Sorting

↳ Permutations → arrangements

↳ arranging data in a particular permutation (this depends on requirement) is called as **Sorting**

Q ⇒ Given n integer values, arrange them in increasing order.

Ex → $[15, -1, 3, 8, 2, 6]$ ← array
n → $n!$

Brute force → we can generate all possible permutations and filter out your reqd ans.

$[3, 2, 1]$ →

$$\left\{\begin{matrix} 3 & 2 & 1 \\ 3 & 1 & 2 \\ 2 & 1 & 3 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \end{matrix}\right\}$$

$O(K)$ → to generate one perm

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

↳ Bubble Sort

↳ Selection Sort

↳ Insertion sort

↳ Counting sort

↳ merge sort

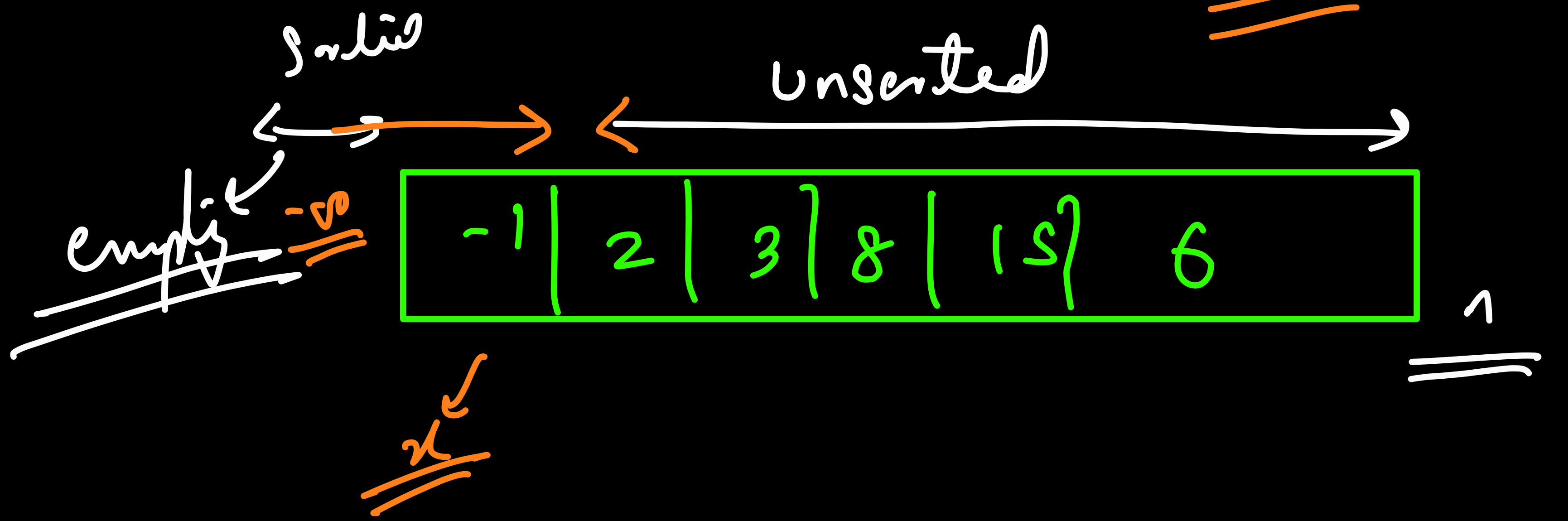
↳ quick sort

Selection Sort

Q ⇒ Given n integer values, arrange them in increasing order.

