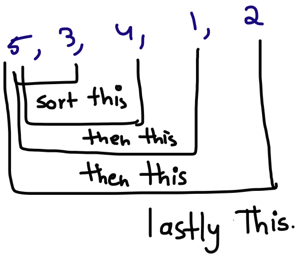
→ Insertion Sort

* Partially Sorting the array.

0 1 2 3 4
5, 3, 4, 1, 2

→ idea is to sort in Small Steps

Example



→ For every index : Put that index at the correct index of L.H.S

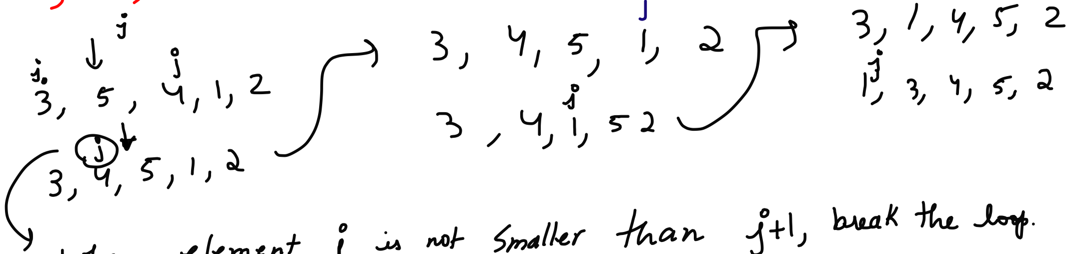
0 1 2 3 4
5, 3, 4, 1, 2

i = 3 , j = 4
i = 4 , j = 5] out of bound.

i ,	j > 0
0	1 Sort array till index 1
1	2 Sort array till index 2
2	3 sort array till index 3
3	4 sort array till index 4

* i will run from 0 to n-2 (n = arr.length) but why?

0 1 2 3 4
5, 3, 4, 1, 2 → j needs to be greater than 0.



when element j is not Smaller than j+1, break the loop. because left hand Side is already sorted.

Complexity Analysis

n= number of elements.

$O(n^2)$ worst case.

↳ (non-increasing sorted Array).

Best case: Sorted array.

	j_1	j_2	j_3	j_4
1,	2,	3,	4,	5
0	1	2	3	4

Total Comparison = n−1

Time complexity $O(n)$.

Why use insertion Sort?

→ Adaptive : Steps get reduced if array is sorted

→ Stable

→ Used for smaller values of N // recommended in Partially sorted Array

Pseudo code:

```
for  i = 0 ; i < arr. length −1 ; i++){
        for j=i+1 ; j >0 ; j−−) {
                if  (arr[j] < arr[j−1]){
                        swap (arr, j, j-1)
                else
                        break;
                }
        }
}
```

* while runnig your program use debugger.