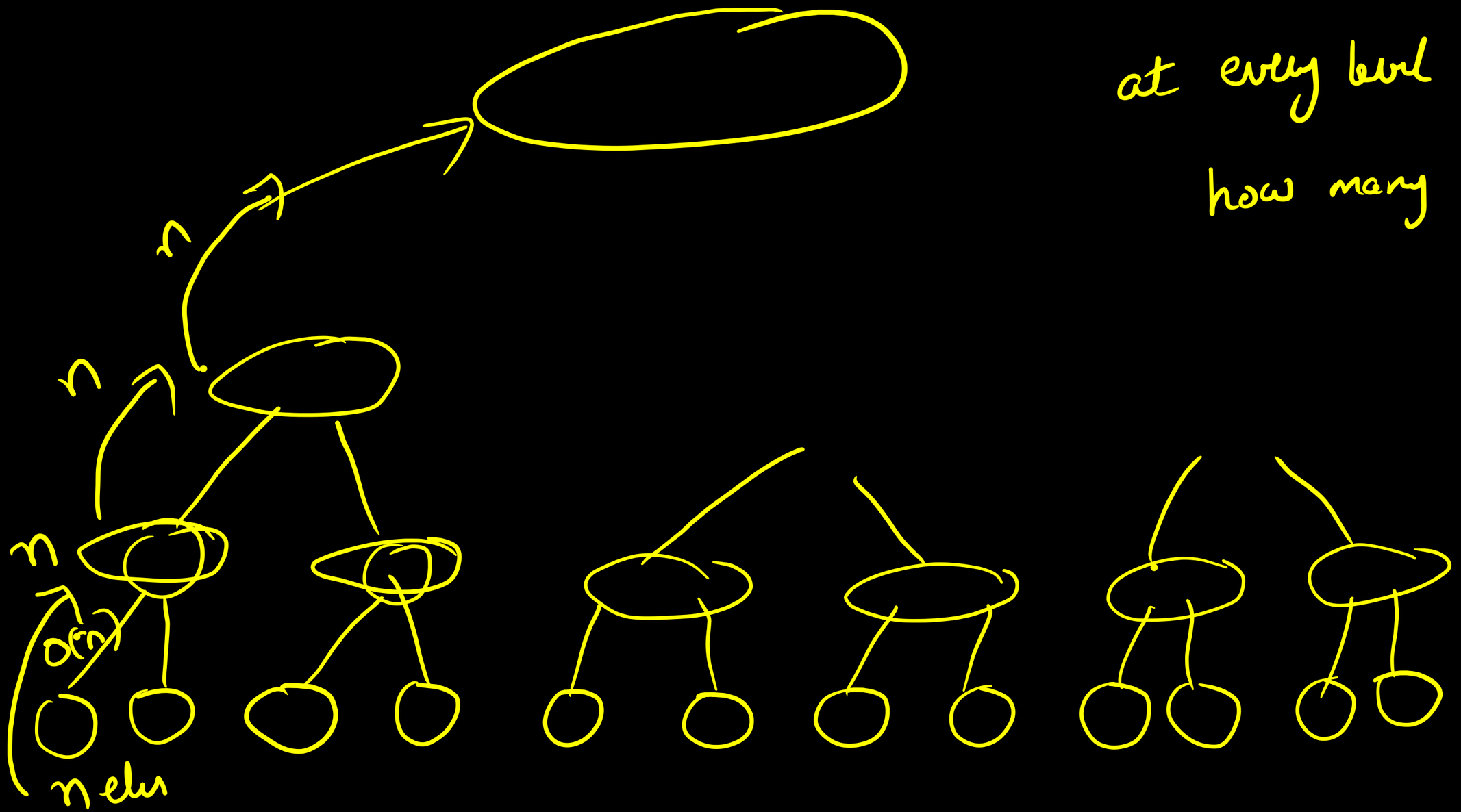
merge sort

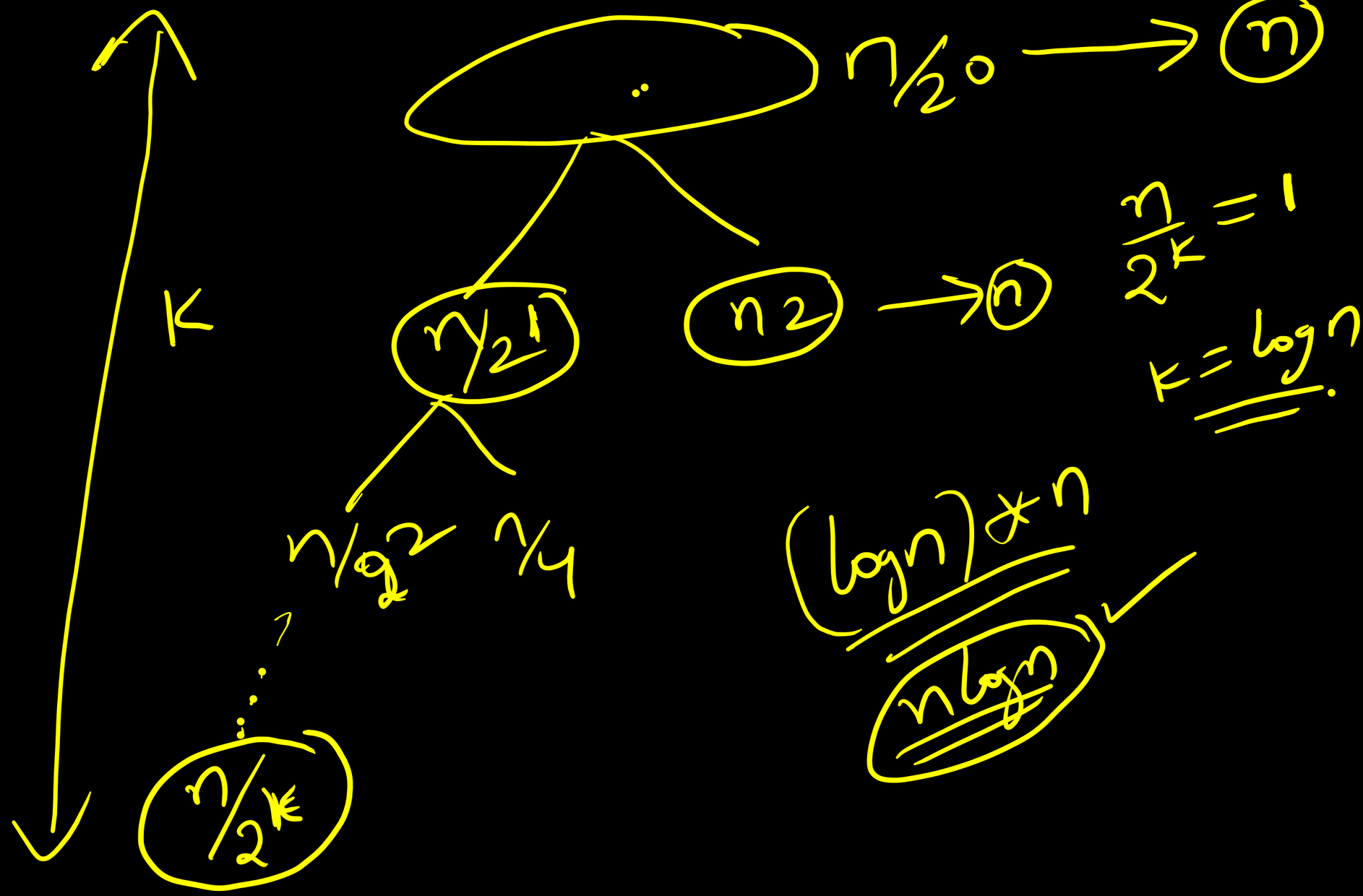






at every level $O(n)$

how many levels


n/2 → n
K
n/2 1
n2 → n
n/2^2
1/4
n/2^K
n/2^K = 1
