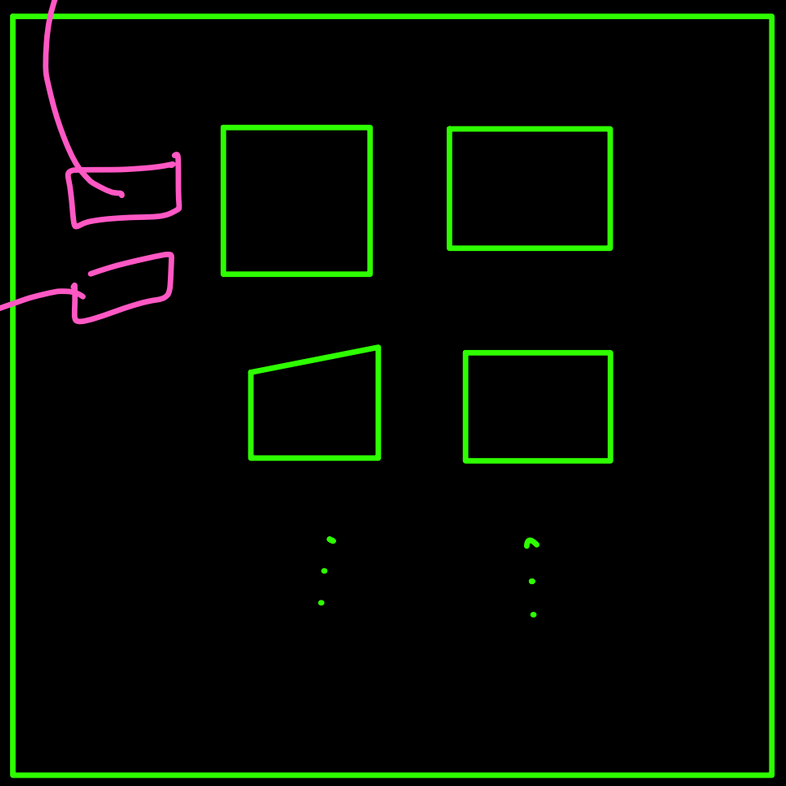
Sorting → generally → inc order

means arranging data in a specific order.

↳ permutations

Sort by price



Sort by ratings

→ list of products on flipkart

Worst possible Sorting algo .....

→ 5, 4, 3, 2, 1 → arrange this data in inc order.

↓

I generate all possible permutations of the data:

↳ 4 5 3 2 1

4 3 5 2 1

2 5 1 2 3

⋮

1 2 3 4 5

↳ n! permutations in worst case.

K amount this to generate one perm.