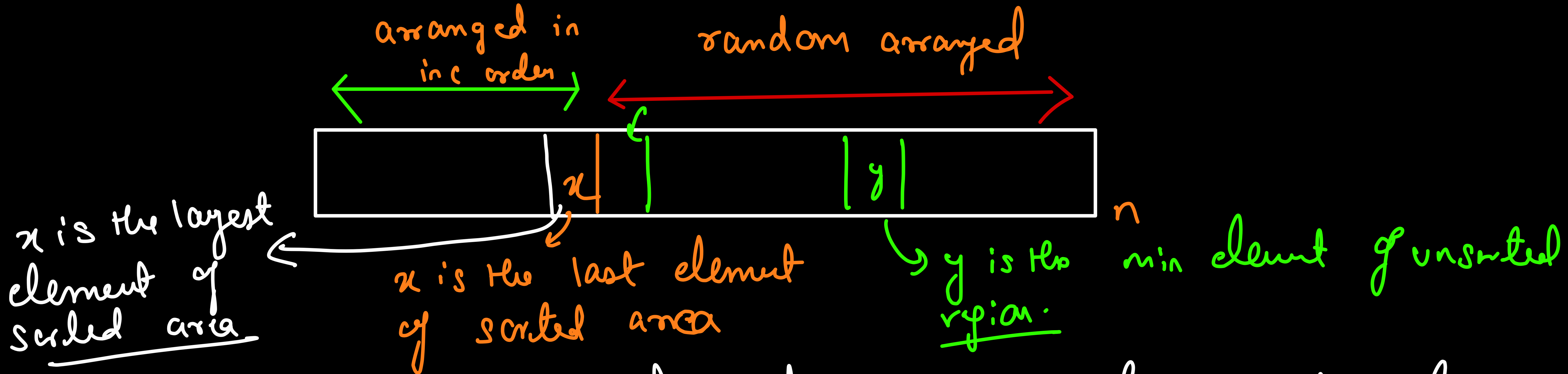
Ex → [15, -1, 3, 8, 2, 6] ← array

n



-1	2	3	8	15	6

In the unsorted region how to get min element. ?!

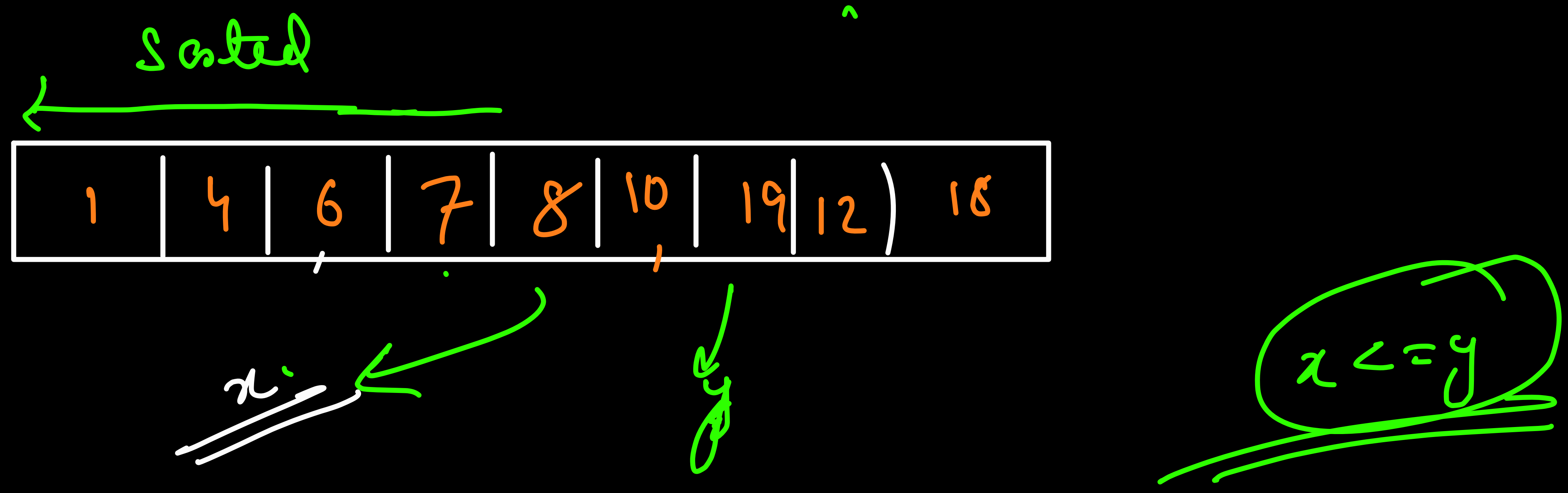

arranged in inc order
random arranged
x
y
n
x is the largest element of sorted area
x is the last element of sorted area
y is the min element of unsorted region.

condition1 → Array is divided into 2 parts where first half is perfectly sorted & the second half is unsorted.

condition2 → the last element of sorted region (x) is less than or equal to the minimum element of the unsorted region.

