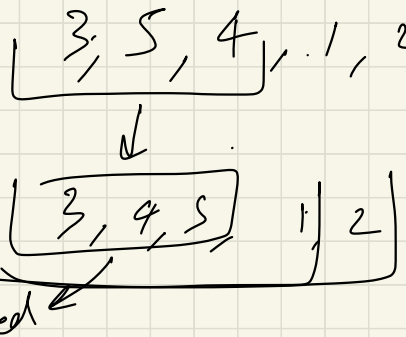
Insertion Sort :

After 1st pass : i = 0

This will be sorted

after 2nd pass : i = 1

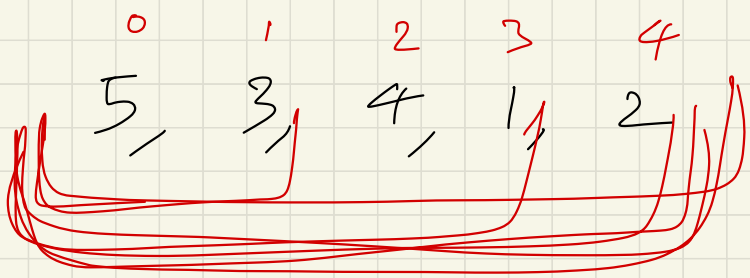


For every index :

Put that index element at the correct index of LHS.

To understand what every i is doing:

outer loop.