K = log n
(log n) * n
n log n

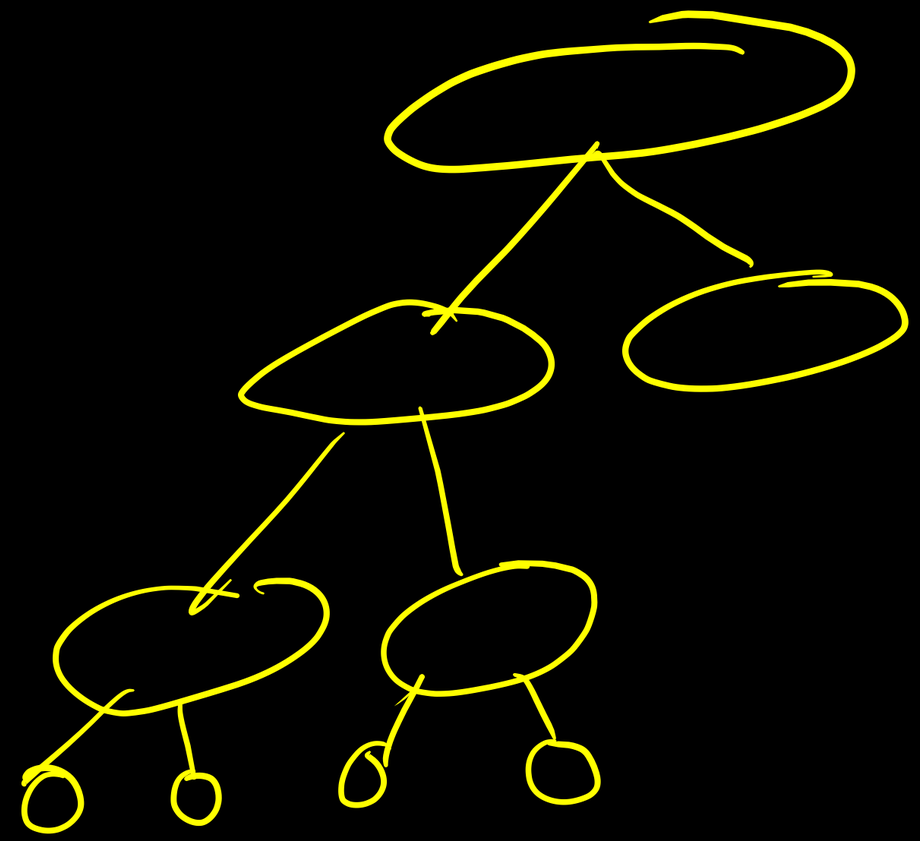
merge sort

↓

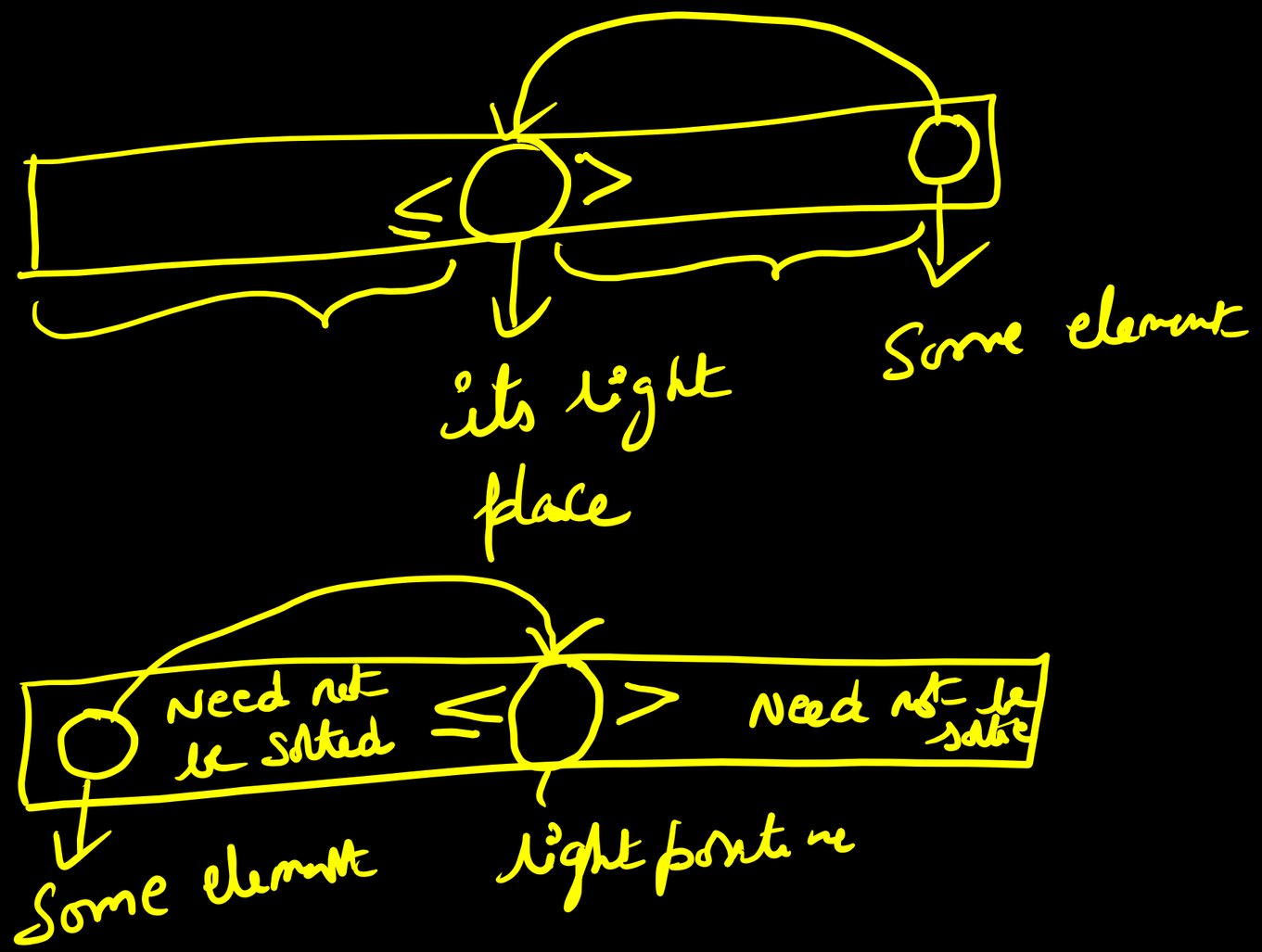
sorting in every node. → single element array is already sorted.