Ex → 3, 2, 1

$O(K \times n!) \approx O(n!)$

3 2 1

3 1 2

2 3 1

2 1 3

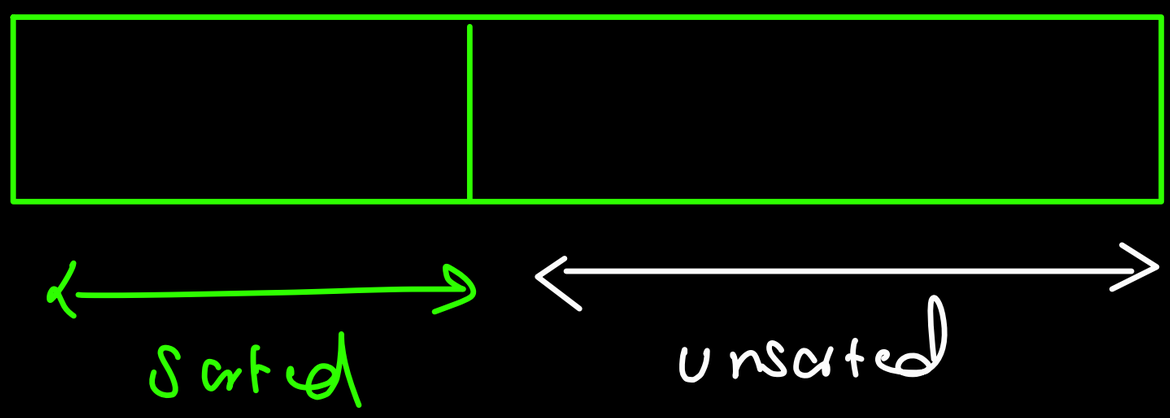
1 2 3 ——→ inc order

1 3 2

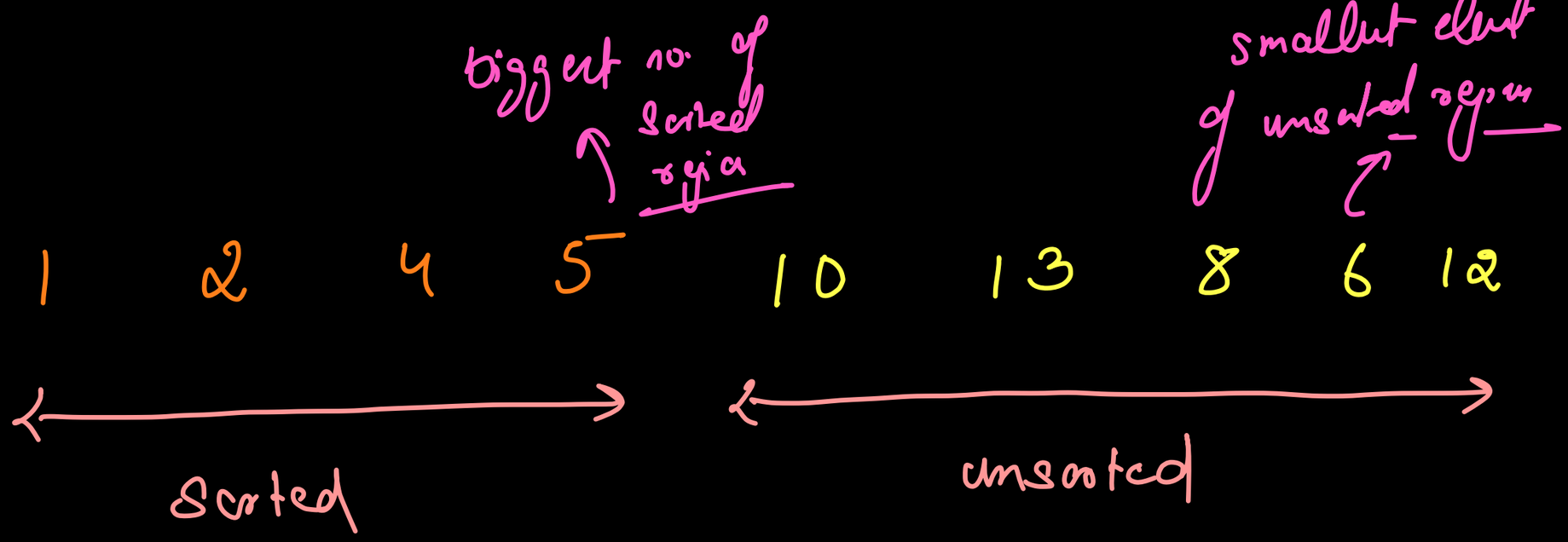
Selection Sort

Consider the following scenario,

↳ 1) we have a collection of no's such that the first part is sorted in inc order, & later part is unsorted



↳ 2) the biggest no of the sorted region is smaller than smallest no. of unsorted region.



$5 < 6$

⇒ figure out how can we expand the sorted region

& storing the unsorted ??

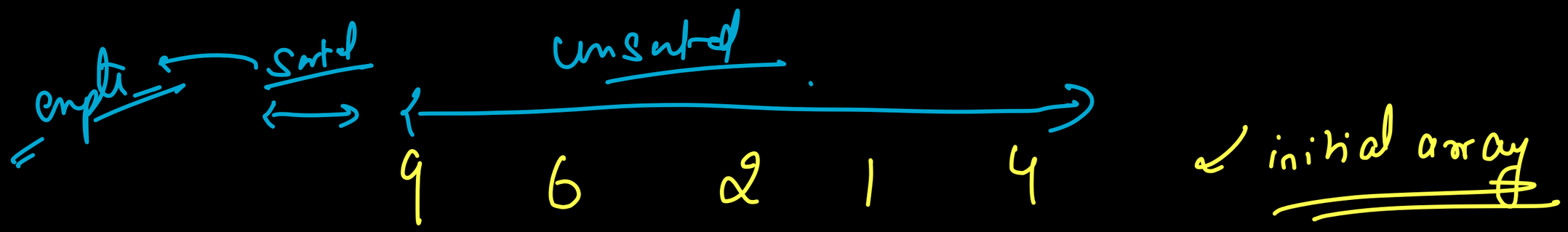
→ how about we find the smallest no of unsorted region, and put it after the biggest no. of sorted region.

