Sorting Algorithms:-

Basic:-

1. easy to understand
2. unoptimised

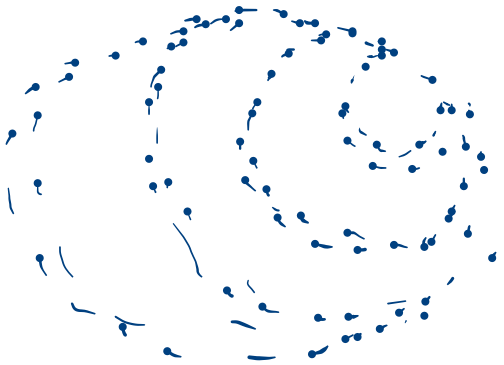
- Bubble Sort
- Selection Sort
- Insertion Sort

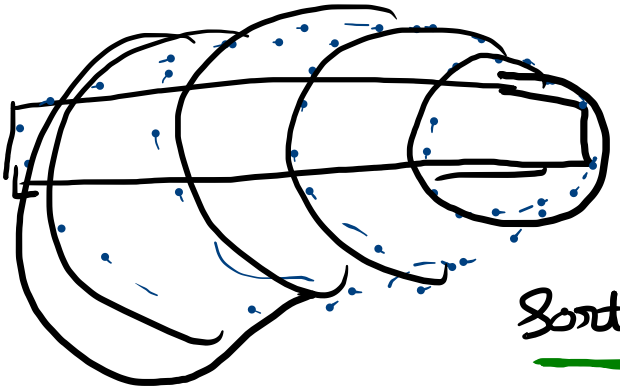
Advanced:-

1. more complex to understand
2. optimised

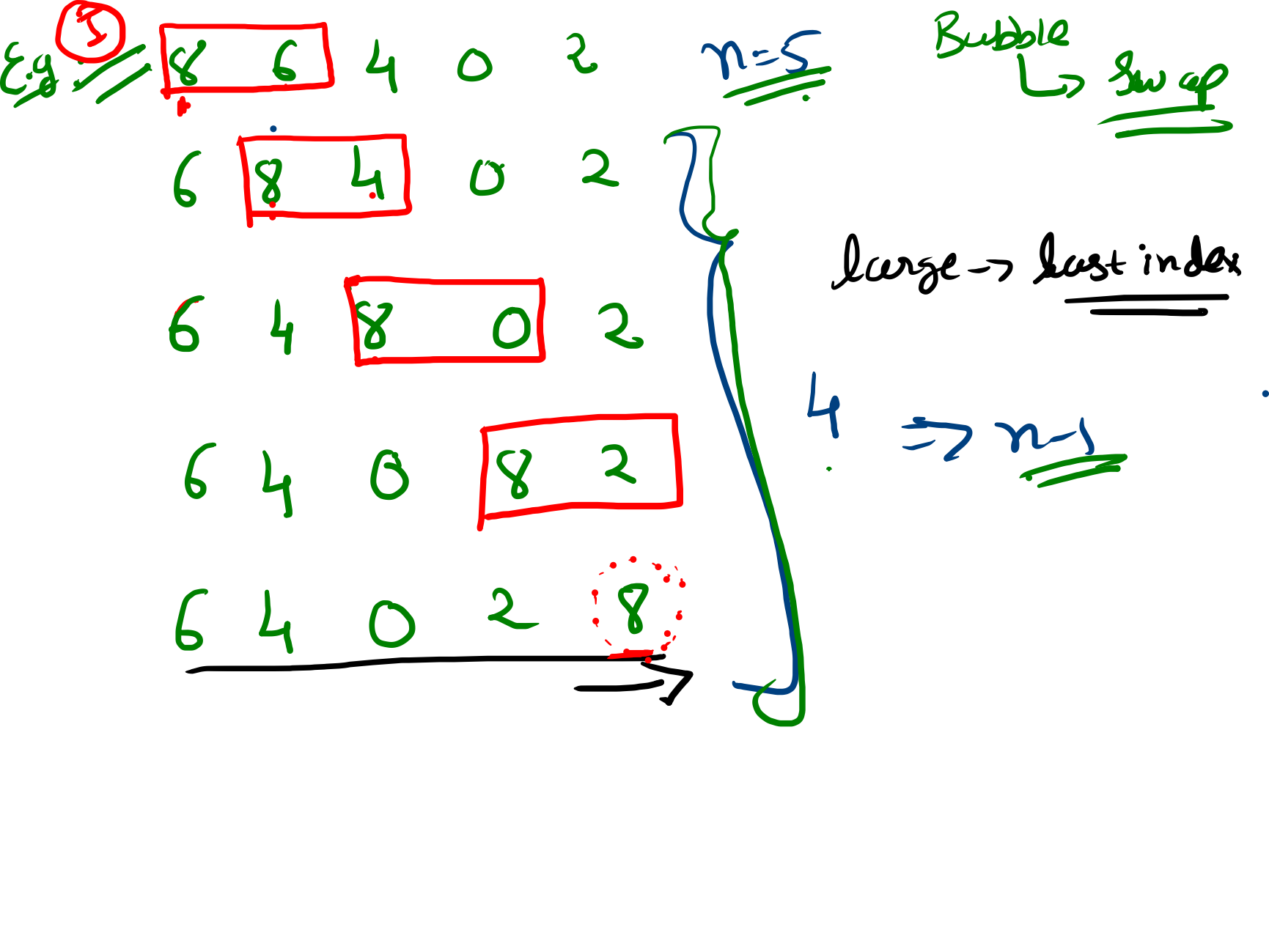
- Merge Sort
- Quick Sort
- Count Sort
- Bucket Sort
- Radix Sort