$$\boxed{x <= y}$$

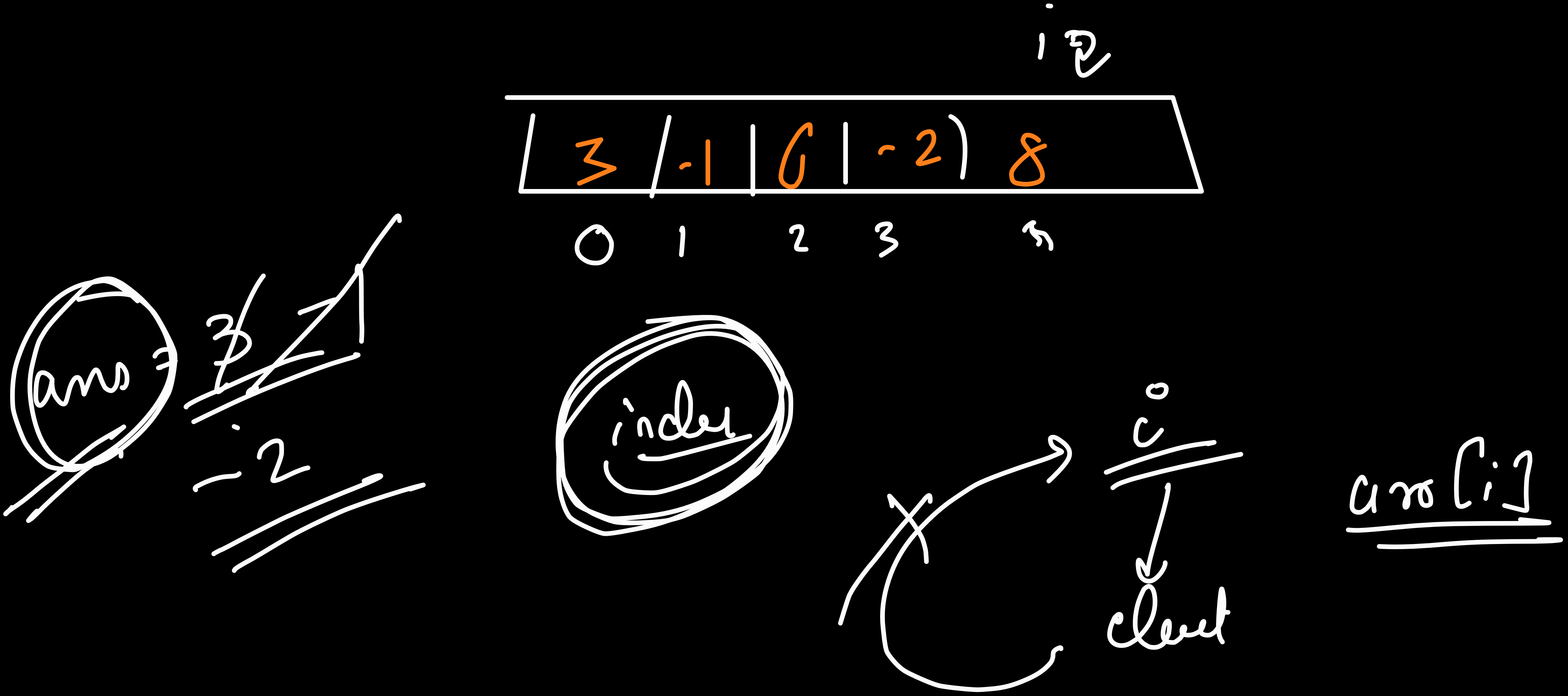
Q → how can we expand the sorted region.



1	4	6	7	8	10	19	12)	18

⟹ if $x <= y$ and x is the largest element of Sorted region & y is the Smallest element of unsorted region & $x <= y$.

Then we can arrange y just after x.


i
3
-1
6
-2
8
0
1
2
3
4
ans = 3 / 1
-2
index
i
check
arr[i]