① find min element: → we know how to do it

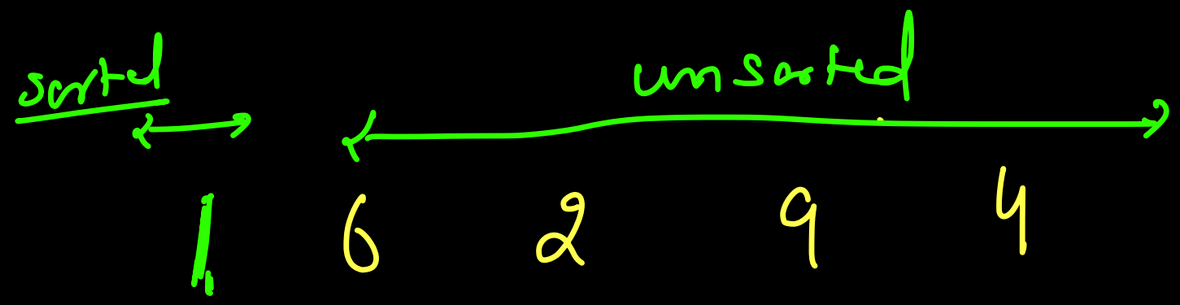
↳ linear search

0	1	2	3	4
9	6	3	1	2

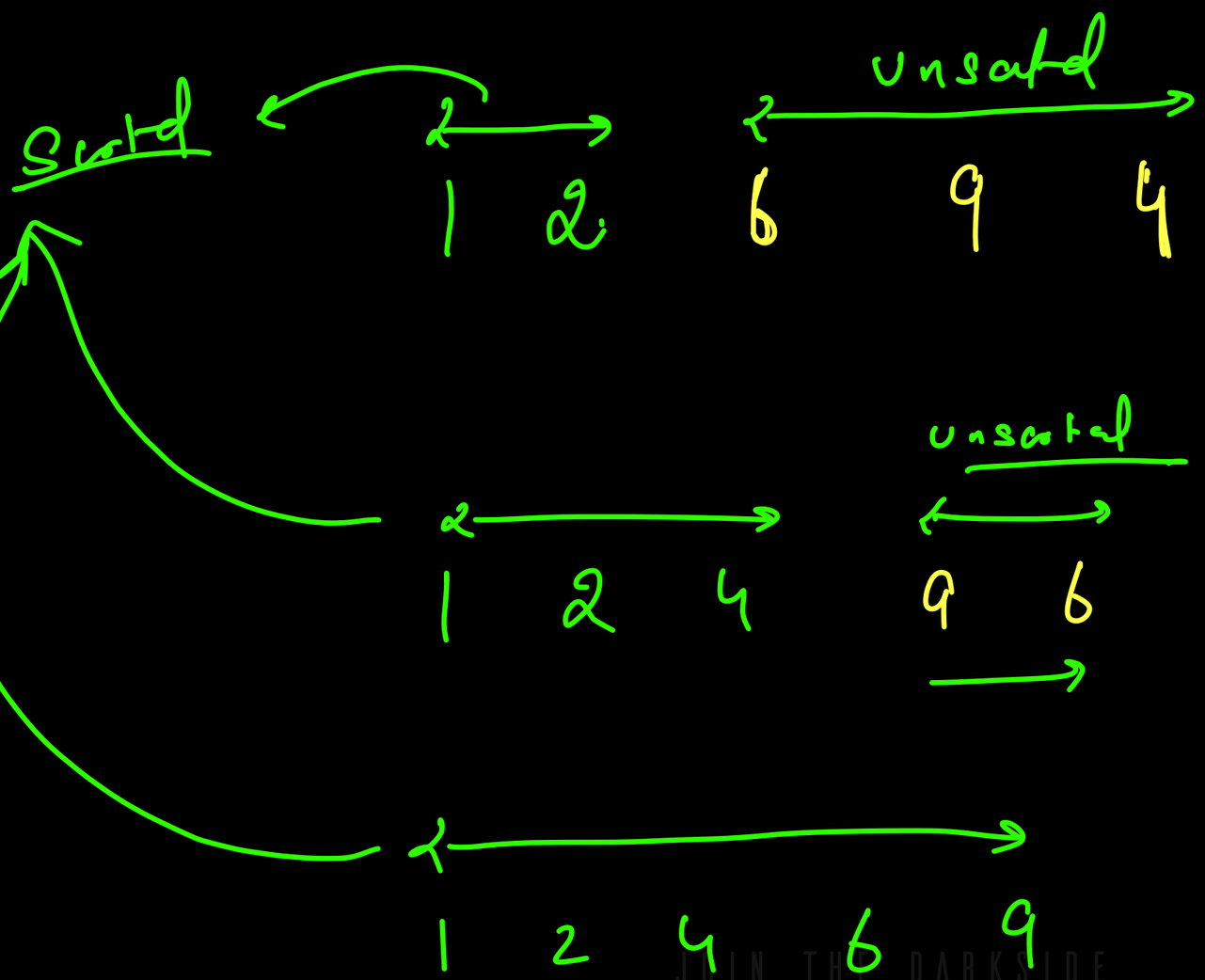
min_el_idx = 0 ~~1~~ ~~2~~ 3



↳ initial array



↳ 2nd min elmt → 2



find the min elt →

find the min elt → 6

Example ↳

5 4 3 2 1 (n = s)

1st iteration → n iteration to find smallest no.

2nd " → (n-1) " " " " " "

3rd " → n-2 " " " " " "

⋮

n + n-1 + n-2 + n-3 2

↓

1

Sum of first n natural nos :- — 1

$$\frac{n\ (n+1)}{2} \quad \text{— 2}$$

$$\frac{n\ (n+1)}{2}$$

$$n\ (n+1)$$

$$n^2 + n$$

$$n^2$$

$$\longrightarrow \quad O(n^2)$$

what if array is sorted ?

1 2 3 4 5

$\Omega(n^2)$

$\Theta(n^2)$

Space $O(1)$

In the worst case how many swaps you do in selection sort??

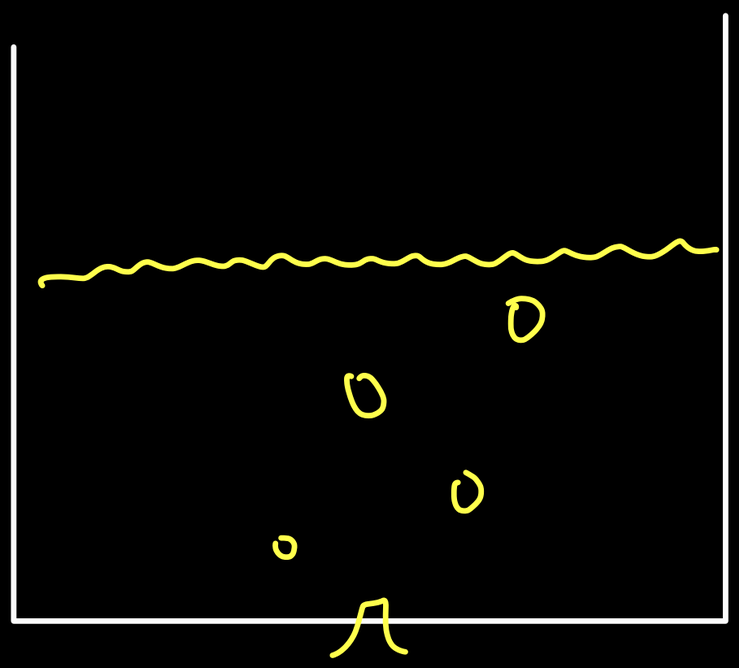
↳ n-1 swaps

how many comp you do in selection sort ?

↳ approx → n^2

uses → for sorting data when writing is a heavy operation

ex → sorting data in hard disk, we use selection sort



Bubble Sort

→ In every iteration push the biggest bubble on the top element.

→ We do pairwise comp. Whoever is bigger is shifted to the right.

3 2 1 4

$i=0 \rightarrow 0^{th}$ item

$i=1 \rightarrow 1^{st}$ item

```
for ( i=0 ; i < n ; i++) {
    is_swapped = false
    for ( j=0 ; j < n-i-1 ; j++) {
        if (a[j] > a[j+1]) {
            swap (a[j], a[j+1])
            is_swapped = true
        }
    }
    if (is_swapped == false) return ;
}
```

Space
=
O(1)

$i = 0$
↓
0 th iter

1 2 3 4 5

↑ j

$\Omega(n)$

if in the loop of j, no swapping occured, the array is sorted.

1 2 3 4 5

n-1 sups

n-2 sups

n-3 sups

n-4 sups

⋮

$(n-1) + (n-2) + (n-3) \quad \cdots\cdots$ (

↳ Sum of n-1 natural no.

$$\frac{n(n-1)}{2} \longrightarrow n(n-1) \longrightarrow n^2 - n \approx O(n^2)$$

$O(n^2)$

bubble sort is a swap heavy algo.

3 2 1 4

when we know for a good no. of time input will always be sorted, bubble sort is the way to go.