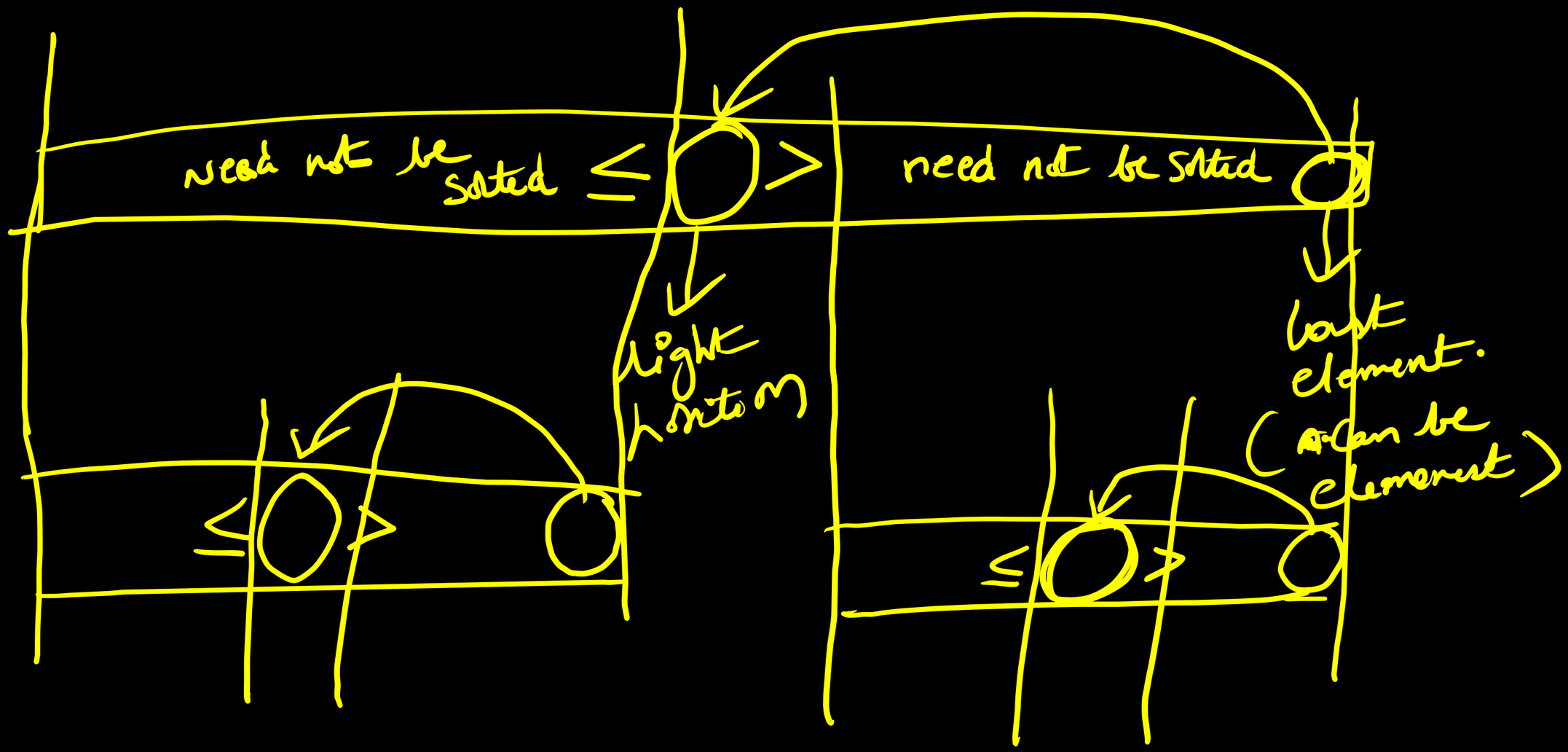
||

we keep on merging to achieve the sorting.

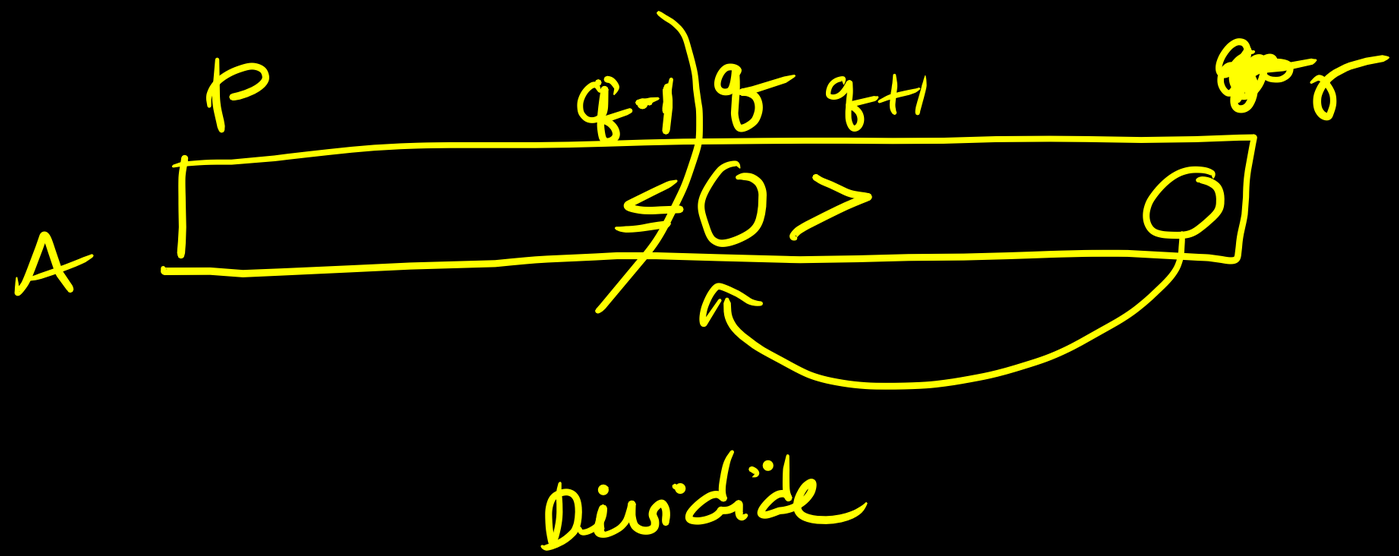


Quick sort → uses divide and conquer.

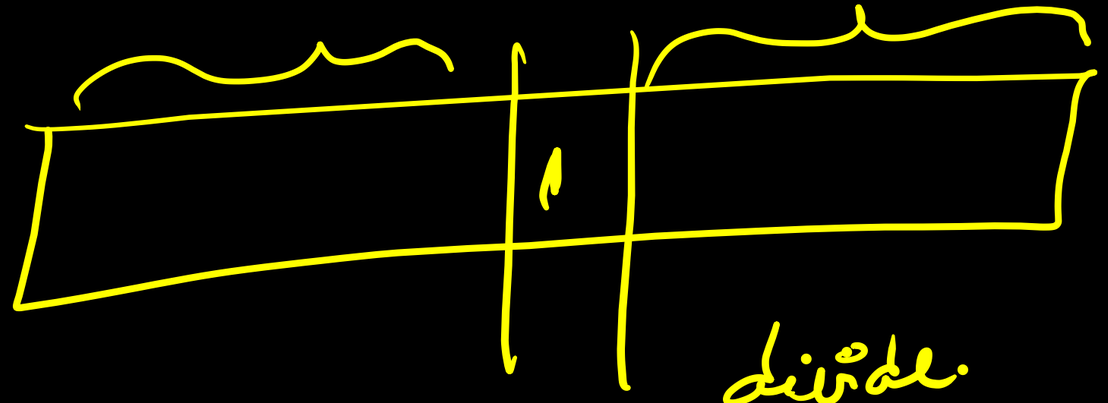



need not be sorted
≤
>
need not be sorted
right
bottom
last
element.
(can be
element)

Divide: Partition (rearrange) the array $A[p..r]$ into two (possible empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that each element of $A[p..q-1]$ is less than or equal to $A[q]$, which is, in turn less than or ~~equal to~~ each element of $A[q+1..r]$. Compute the index q as a part of the partitioning algo.