🟥 → unsorted

🟩 → sorted

i = 0

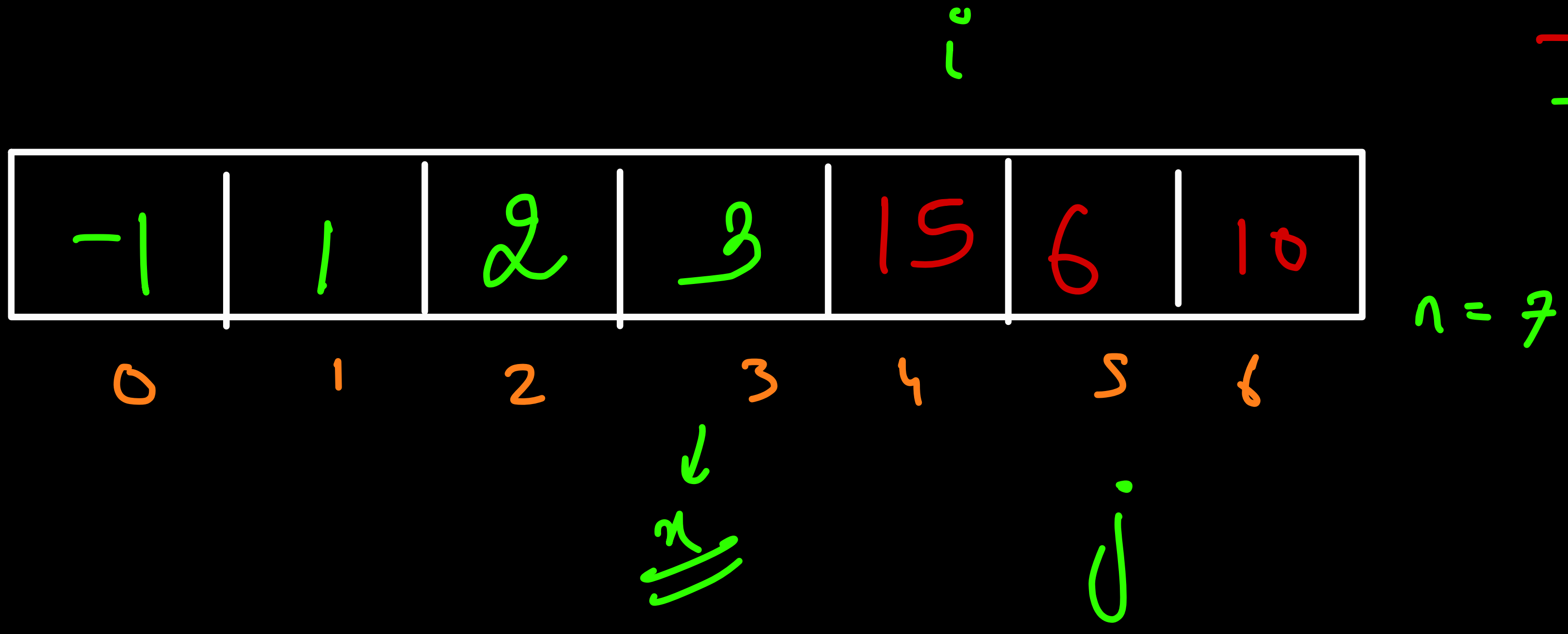
↓

[i, n-1]

