

Insertion Sort:

Array \Rightarrow 5, 3, 4, 1, 2 [partially sorting the array]

For every index, put that index element at the correct index of LHS.

i.e., 1st pass $\rightarrow i = 0$ \rightarrow this will be sorted

5, 3, 4, 1, 2



3, 5, 4, 1, 2

2nd pass $\rightarrow i = 1$ \rightarrow this will be sorted

3, 5, 4, 1, 2



3, 4, 5, 1, 2

3rd pass $\rightarrow i = 2$ \rightarrow this will be sorted

3, 4, 5, 1, 2



1, 3, 4, 5, 2

4th pass $\rightarrow i = 3$ \rightarrow this will be sorted.

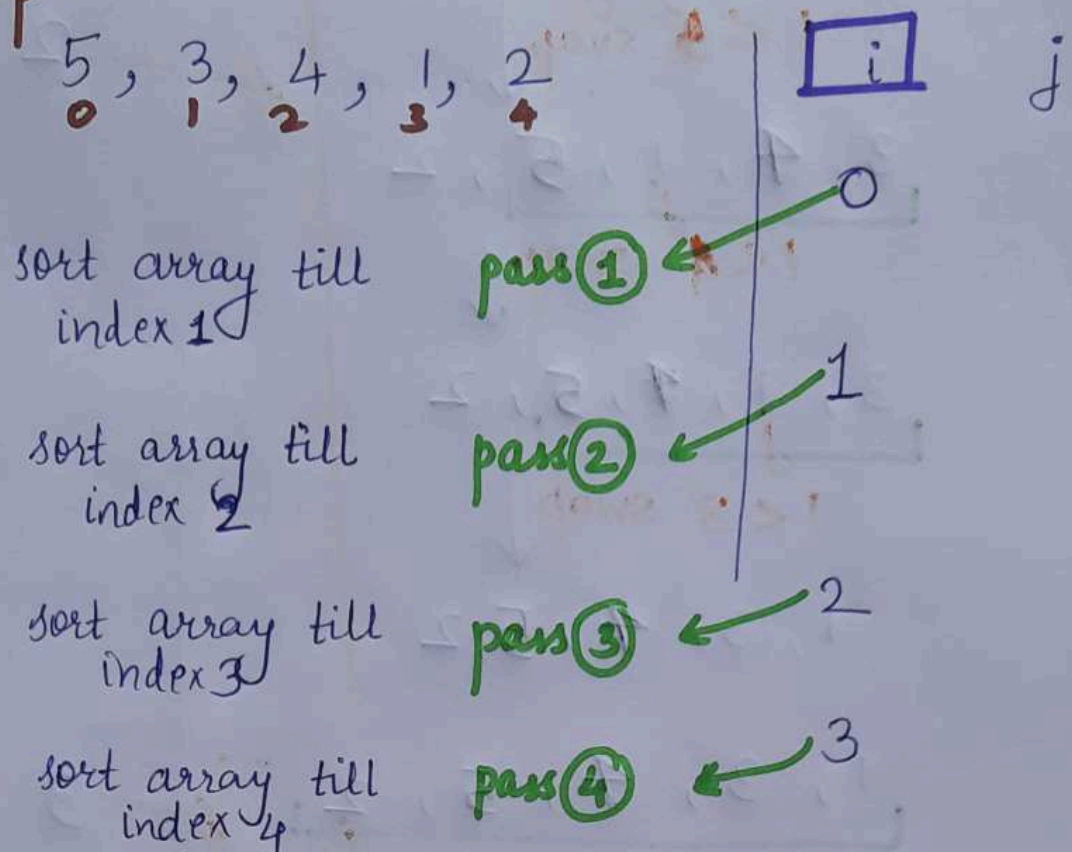
1, 3, 4, 5, 2



1, 2, 3, 4, 5 \rightarrow array is sorted !!

