Pre Requisites:

- Basic Java syntax

List of concepts involved :

- Sorting in an array
- Bubble Sort
- Insertion Sort
- Selection Sort

What is Sorting in an array?

Sorting in an array means to arrange the elements of the array in a certain order. The order can be ascending order or descending order. There are various sorting algorithms used in Java.

Sorting can be both Internal/Inplace and External/Outplace Sorting.

Internal Sorting can be defined as sorting which takes in place. It means all the data which needs to be sorted is stored in the same place while sorting is in progress.

External sorting: If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in a hard disk, floppy disk, or any other storage device, then we need to do sorting outside the memory. This means for external sorting we require additional memory or space to store the data which is sorted.

There are various ways to judge a sorting algorithm. We would consider 2 important things
1. Time complexity
2. Space Complexity
3. Stability of sorting.

An algorithm is said to be stable if for two equal keys/elements, the order before and after sorting remains the same. In simple terms, if 2 elements are equal they should be in the same order before and after sort.

Now we will study various sorting algorithms.

BUBBLE SORT

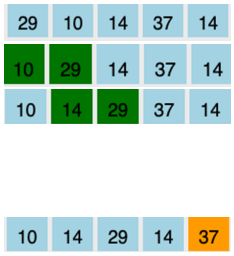
Bubble sort is the simplest amongst all sorting algorithms. The bubble sort works on swapping adjacent elements if they are in the wrong order and keeps on repeating this process till the list is sorted.

We would consider a simple array which becomes sorted using bubble sort.

Working of Bubble Sort
Suppose we are trying to sort the elements in ascending order.
eg . array:

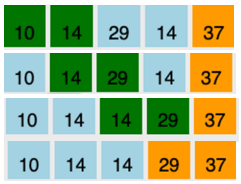
First Iteration (Compare and Swap)

1. Starting from the first index, compare the first and the second elements.
2. If the first element is greater than the second element, they are swapped.
3. Now, compare the second and the third elements. Swap them if they are not in order.
4. The above process goes on until the last element.



Elements swapped
Elements swapped
Nothing happened.
Elements swapped
we found our largest element of the array at the end.

2. Remaining iterations, we will keep on repeating this procedure until the array is sorted.



Nothing happened
Nothing happened
Elements swapped
29 is now at its correct position

3. We will keep repeating this procedure.

Nothing swapped
Nothing Swapped
Array is sorted.

Program to write bubble sort.

[LP_CODE1.java]

Note: Optimisation Technique.
If there is no swapping taking place in the current turn, then we can break the for loop as the array is already sorted. We don't need to go for indexes ahead. We can use a flag for this as mentioned in code.

Output

```
Enter the size of array
5
Enter the elements
4 2 3 1 5
Array after sorting
1 2 3 4 5
```

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]