Bubble sort is capable of giving the k^{th} largest element.

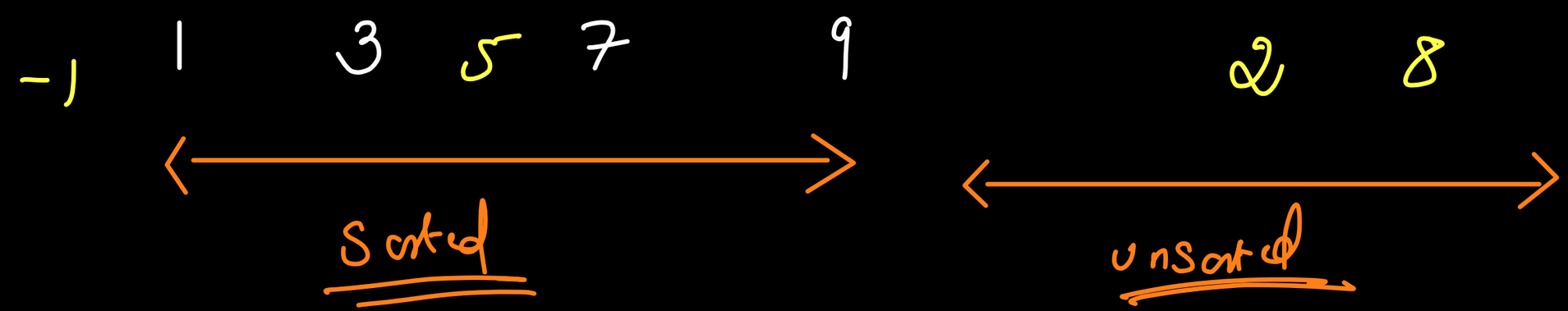
$K = 2^{n-1}$

largest

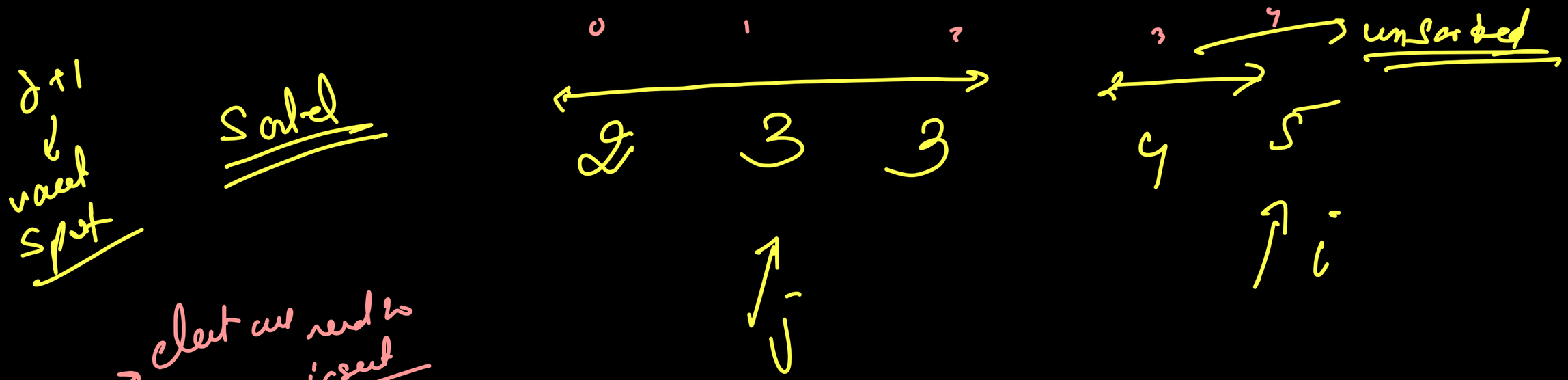
deal

3 2 1 4 5

Insertion Sort



In insertion sort, you one by one pick an elet from unsorted region & insert it to the sorted part.




```
for ( i = 1 ; i < n ; i++ ) {
    element = arr[i];
    for ( j = i-1 ; j >= 0 ; j-- ) {
        if (arr[j] > client) {
            arr[j+1] = arr[j]      → shifting j to right
            break;
        }
    }
}
```

arr[j+1] = elem

3

3 4 5 2 1

1 + 2 + 3 ..... n-1

$\frac{(n)(n-1)}{2}$ → $O(n^2)$

Interview Tip → DSA round → Qun → code → T.S

write a runnable code

↳ test cases to run this code

So that we can test the logic rigorously