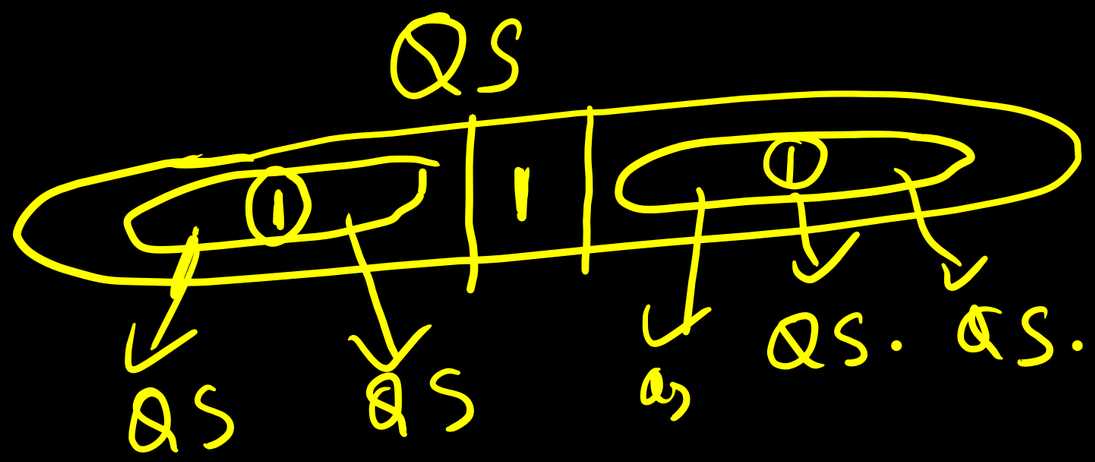


Divide



divide.

Conquer :- Sort the two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ by recursively calling QS.



Combine: Since the subarrays are sorted in place, no need to combine them

D — Divide
A
C — Conquer
~~C — Combine~~
C — Combine (Something)

QS in in place

MS was out of place

MS

A

O(n)

L R

A

extra space

MS

define

{

MS

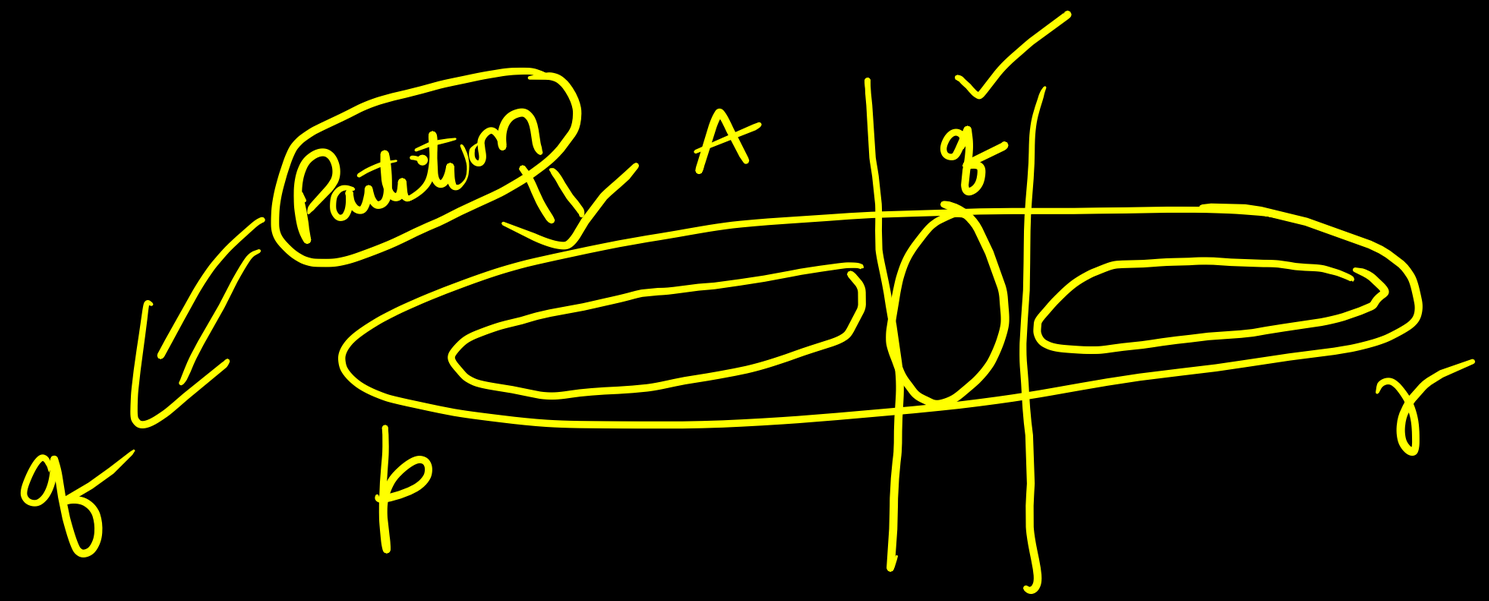
MS

Conquer

merge.

Combine

y



partition is the heart of quick sort.

$A[p \text{ to } q-1] \leq A[q]$

$A[q+1 \text{ to } r] > A[q]$

Quick Sort (A, p, r)

{ if (p < r) → two elements

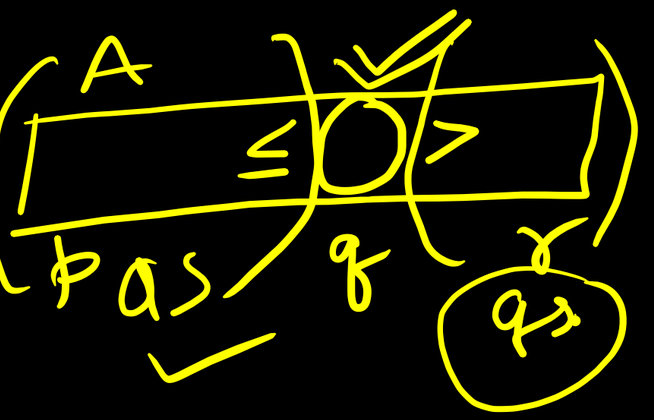
{ q = partition (A, p, r)

Quick sort (A, p, q-1)

Quick sort (A, q+1, r)

}

}



(q) ⇒ inserted and so not included in recursive calls again.

Partition is putting one element in its place one at a time.

Partition (A, p, r)

{

→ x = A[r]

→ i = p-1

→ for (j = p to r-1):

{

→ if (A[j] ≤ x)

{

→ i = i+1

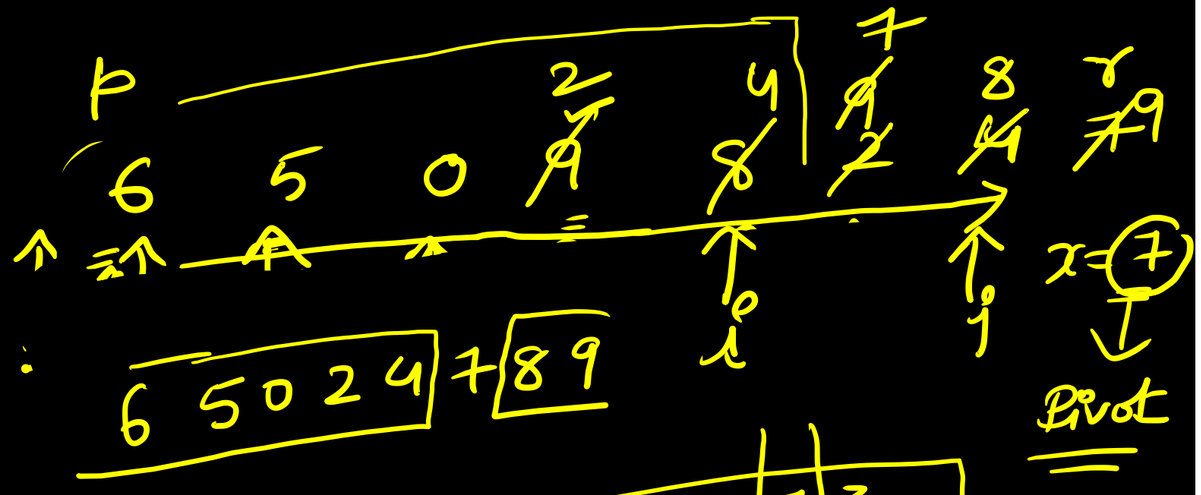
exchange A[i] with A[j]