↓

unsorted region

< i → sorted



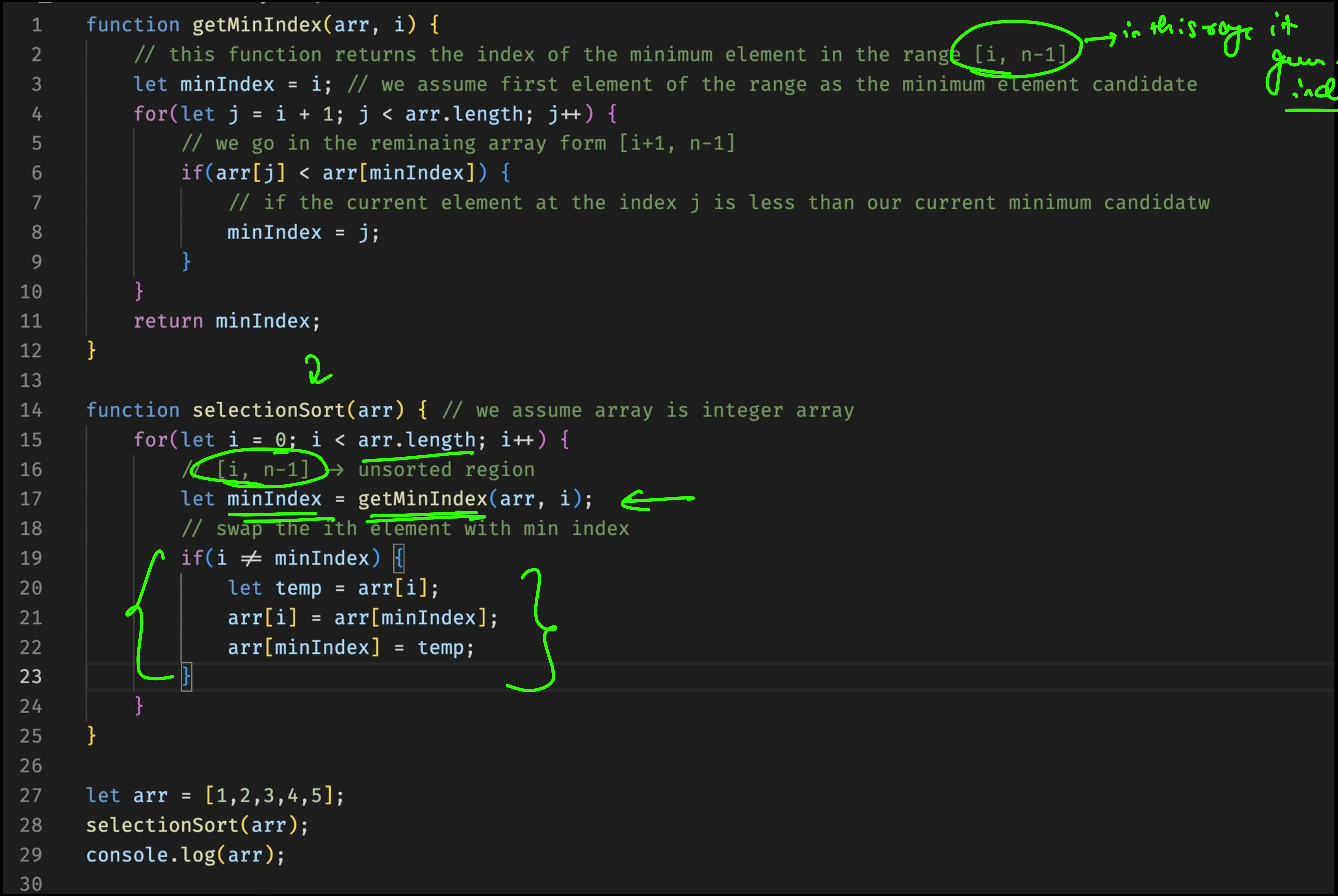
0	1	2	3	4	5	6
-1	1	2	3	15	6	10

min_index = 4

Swap → $\boxed{\text{min\_index} , i}$

j ∈ [i+1, n-1]