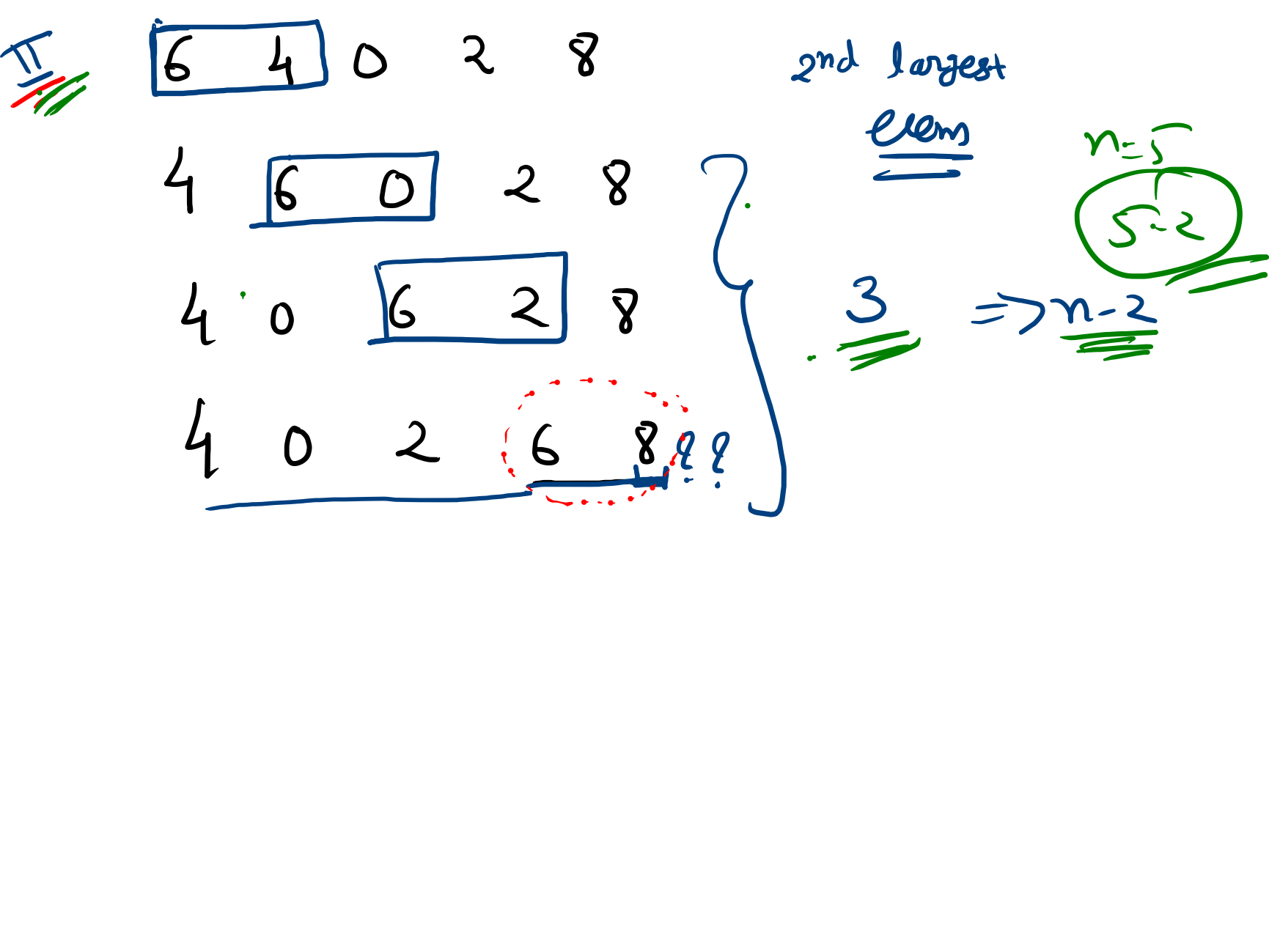
① Bubble Sort :-

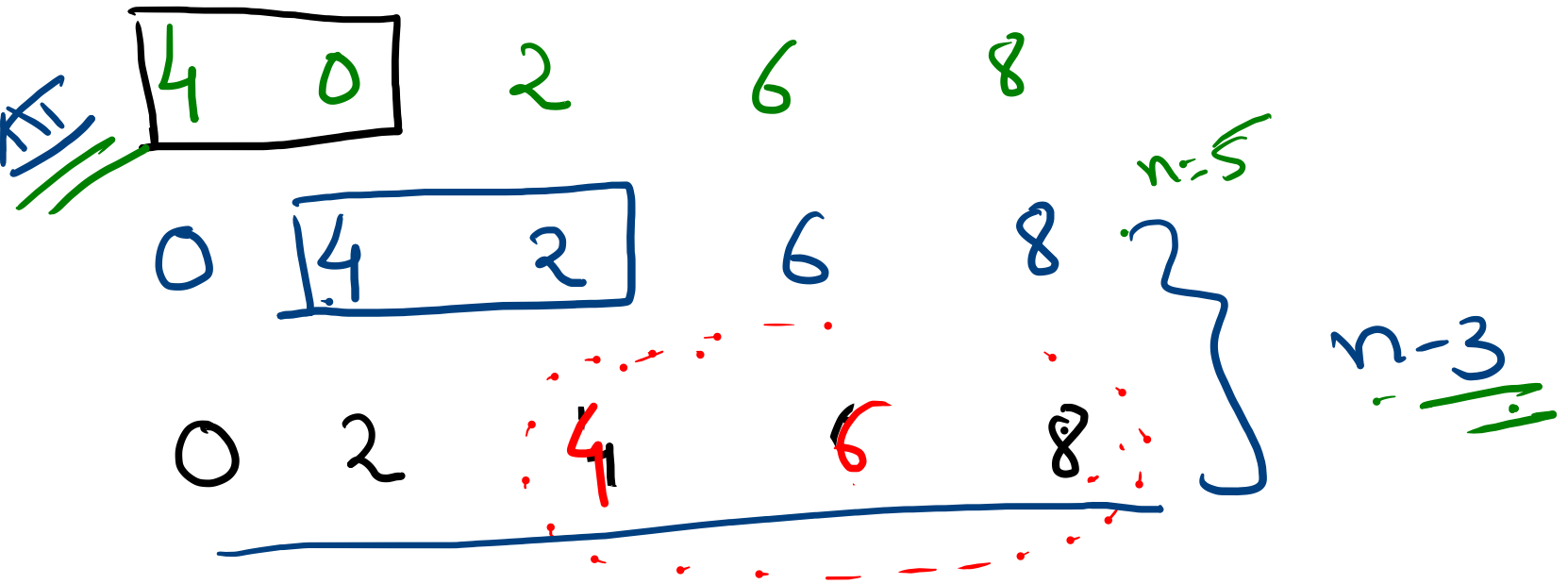


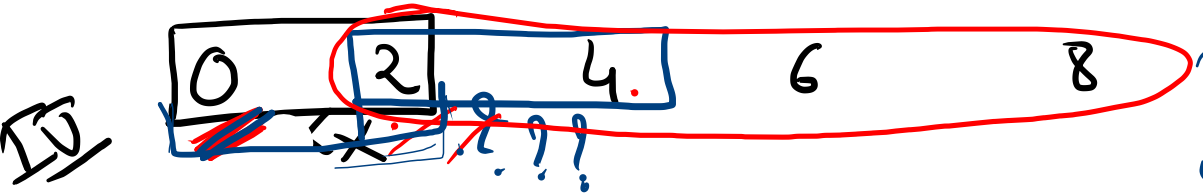


Sort -> ASC


Eg
8 6 4 0 2
n=5
Bubble
Swap
6 8 4 0 2
6 4 8 0 2
6 4 0 8 2
6 4 0 2 8
large -> last index
4 => n-1