Sort array till index 1 ← pass no. 1

Sort array till index 2 ← pass no. 2

pass no. 3

pass no. 4

i j

0 1

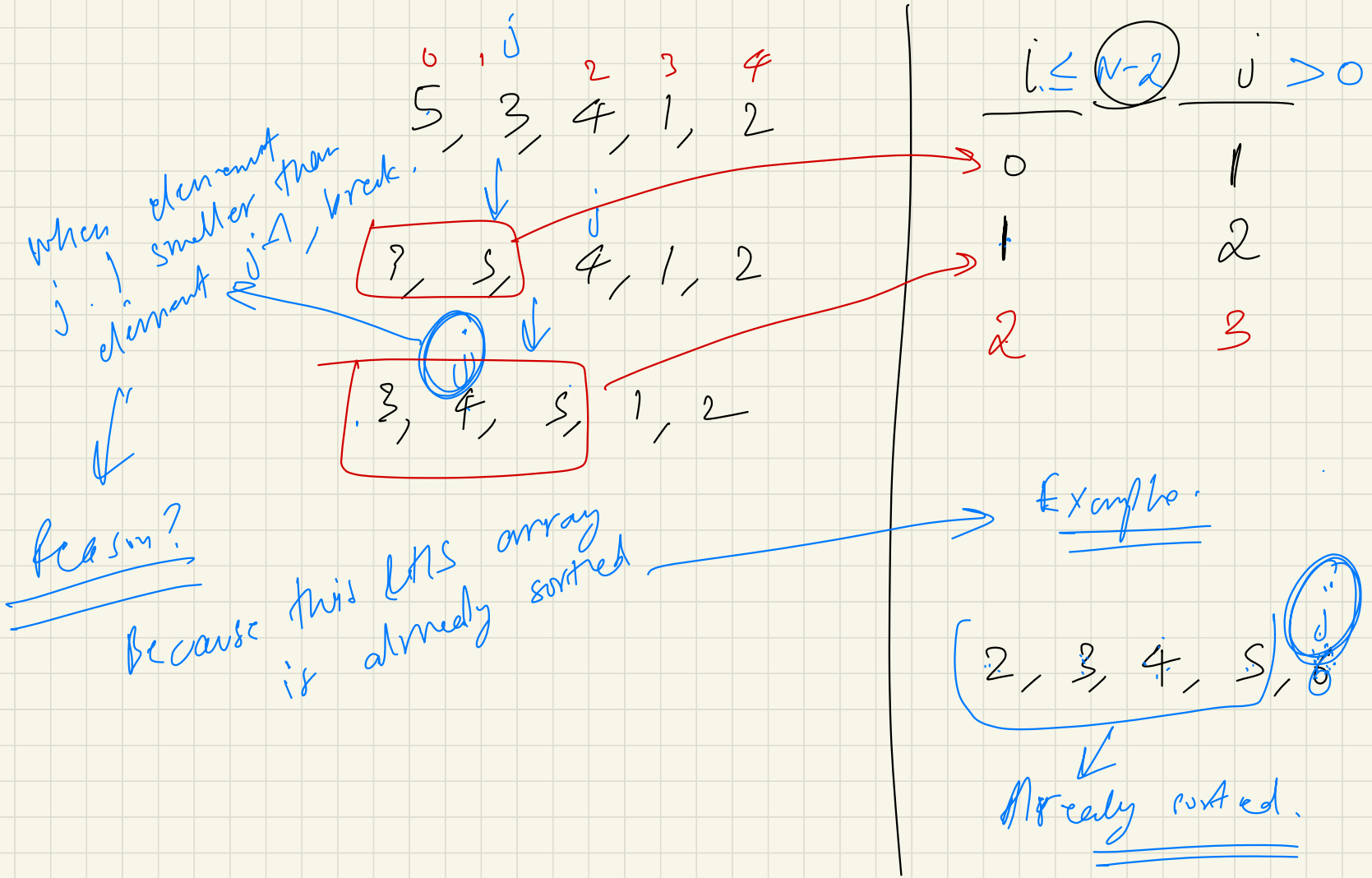
1

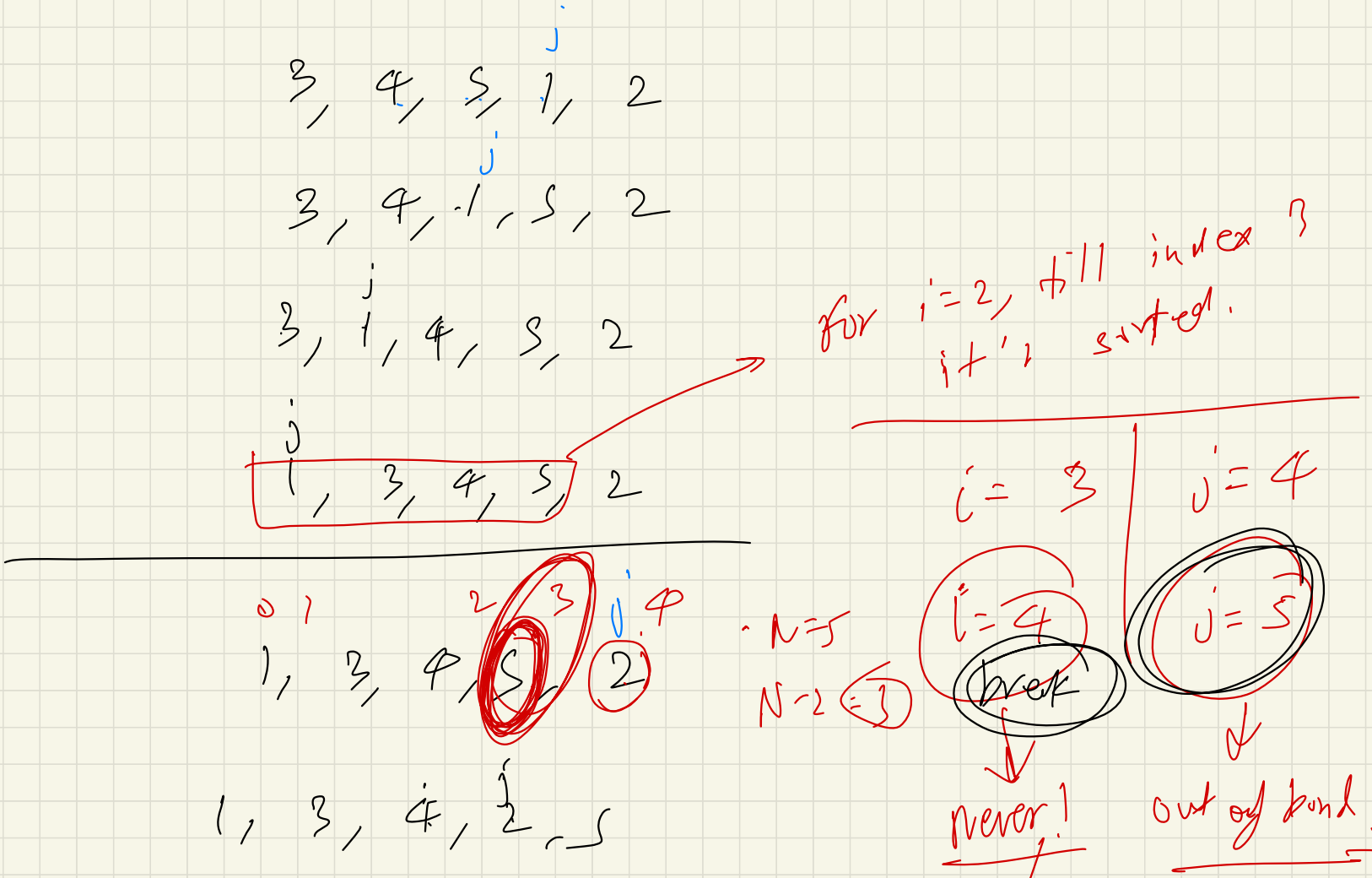
2

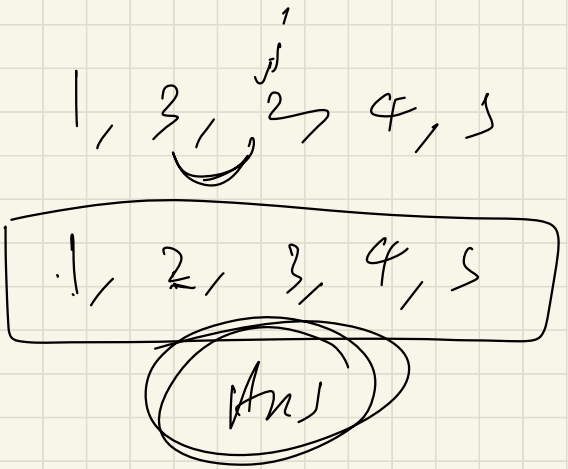
3

i will run from [0, N-2]

length of array !!


0 1 j 2 3 4
5, 3, 4, 1, 2
when element smaller than j-1 element, break.
j-1
j
?, 5, 4, 1, 2
3, 4, 5, 1, 2
i ≤ N-2
j > 0
0 1
1 2
2 3
Reason?
Because this LHS array is already sorted
Example.
2, 3, 4, 5, 6
Already sorted.


3, 4, 5, 1, 2
3, 4, 1, 5, 2
3, 1, 4, 5, 2
1, 3, 4, 5, 2
for i = 2, till index i is sorted.
i = 3
j = 4
i = 4
j = 5
break
N = 5
N-2 = 3
1, 3, 4, 5, 2
1, 3, 4, 2, 5
never!
out of bound!



Hence, $i \leq N-2$

length of arr

Complexity :

Worst Case : $O(N^2)$

(desc sorted)

no. of elements.

5, 4, 3, 2, 1

1, 2, 3, 4

1, 2, ... (N-1)

2, 3, 4, 5, 1

2, 3, 4, 1, 5

Sum of N nos: $\frac{N(N+1)}{2}$

$$\frac{(N-1)(N-1+1)}{2} = \frac{N(N-1)}{2}$$

Total no. of comparisons

$$= O\left(\frac{N^2-N}{2}\right) = O(N^2)$$

Why? Como watch the Time complexity

lecture.

Best case: Array is already sorted.

1, 2, 3, 4, 5

$(N-1)$ comparisons

Linear

$O(n)$

Why use it ?

- Adaptive : steps get reduced if array is sorted.
  No. of swaps reduced as compared to bubble sort.
- Stable
- Used for smaller values of N ⟹ works good when array is partially sorted. → It takes part in hybrid sorting algorithms.