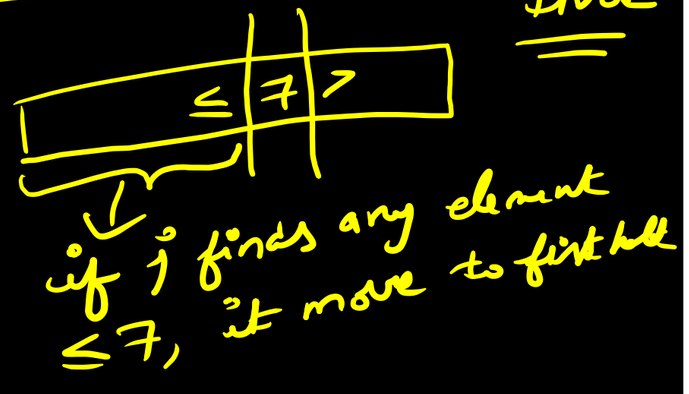
}

}

→ exchange (A[i+1] with A[r])

→ return i+1





p r

9 6 5 0 8 2 4 7 x=7

0 i

p

-26 8 89 26 8 7 8 89

i i j j

$A[i+1] \Leftrightarrow A[i]$

i++

A[i] A[j]

-2 1 4 6 8 7 8 89

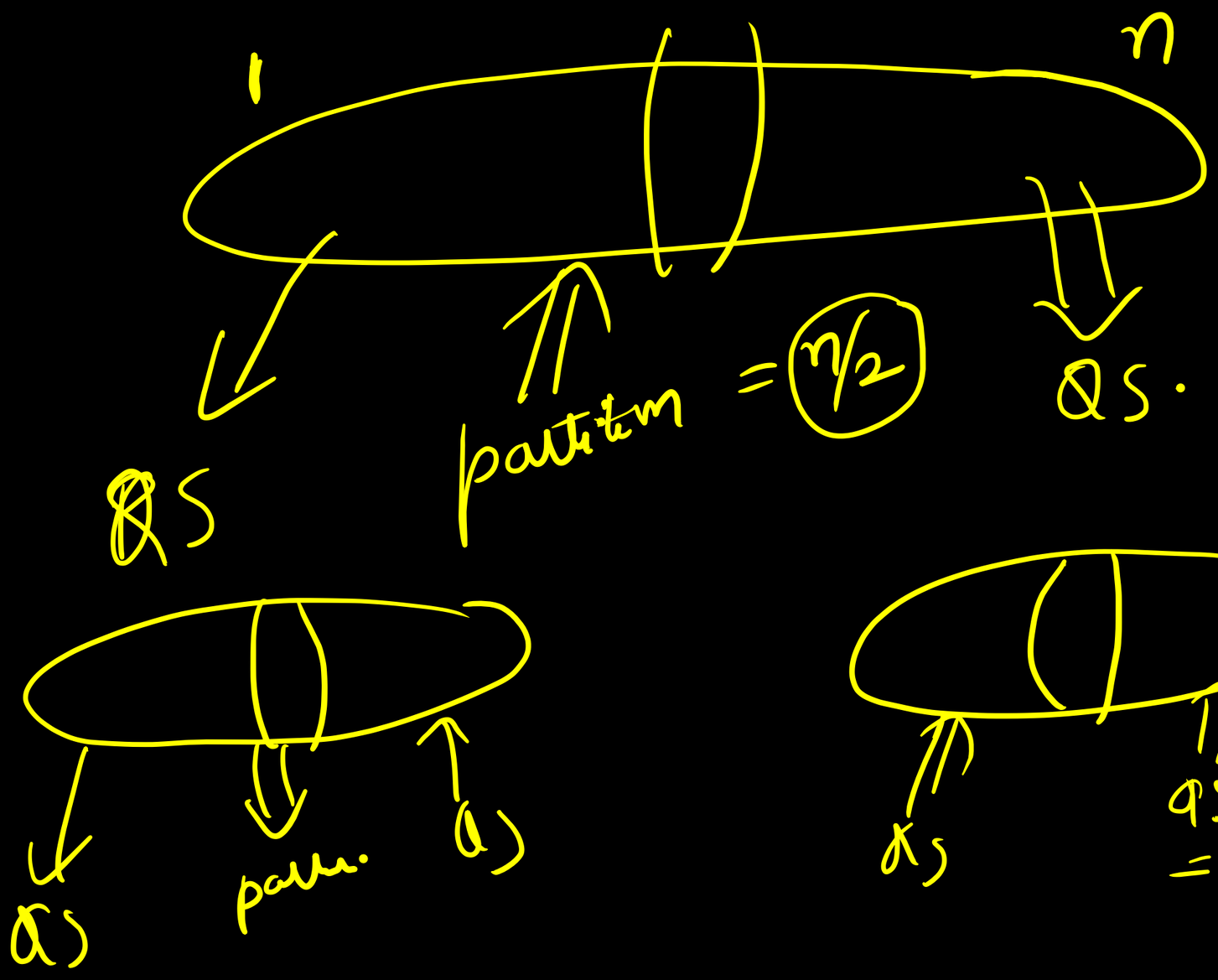
<=

>

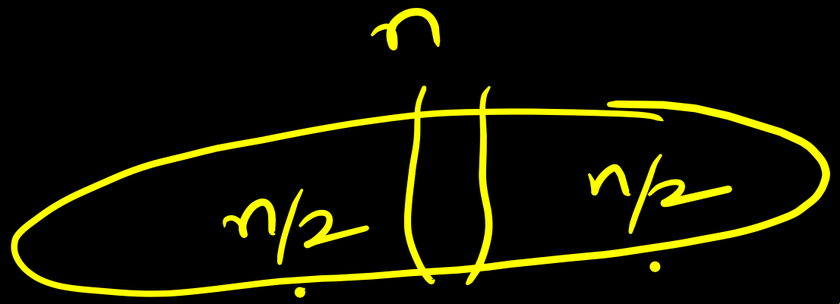
x = a[r]

= 4

r-1


1
n
partition = n/2
QS.
QS
QS
part.
QS
QS.
=

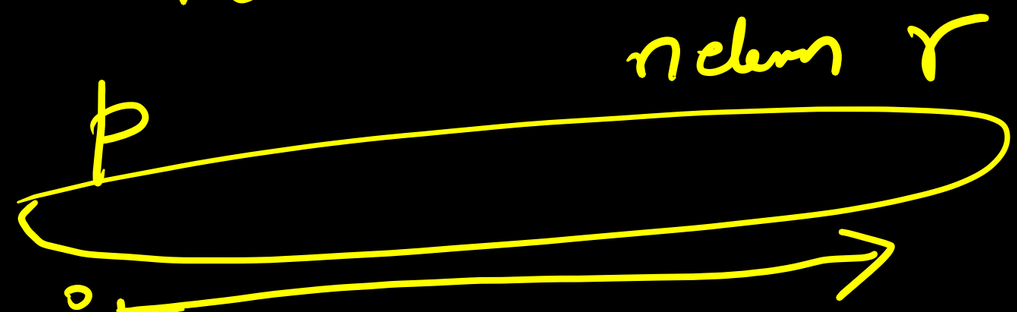
Analysis of QS :-



BC =

TC = for partition.

```
QS(n)
{
    partition → O(n)
    QS - n/2
    QS - n/2
}
```