6 4 0 2 8
4 6 0 2 8
4 0 6 2 8
4 0 2 6 8 ??
2nd largest elem
n=5
S-2
3
=> n-2


4 0 2 6 8
0 4 2 6 8
n-5
0 2 4 6 8
n-3



for array of size n =>

5 => 4

Size , n

Sort n-1

n -> n-1

n = 5

(n-1) + (n-2) + (n-3) + ...

n-1 terms

step => 4

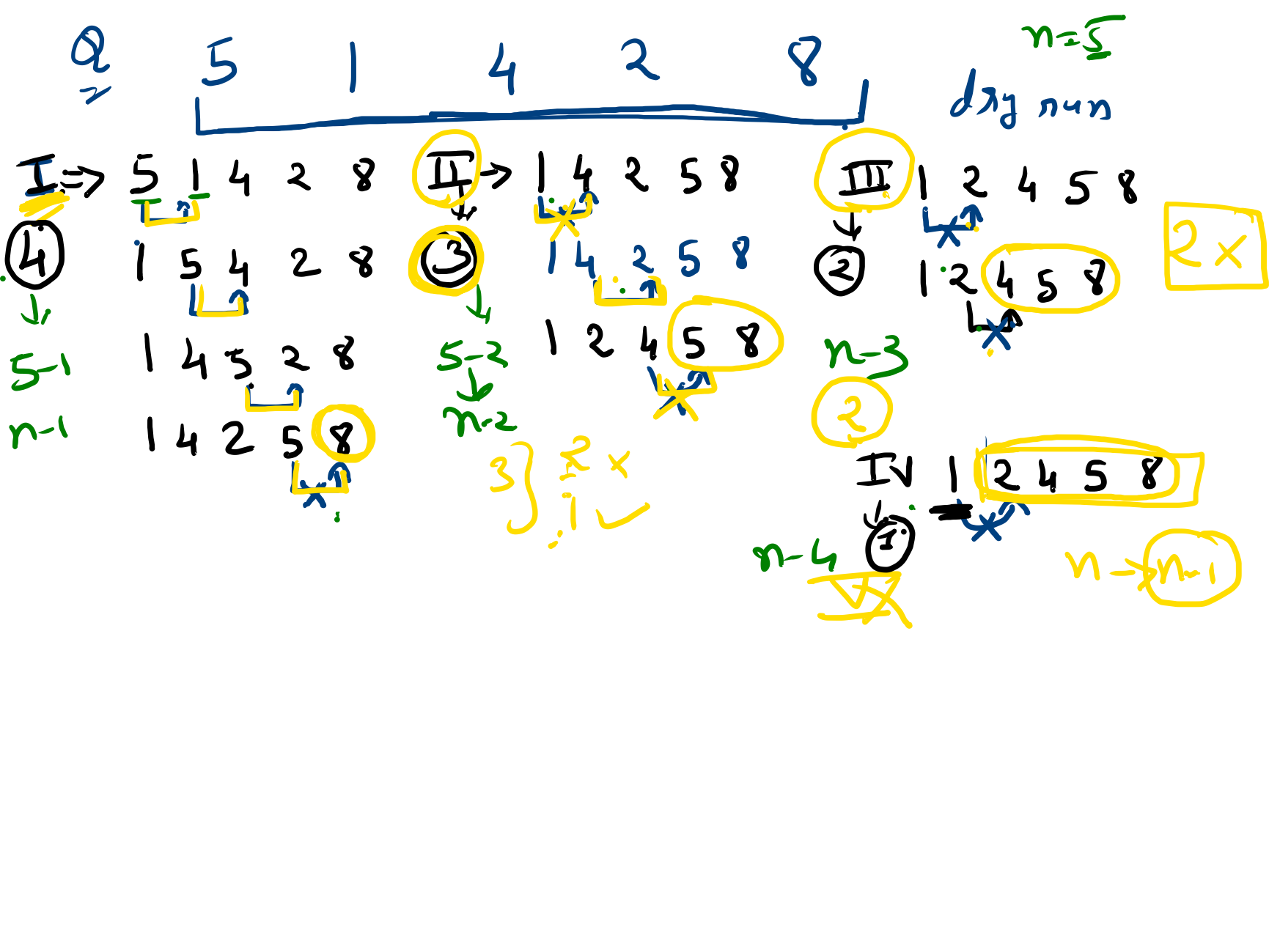
TC O(_)