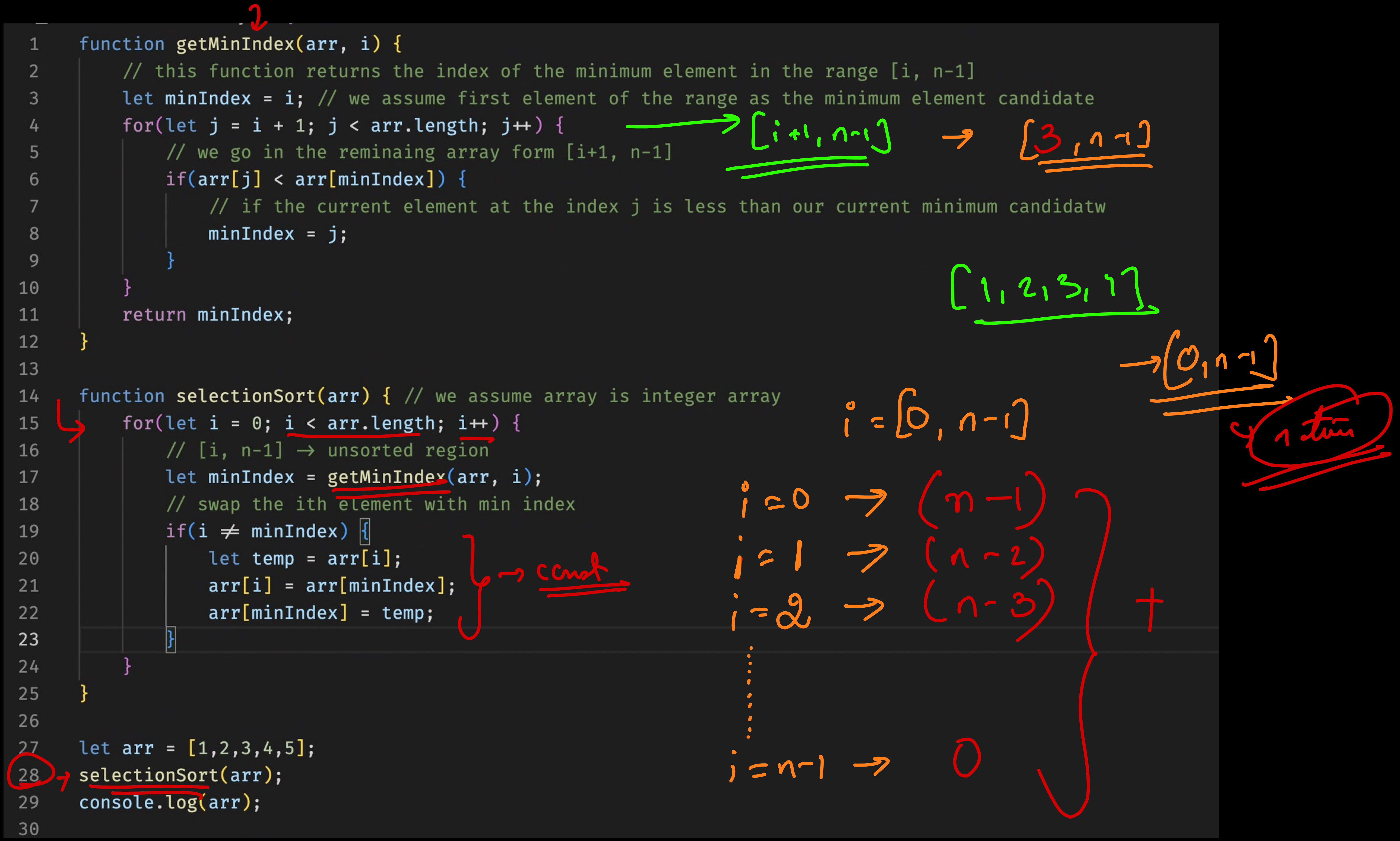
↳ thus j goes from i to n-1 & get the min, In every iteration, we will scan the whole unsorted region & get the index of min element



```javascript
function getMinIndex(arr, i) {
    // this function returns the index of the minimum element in the range [i, n-1]
    let minIndex = i; // we assume first element of the range as the minimum element candidate
    for(let j = i + 1; j < arr.length; j++) {
        // we go in the reminaing array form [i+1, n-1]
        if(arr[j] < arr[minIndex]) {
            // if the current element at the index j is less than our current minimum candidatw
            minIndex = j;
        }
    }
    return minIndex;
}

function selectionSort(arr) { // we assume array is integer array
    for(let i = 0; i < arr.length; i++) {
        // [i, n-1] -> unsorted region
        let minIndex = getMinIndex(arr, i);
        // swap the ith element with min index
        if(i != minIndex) {
            let temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}

let arr = [1,2,3,4,5];
selectionSort(arr);
console.log(arr);
```



```
function getMinIndex(arr, i) {
    // this function returns the index of the minimum element in the range [i, n-1]
    let minIndex = i; // we assume first element of the range as the minimum element candidate
    for(let j = i + 1; j < arr.length; j++) {
        // we go in the reminaing array form [i+1, n-1]
        if(arr[j] < arr[minIndex]) {
            // if the current element at the index j is less than our current minimum candidatw
            minIndex = j;
        }
    }
    return minIndex;
}

function selectionSort(arr) { // we assume array is integer array
    for(let i = 0; i < arr.length; i++) {
        // [i, n-1] -> unsorted region
        let minIndex = getMinIndex(arr, i);
        // swap the ith element with min index
        if(i != minIndex) {
            let temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}

let arr = [1,2,3,4,5];
selectionSort(arr);
console.log(arr);
```

$$(n-1) + (n-2) + (n-3) \; . \; - \; - \; - \; - \; - \; - \; - + 2 + 1$$

$$\Downarrow$$

$$\frac{n(n-1)}{2} \rightarrow \frac{n^2}{\underset{\times}{\boxed{2}}} - \underset{\times}{\boxed{\frac{n}{2}}} \rightarrow \text{lower degree term}$$

$$\underline{\underline{O(n^2)}}$$

Space → $O(1)$

Comparison

is Inplace ??