Best Case

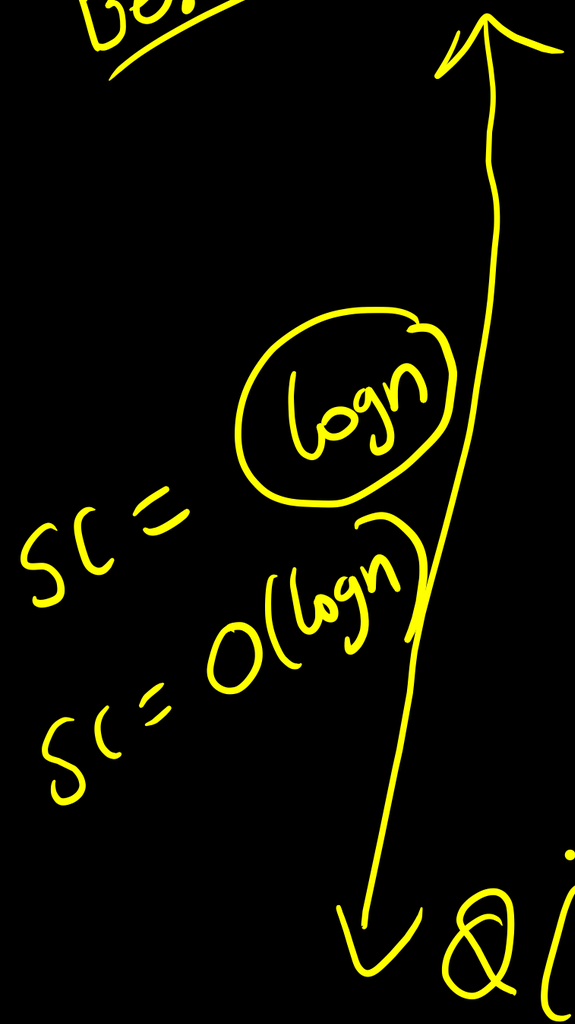
```
QS(n) => T(n)
{
    partition - O(n);
    QS(n/2) => T(n/2)
    QS(n/2) => T(n/2)
}
```

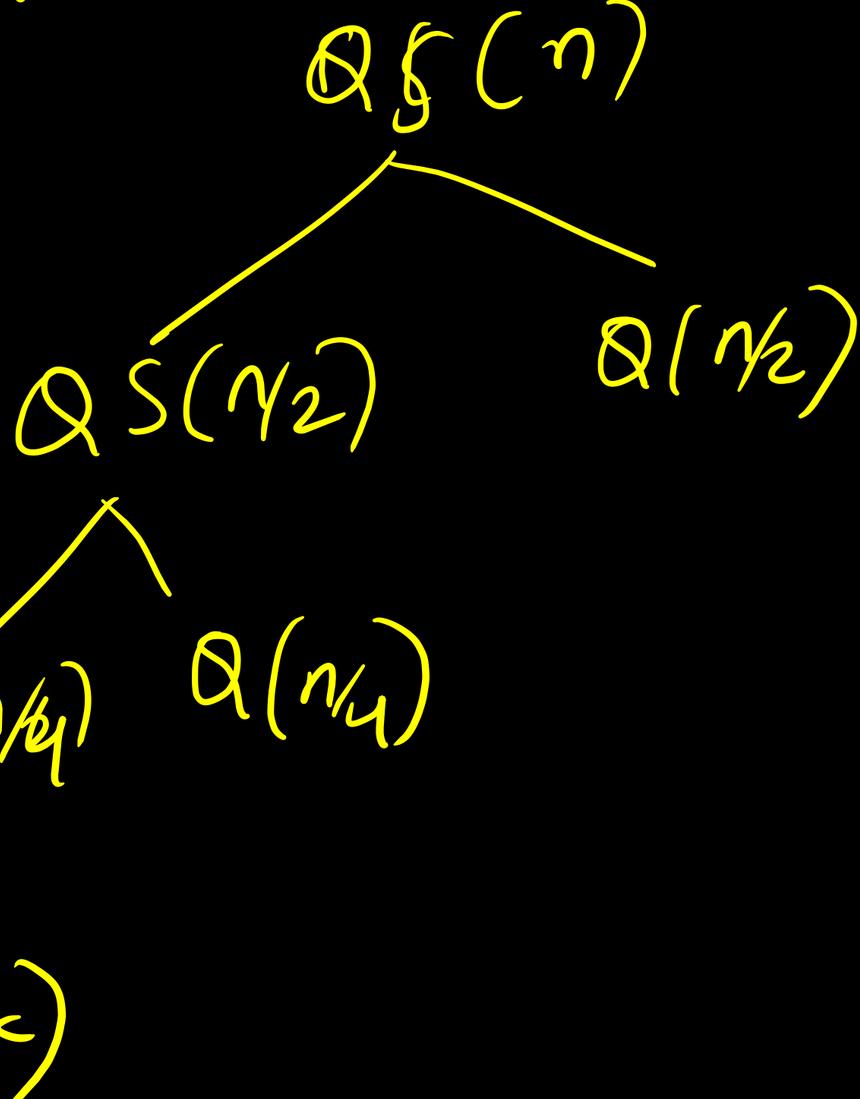
$T(n) = 2T(n/2) + cn$

$= O(n \log n)$.

Best Case:

SC:





BC TC: O(nlogn)

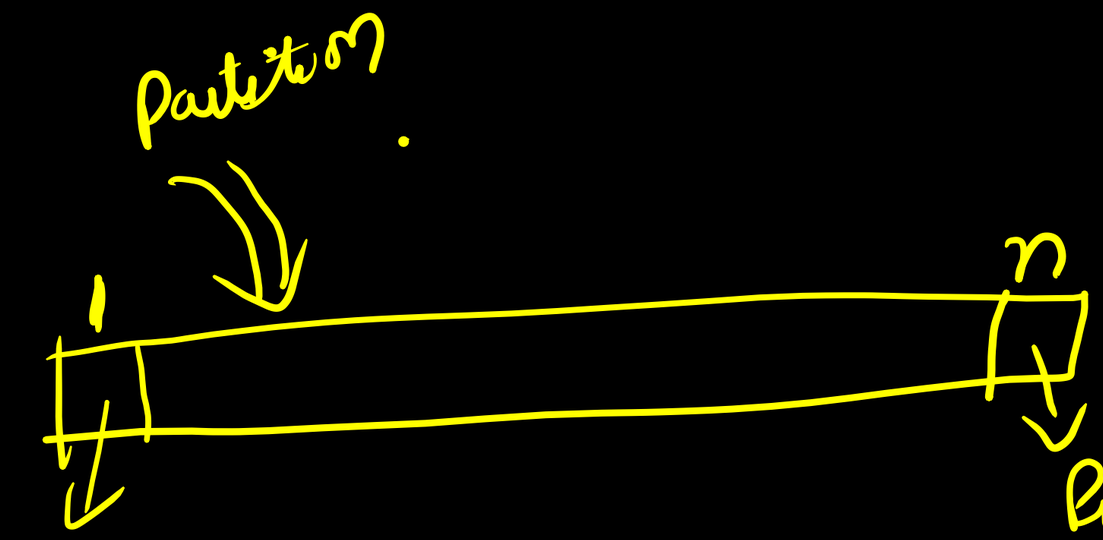
SC: O(logn).

$n/2^k = 1$

$n = 2^k$

$k = \log n$.