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

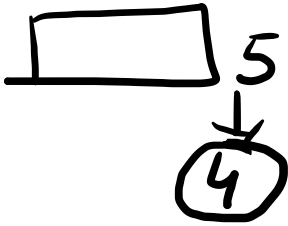
$= \frac{n(n+1)}{2} = n^2$

$1+2+3++\cdots+n \Rightarrow \frac{n(n+1)}{2}$

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


Q
5 1 4 2 8
n=5
dry run
I => 5 1 4 2 8
II => 1 4 2 5 8
III 1 2 4 5 8
IV 1 2 4 5 8
4
3
2
1
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
1 4 2 5 8
1 2 4 5 8
1 2 4 5 8
5-1
n-1
5-2
n-2
n-3
n-4
3 } 2x
2x
n → n-1

Psudo :-



(1) Iterate n-1 times

(2) for loop

1 -> n-1 times

2 -> n-2 times

3 -> n-3

i=0 => n-1

i=1 -> n-2

i=2 -> n-3

```
if ( compare adjacent
     arr[i] > arr[i+1] )
{
    swap
}
```

n=5

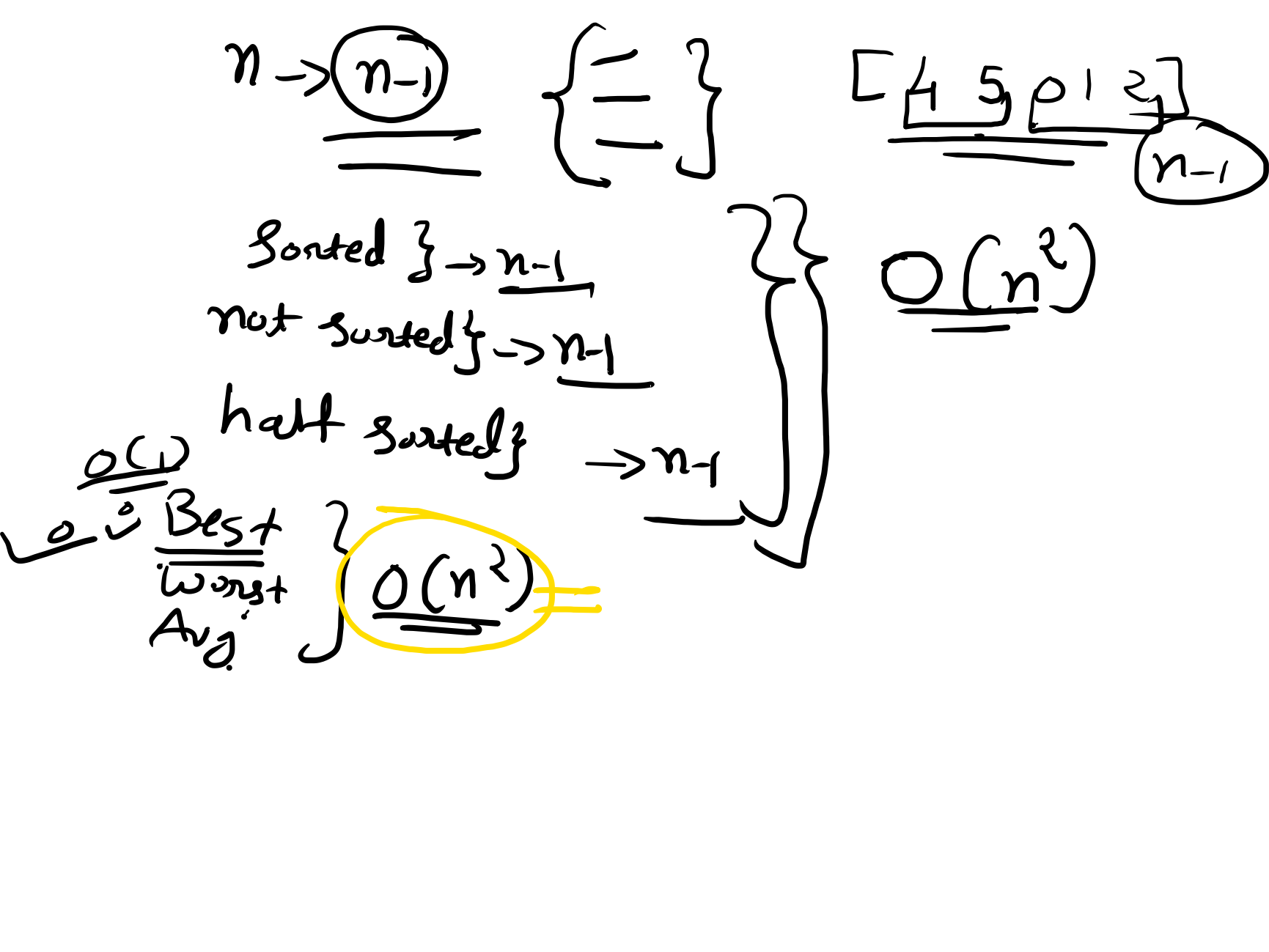
i=0 => n-1 swaps => 4 — 5-1-0

i=1 => n-2 swaps => 3 — 5-1-1

i=2 => n-3 swaps => 2 — 5-1-2

} n-1-i

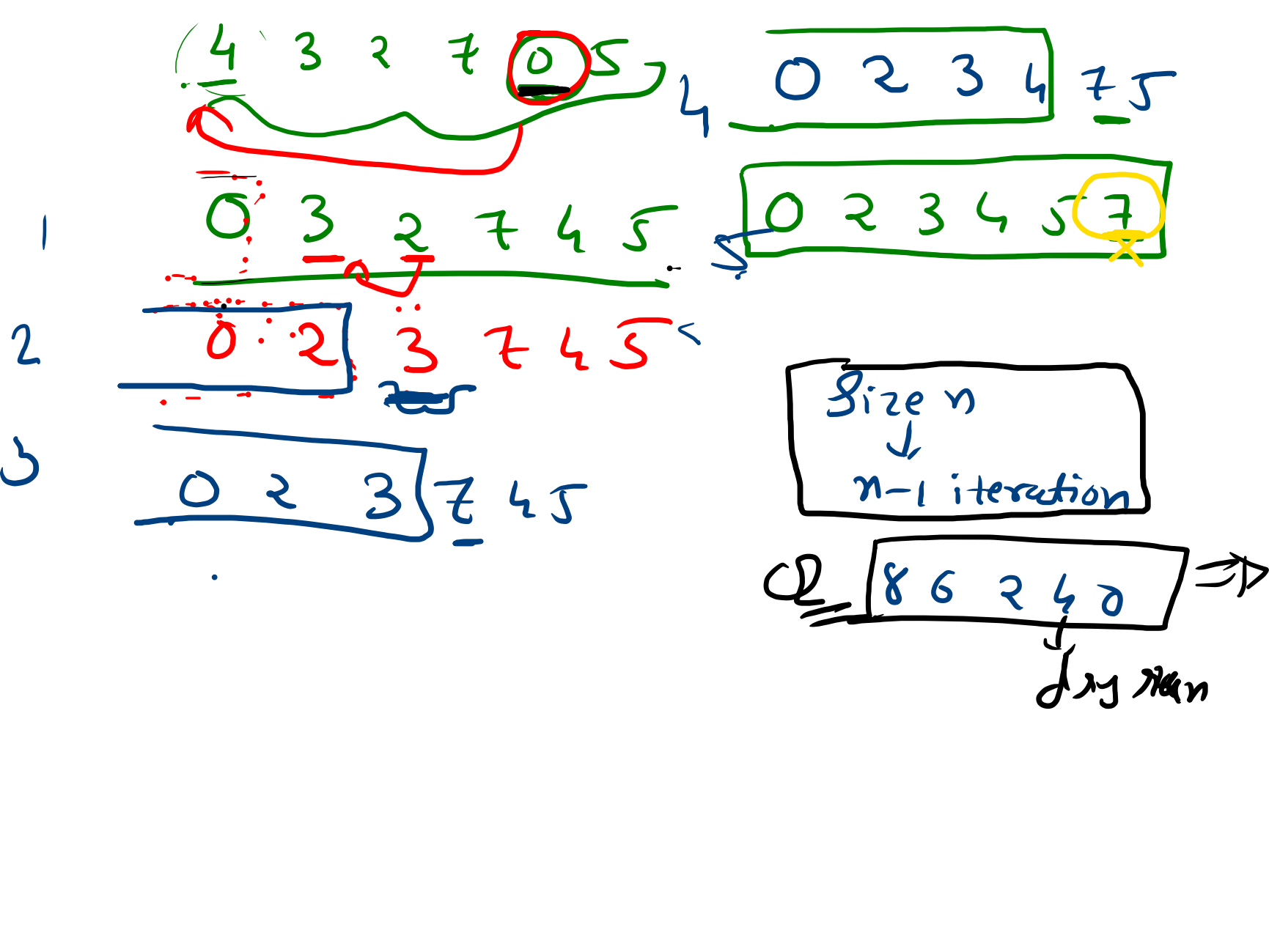
```
for(i=0; i< n-1; i++)
{
    for(j=0; n-1-i ; j++
    {
    }
}
```