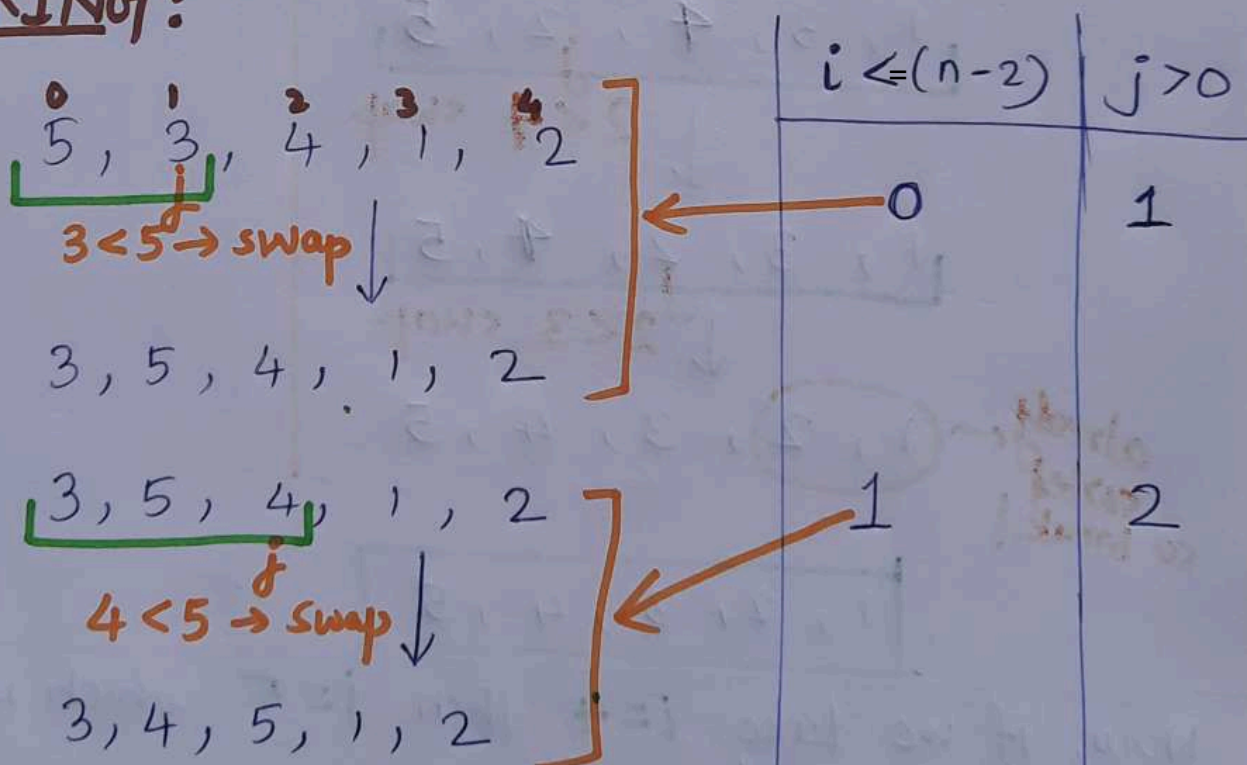
★ To understand what every 'i' is doing:

Outer loop:



i.e., 'i' will run from 0 to (n-2)

★ **WORKING:**



[now, since 3 < 4 already sorted]
when element j is not smaller than
element (j-1) break the loop because
the previous (LHS) side array is already
sorted.

	$i < (n-2)$	$j > 0$
$3, 4, 5, 1, 2$ $1 < 5$ swap $3, 4, 1, 5, 2$ $1 < 4$ swap $3, 1, 4, 5, 2$ $1 < 3$ swap $1, 3, 4, 5, 2$	2	3
$1, 3, 4, 5, 2$ $2 < 5$ swap $1, 3, 4, 2, 5$ $2 < 4$ swap $1, 3, 2, 4, 5$ $2 < 3$ swap $1, 2, 3, 4, 5$ <i>already sorted so break!</i>	3	4
$1, 2, 3, 4, 5$		

Now, if we take $i=4$ then $j=5$ which is index out of bound.

therefore, we take

$$i < (n-2)$$

where n is length of array.

Time Complexity :

① Worst case
[descending sorted] $\Rightarrow O(n^2)$

② Best case
[already sorted] $\Rightarrow O(n)$

Why to use insertion sort?

* **Adaptive** : steps get reduced if array is sorted
[i.e., no. of swaps are reduced (as compared to bubble sort)]

* **Stable Sorting Algorithm**

* **Used for smaller values of n** : works good
when array is partially sorted.
it takes part in hybrid sorting algorithm