Time Complexity:
The time complexity of the following algorithm will be $O(n^2)$ as at each pass we are traversing the array and swapping adjacent elements whenever needed. Each traversal is $O(n)$ and there are n traversals in total so $O(n * n) = O(n^2)$ in worst case time complexity.

Space Complexity:
It does not require any extra space so space complexity is $O(1)$. Thus it will be an in-place sorting algorithm.

Is Bubble Sort Stable?
Yes, Bubble Sort is a stable sorting algorithm. We swap elements only when A is less than B. If A is equal to B, we do not swap them, hence relative order between equal elements will be maintained.

Insertion Sort
Let's suppose you have an array of integer values and you want to sort these values in non-decreasing order. Then the logic of Insertion sort states that you need to insert the element into the correct position by looping over the array, thereby forming a sorted array on the left.
Then increase the pointer and similarly for the next element find its correct position in the left sorted array and insert it. Do this till the pointer reaches the end of the array.

Steps involved in Insertion Sort
- Traverse the array from left to right
- Compare the current element to its previous element and keep swapping it until the previous element is smaller than the current element.
- This way each element reaches it correct position

Note: For inserting the element in the left sorted array, we can also use binary search also to find the best place. Though overall complexity will remain the same for the code. For this we need to track the min and max element in the left portion and then we can use binary search to find the correct index.

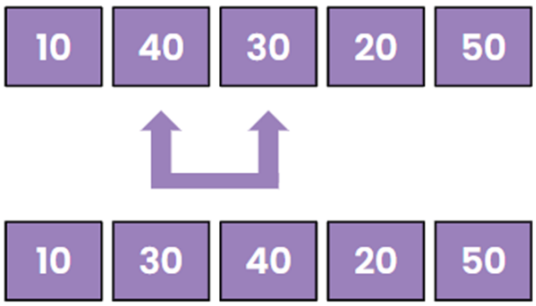
Let's understand insertion sort with an example.

Example (with all the steps):
Array = [10, 40, 30, 20, 50]

In the above example, as we iterate through the array from left to right, we keep inserting elements into their corresponding correct positions thereby forming a sorted array on the left.
First Pass: 10 is to be compared with its predecessor, here there is no previous element so simply move to the next element.
The sorted part of the array on the left is currently 10.
[10, 40, 30, 20, 50]

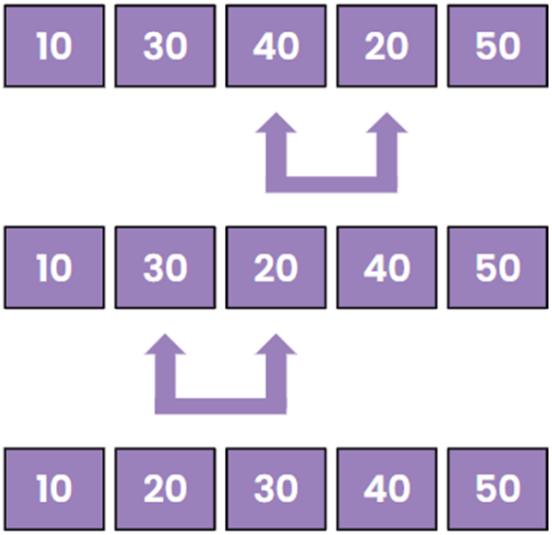
Second Pass: 40 is compared with its predecessor and the previous element is already smaller than the current element (10 < 40) so no need to swap anything and move to the next element.
The sorted part of the array on the left is currently [10,40].
[10, 40, 30, 20, 50]

Third Pass: 30 is now compared with its predecessor and here it is 40. As 40 < 30, we need to keep swapping the current element with its predecessors until the previous element is smaller than the current element. This way 30 will reach its correct position.
[10, 40, 30, 20, 50] (arrow between 40 and 30)
[10, 30, 40, 20, 50]



The sorted part of the array on the left is currently [10,30,40].

Fourth Pass: 20 is now compared with its predecessor that is 40 so as it is smaller than 40. We keep swapping it, it now reaches and array looks after swapping will be [10,30,20,40]. Again 20 is checked with its predecessor and as it is smaller than 30 also, it will be swapped again and the array will look like [10,20,30,40]. Now 20 has reached its correct position and no more swaps are needed.
[10, 30, 40, 20, 50] (arrow between 20 and 40)
[10, 30, 20, 40, 50] (arrow between 30 and 20)
[10, 20, 30, 40, 50]



Fifth Pass: Here the current element is 50 and is compared with its predecessor and no swaps are needed here. So the final array looks like this - [10,20,30,40,50]
[10,20,30,40,50]

Program for Insertion sort is

LP_CODE2.java

```
Enter the size of array
5
Enter the elements
10 40 30 20 50
Array after sorting
10 20 30 40 50
```

Time Complexity:
The time complexity of the following algorithm will be $O(n^2)$ as for every particular element we find its correct position and insert it at that particular position. This requires $O(n)$ time complexity for one element. We are doing this for every element so $O(n^2)$ time complexity in total

Space Complexity:
It does not require any extra space so space complexity is $O(1)$.

Is insertion sort stable?
Here we just pick an element and place it in its correct place and in the logic we are only swapping the elements if the element is larger than the key, i.e. we are not swapping the element with the key when it holds equality condition, so insertion sort is stable sort.

Selection Sort
This algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array.

- The subarray which is already sorted.
- The remaining subarray was unsorted

Let's understand the selection sort for sorting the array in non decreasing order.

As the name suggests, in the Selection Sort algorithm we select the index having minimum value by traversing through the array and swap it with the current index, then increment the current index.
This way we keep on traversing the rest of the array and finding the minimum index and swap it with the current index. This will result in forming a sorted array on the left side.

- Iterate through the array and consider the current element index.
- Now traverse the rest of the array elements and find the minimum element index.
- Swap it with the current element index and increment the current element index by one.
- Repeat until we reach the end of the array.

Now, let's take an example to understand the working of selection sort with sorting the array in increasing order.