↓

yes

Worst time → $O(n^2)$

Best time → $O(n^2)$

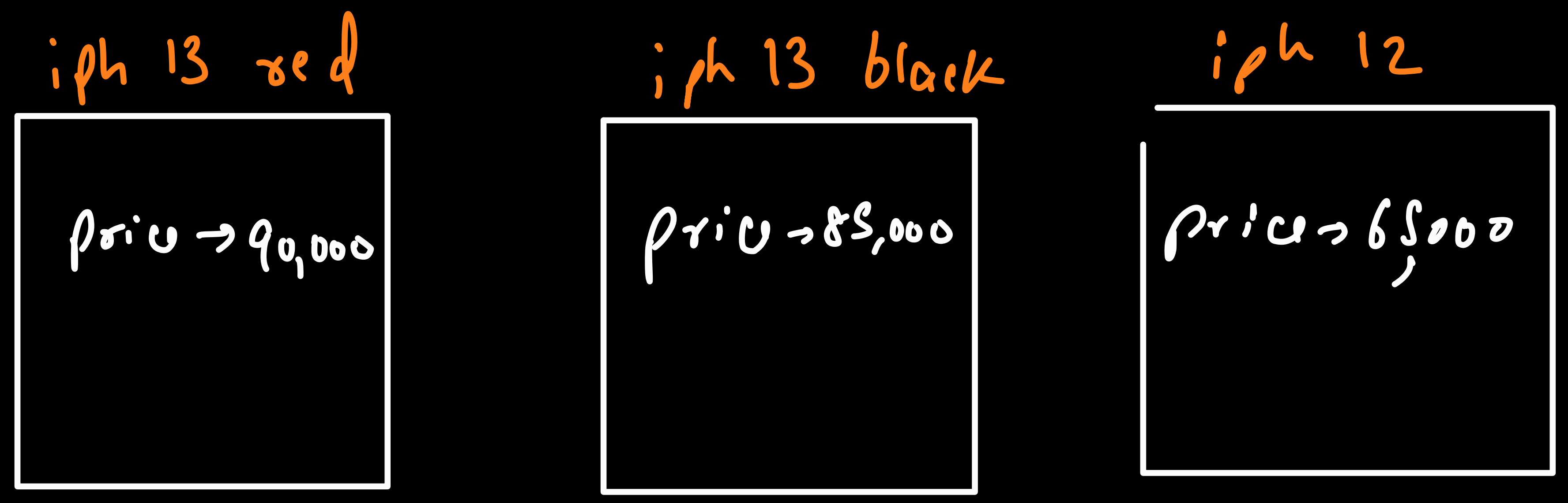
Avg → $O(n^2)$

In selection sort swapping happens → at max approx n times → $O(n)$

Comparisons → $O(n^2)$

Inplace → An algo is considered inplace if they donot depend on extra data structures for modifying data & implementing the algo.

