Insertion Sort:

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

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

i.e., 1^{st} pass ⟶ $i = 0$ ⟶ this will be sorted

5, 3, 4, 1, 2

↓

3, 5, 4, 1, 2

2^{nd} pass ⟶ $i = 1$ ⟶ this will be sorted

3, 5, 4, 1, 2

↓

3, 4, 5, 1, 2

3^{rd} pass ⟶ $i = 2$ ⟶ this will be sorted

3, 4, 5, 1, 2

↓

1, 3, 4, 5, 2

4^{th} pass ⟶ $i = 3$ ⟶ this will be sorted.

1, 3, 4, 5, 2

↓

1, 2, 3, 4, 5 ⟶ array is sorted !!

* To understand what every 'i' is doing:

Outer loop:

0	1	2	3	4
5	3	4	1	2

	$\boxed{i}$	j
sort array till index 1 — pass ①	0	
sort array till index 2 — pass ②	1	
sort array till index 3 — pass ③	2	
sort array till index 4 — pass ④	3	

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

* WORKING:

0	1	2	3	4
5	3	4	1	2

j

3 < 5 → swap ↓

3, 5, 4, 1, 2

3, 5, 4, 1, 2

j

4 < 5 → swap ↓

3, 4, 5, 1, 2

$i <= (n-2)$	$j > 0$
0	1
1	2

[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.

$i<(n-2)$	$j>0$
2	3
3	4

i = 2

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

i = 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

already sorted so break!

$\boxed{1, 2, 3, 4, 5}$

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

therefore, we take $\boxed{i < (n-2)}$ where n is length of array.

Time Complexity :

① Worst Case [descending sorted] ⇒ $O(n^2)$

② Best Case [already sorted] ⇒ $O(n)$

Why to use insertion sort?

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