Worst Case

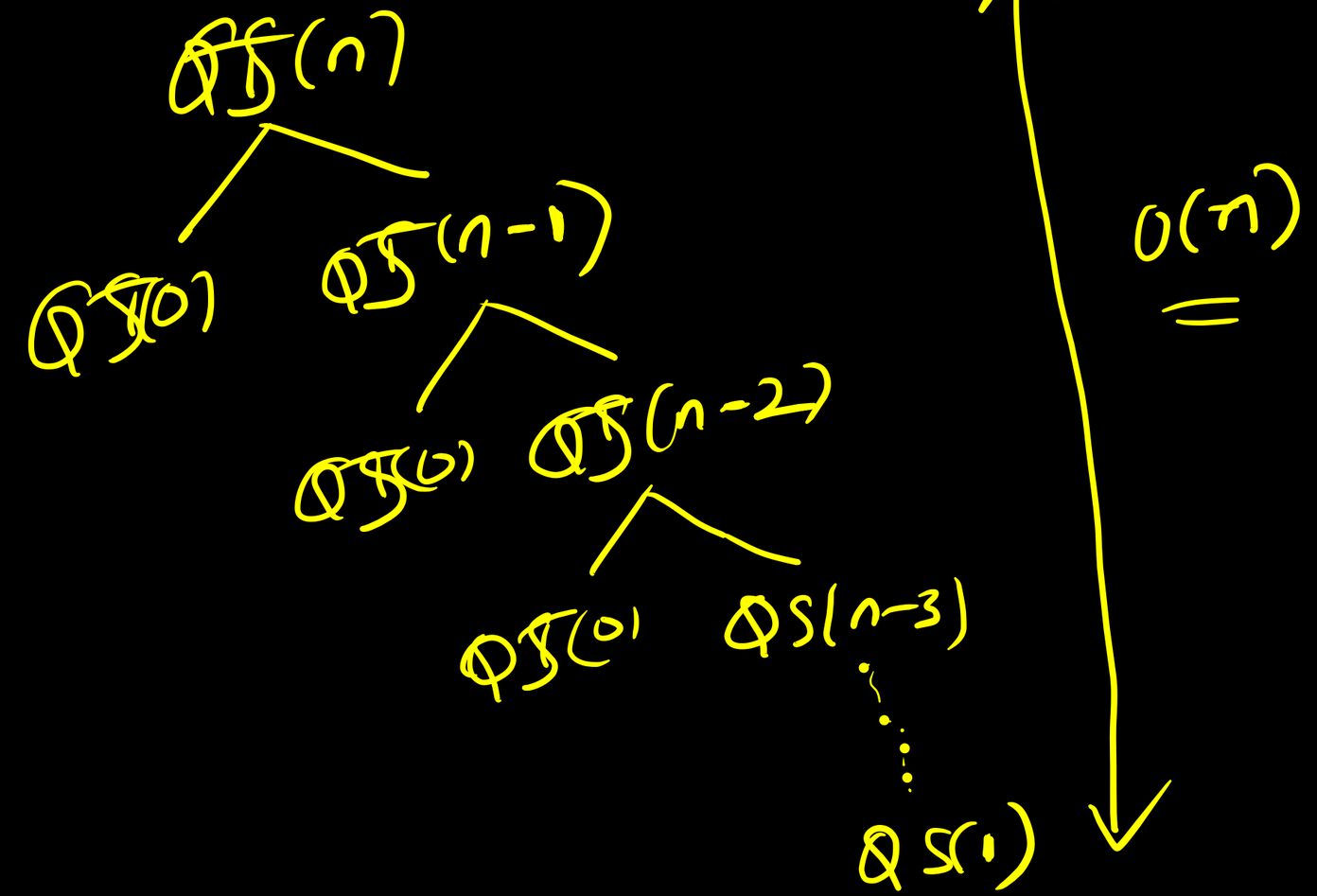


$$T(n) = T(0) + T(n-1) + O(n)$$
$$= T(n-1) + cn$$
$$= O(n^2)$$

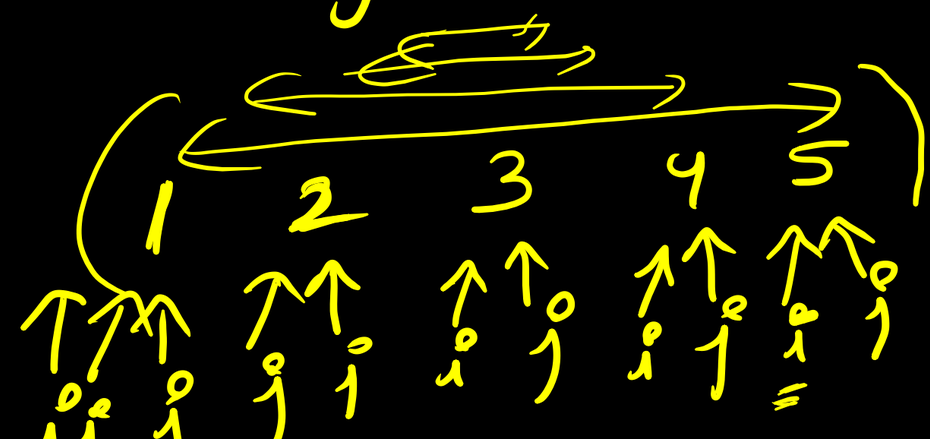
↓ Substitution

Work Done : SC

WC SC = O(n)



what happens if QS is applied on an array sorted in ascending order



$x = 6$

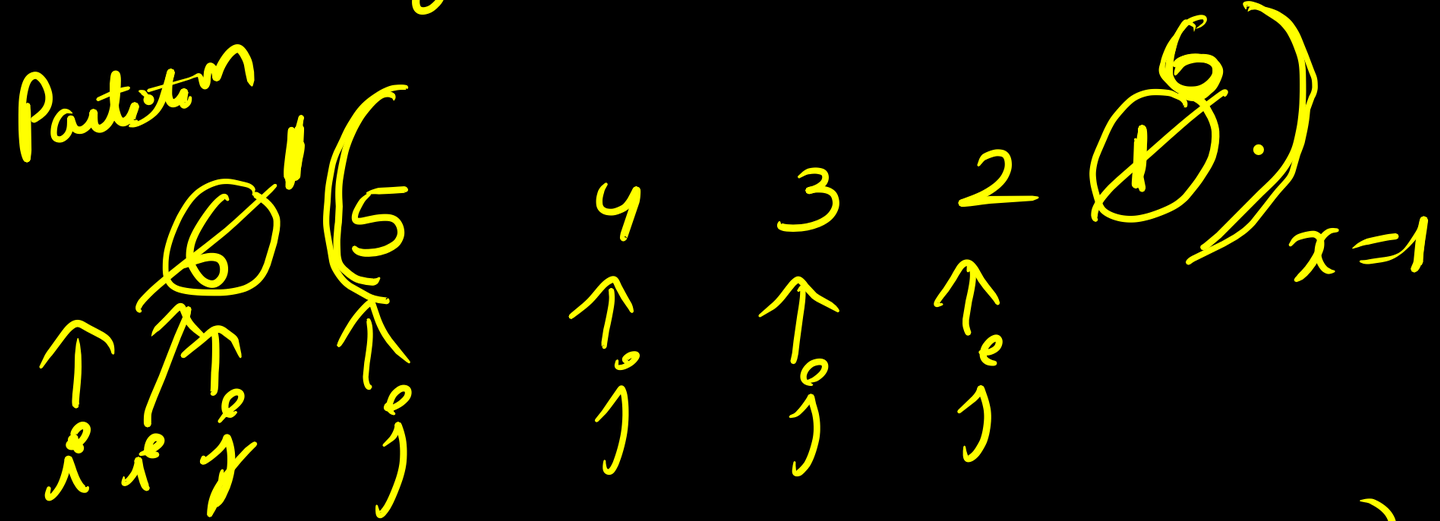
partition.