Stability of the sorting algo



iph 13 red

price → 90,000

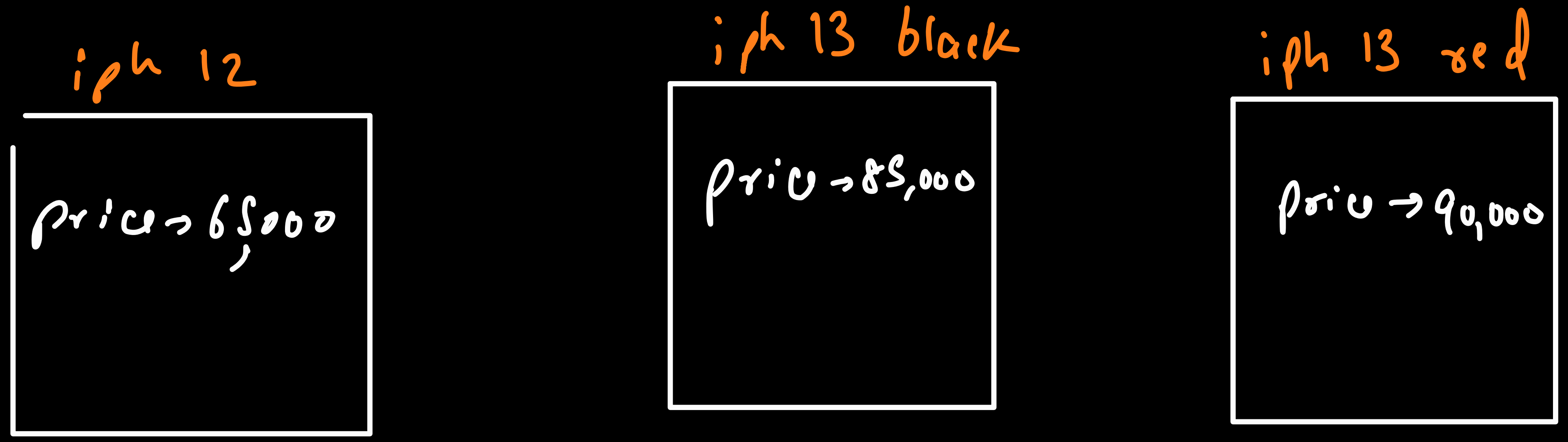
iph 13 black

price → 85,000

iph 12

price → 65,000

→ Sort it based on price in inc order



iph 12

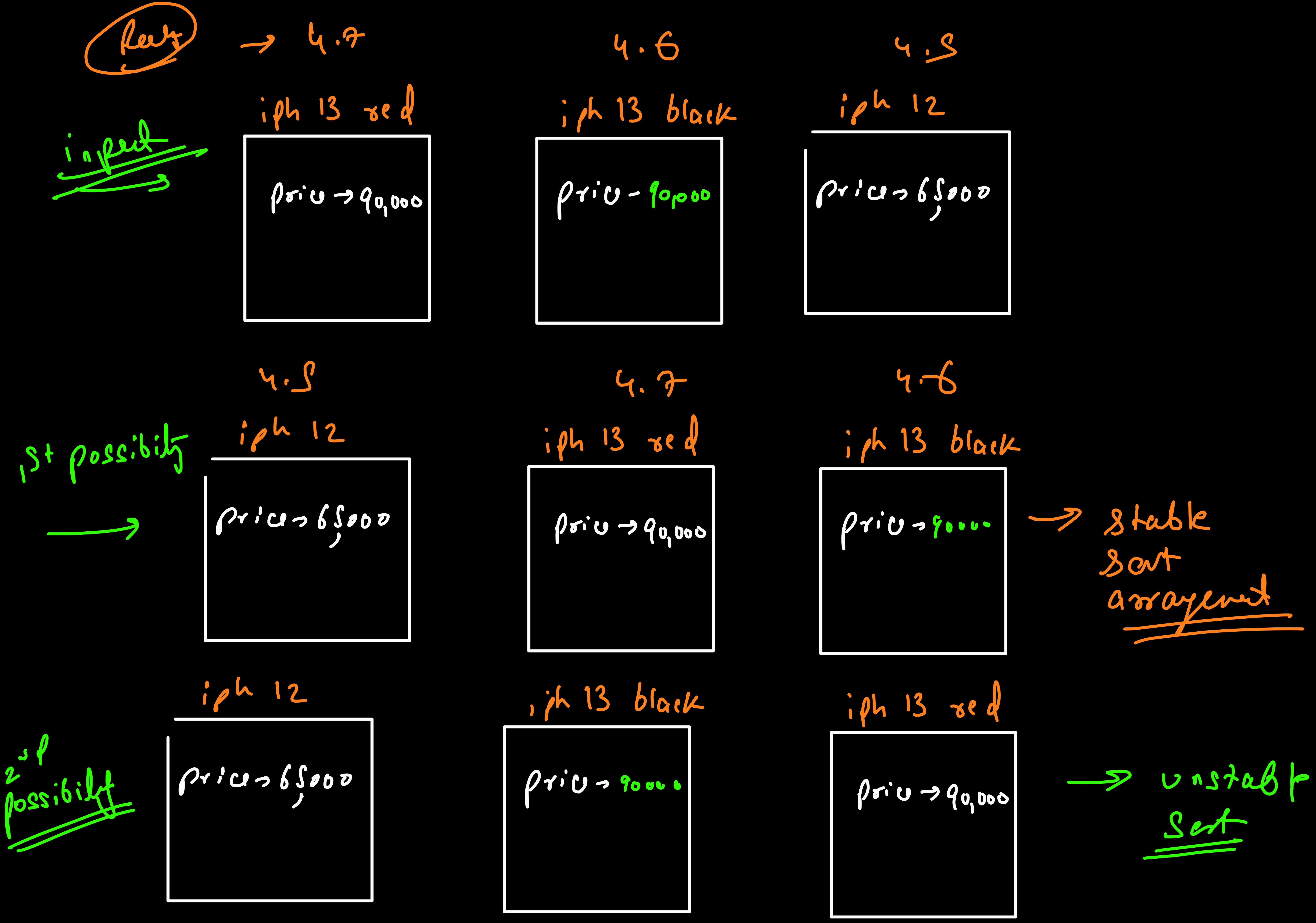
price → 65,000

iph 13 black

price → 85,000

iph 13 red

price → 90,000


Rating → 4.7
4.6
4.5
input
iph 13 red
price → 90,000
iph 13 black
price - 90,000
iph 12
price → 65,000
1st possibility →
4.5
iph 12
price → 65,000
4.7
iph 13 red
price → 90,000
4.6
iph 13 black
price → 90000
→ stable sort arrangement
2nd possibility
iph 12
price → 65,000
iph 13 black
price → 90000
iph 13 red
price → 90,000
→ unstable sort

1) You went to amazon

2) You first sort the products by rating

3) Resorted the products Based on price

Stable sort → after a stable sort the data which has the same value retains the same relative order as it was before sorting

HW → Is selection sort stable ??