n -> n-1
{ = }
[4 5 0 1 2]
n-1
Sorted } -> n-1
not sorted } -> n-1
halt sorted } -> n-1
O(n²)
O(1)
Best
Worst
Avg
O(n²)

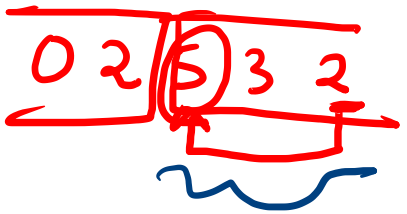
Selection Sort

Unsorted array:- 8 6 2 4 0

→ Selection Sort ⇒ Selecting right elem & putting it in correct position.


4 3 2 7 0 5
0 2 3 4 7 5
0 3 2 7 4 5
0 2 3 4 5 7
0 2 3 7 4 5
0 2 3 7 4 5
Size n
n-1 iteration
8 6 2 4 0
log n

1. Find min elem of unsorted region.
2. Swap the min & leftmost element in unsorted array.



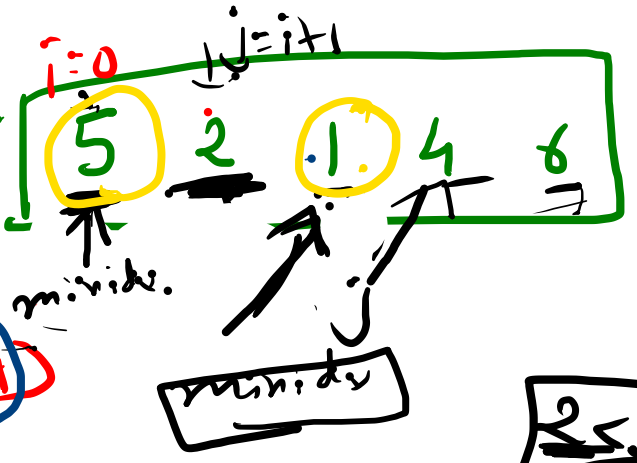
$n-1$ times

unsorted region

Q 8 6 2 4 0

. dry run

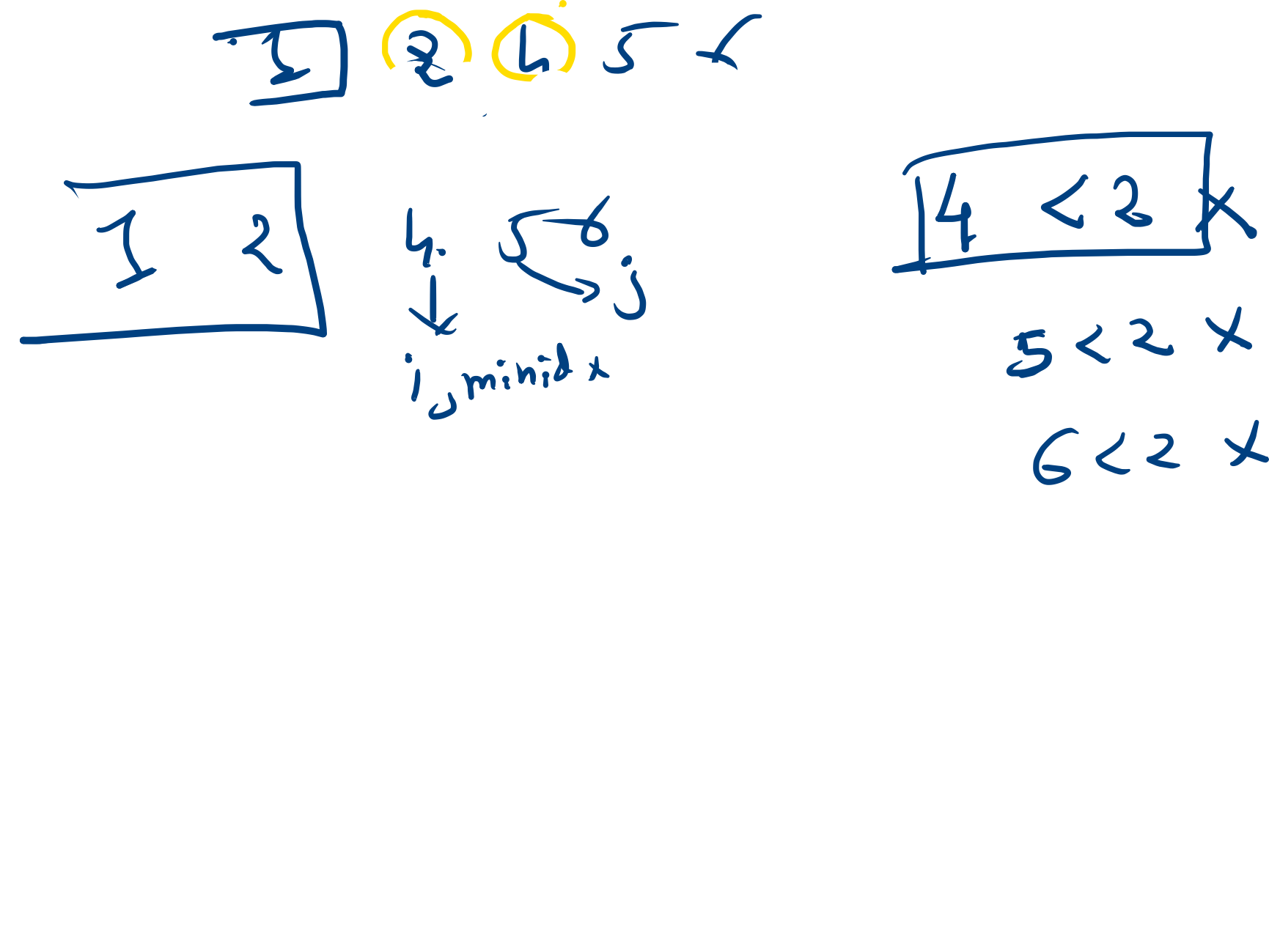
```
for(i=0; i< n-1; i++)
{
    let minIdx = i
    for(j= i+1 ; j<n ; j++)
    {
        if (arr[j] < arr[minIdx])
        {
            minIdx = j
        }
    }
    [arr[minIdx], arr[i]] = [arr[i], arr[minIdx]]
}
```



2<5

1<2

4<1


1 2 4 5 6
1 2
4
5
6
j
i, minid x
4 < 2 X
5 < 2 X
6 < 2 X

n + (n-1) + (n-2) + (n-3) + ...

```
function selectionSort(arr){
    let n = arr.length;
    for(i=0;i<n-1;i++){
        let minidx=i;
        for(j=i+1;j<n;j++){

            if(arr[j]<arr[minidx]){
                minidx=j;
            }

        }
        [arr[minidx],arr[i]]= [arr[i],arr[minidx]];
        console.log(arr)
    }
    return arr;
}

selectionSort([4,5,3,2,9])
```

O(1)

Bubble Sort O(1)

```
function bubbleSort(arr){
    let n = arr.length; // 5
    for(let i=0;i<n-1;i++){
        var isswapped = false;
        for(let j = 0;j < n-1-i;j++){
            if(arr[j] > arr[j+1]){
                //swap
                [arr[j],arr[j+1]] = [arr[j+1],arr[j]];
                isswapped=true;
            }
        }
        if(!isswapped){
            break;
        }
        console.log(arr)
    }
    return arr;
}

bubbleSort([5,1,4,2,8])
```

arr.length

early stopping

Such algo. which have a space comp. O(1) are known as inplace Sorting algo