$T(n) = T(n-1) + T(0) + O(n)$

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

$SC = O(n)$

descending order array:



$$T(n) = T(0) + T(n-1) + O(n)$$

$$= O(n^2)$$

$$SC = O(n).$$

if all elements are same

Partition

(1 5)

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

Let us say each split in 1:9

1:9
∧
1:99
∧
1:999
∧
1:9999

$O(n \log n)$

$T(n) \rightarrow O(n)$

$T(n/10)$ $T(9n/10)$ $\rightarrow O(n)$

$T(n/100)$ $T(\frac{9n}{100})$ $T(\frac{9n}{100})$ $T(\frac{81n}{100})$ $\rightarrow O(n)$

$T(\frac{n}{(\frac{100}{9})^k})$

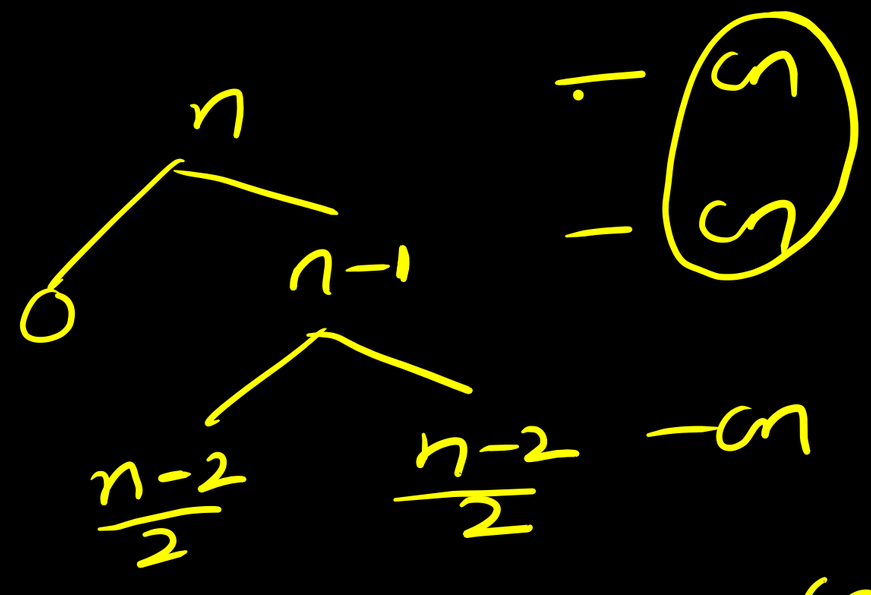
$\frac{n}{(\frac{10}{9})^k} = 1$

$\Rightarrow k = \log_{\frac{10}{9}} n$

$= O(\log n)$

$O(n \log n)$

If alternatively worst and best happened

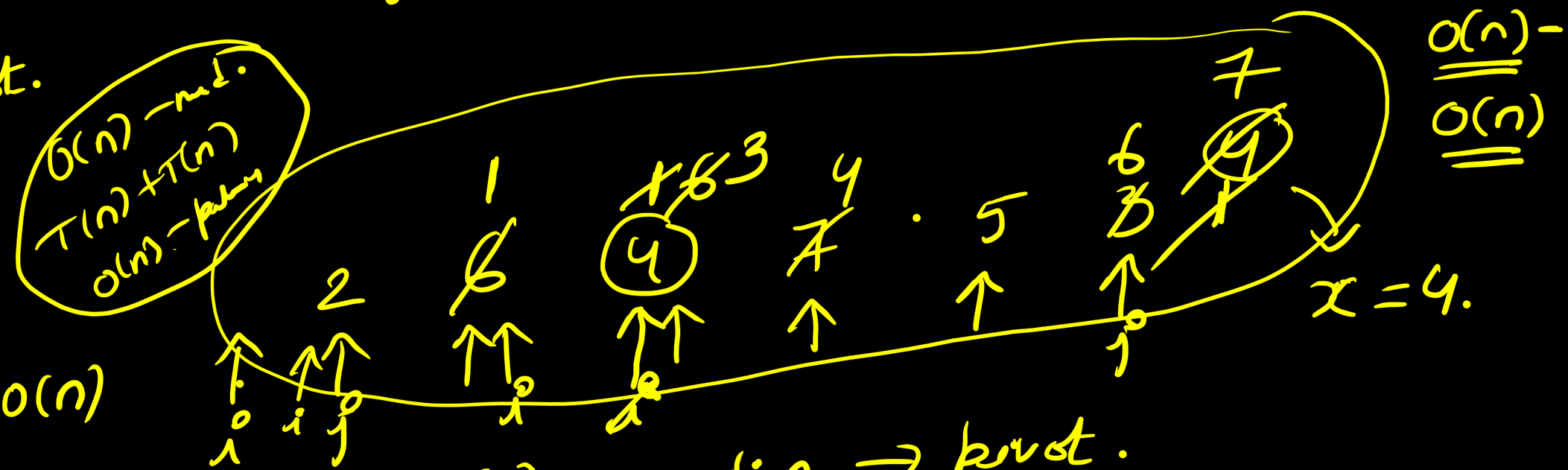


$$T(n) = cn + cn + cn + 2T\left(\frac{n-2}{2}\right)$$

$$\leq 3Cn + 2T\left(\frac{n}{2}\right)$$

$$= O(n \log n) \rightarrow \text{master theorem.}$$

The median of n elements can be found in O(n) time.
what is the TC of QS, in which median is selected as pivot.



$O(n) \rightarrow$ median $\rightarrow$ pivot.

$$T(n) = \begin{cases} T(n/2)+T(n/2) \\ \quad +O(n)+O(n) \\ 2T(n/2)+O(n) \\ = n\log n \end{cases}$$

(2 1 3) (4) (5 6 7)

Gate: In quick sort, if sorting n elements, the $(n/4)^{th}$ smallest element is selected as pivot using $O(n)$ time algorithms what is the worst case time complexity of QS?

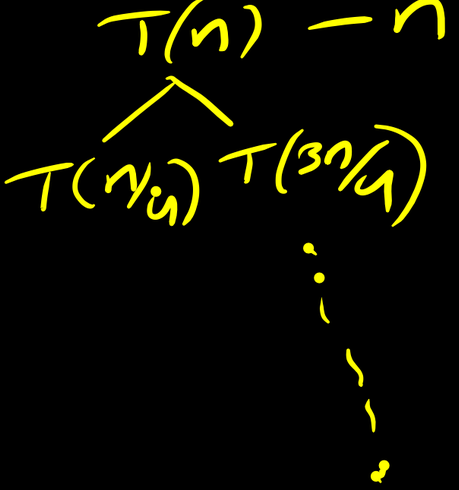
$O(n) \rightarrow$ $n/4^{th}$ element.

$O(n) \rightarrow$ partition

$T(n) = T(n/4) + T(3n/4) + O(n)$

1:3

1:9
1:99
1:999
} $O(n \log n)$



Getting in QS if elements are $1, 2, 3, \ldots n \rightarrow T_1$

$n, n-1, n-2 \ldots 1 \rightarrow T_2$

$T_1 \; ? \; T_2$

$=$

→ In QS by using an $O(n)$ partition algo, we always partition array into one part $1/5\,n$ other part $4n/5$, then what is the TC?

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(4n/5\right) + O(n).$$

$$= O(n \log_{5/4} n) = O(n \log n)$$