**1. Problem Discussion:**

In this article, we are going to have a detailed discussion about the experimental comparison between two very important algorithms in the module “Time And Space Complexity”. These algorithms are “Merge Sort” and “Bubble Sort”.

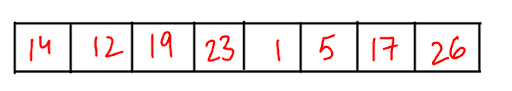
[**Merge Sort**](https://www.youtube.com/embed/aiUHB-3EOg8)

[**Bubble Sort**](https://www.youtube.com/embed/Jv-eGC2xmtU)

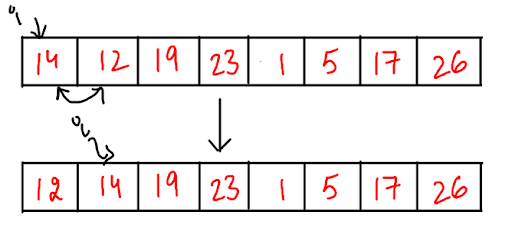
We already know about these algorithms and their implementation in detail. In case you haven’t watched the videos yet, we highly suggest you to go and take a look and understand them properly before heading over to this article.

**2. Approach:**

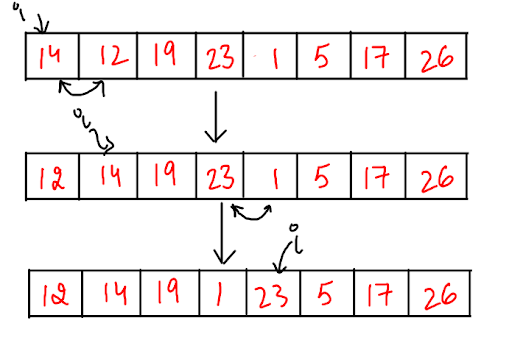
So, we will take an array and try to sort it using both of these algorithms and see the differences in the approaches as well as the complexities of both of these algorithms. Let’s say our array is as follows:

****

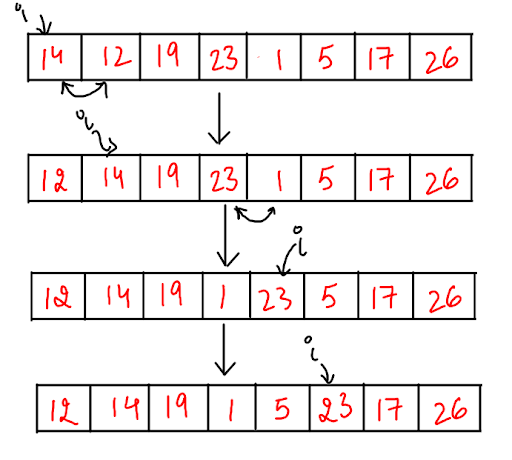
Let’s sort this using bubble sort first. In bubble sort, we check neighbour elements, and swap them if the ith element is greater than the (i+1)th element when we need to sort the array in increasing order. So, the first thing we see is 14 > 12. So, we will swap the two, creating the array as follows:

****

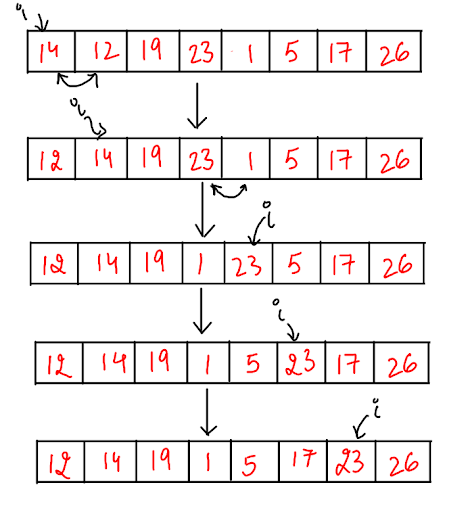
Now, 14 < 19, so these will not be swapped. i will move to 19. 19 < 23. So, these two will also not be swapped. Now, i will point at 23. 23 > 1. So, these two will be swapped, and the array will look as follows:

****

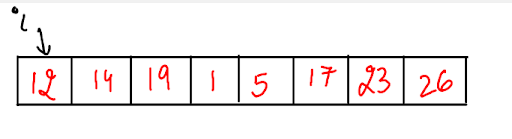
23 > 5, so we will swap the two and array will become as follows:

****

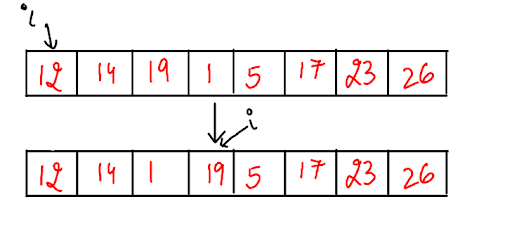
23 > 17, so these two will be swapped and the array becomes:

****

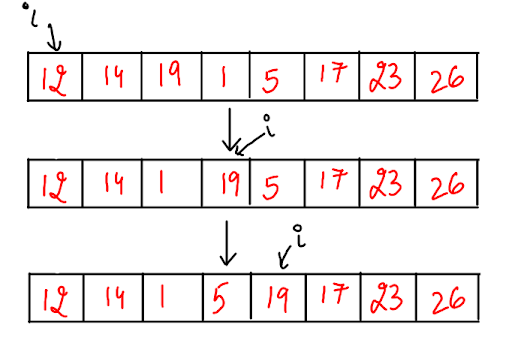
Now, 23 < 26. So, these two will not be swapped. This marks the end of the first parse. Now, we will parse the array again.

****

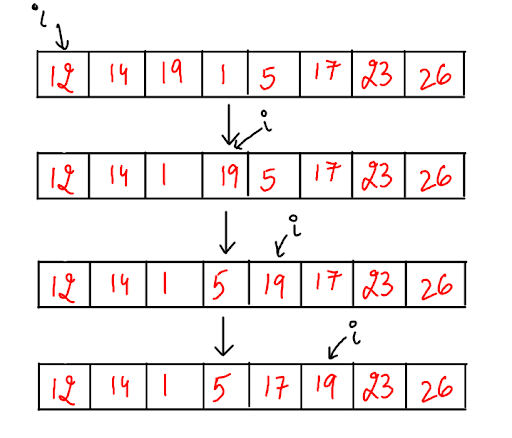
12 < 14. The two will not be swapped. So, i moves to 14. 14 < 19. These two will also not be swapped. So, i moves to 19. 19 < 1. So, these two will be swapped and array becomes:

****

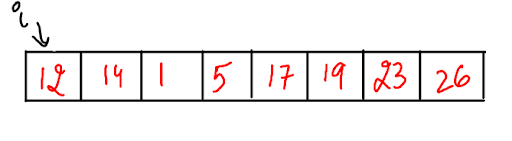
19 > 5, so these two will also be swapped and the array becomes:

****

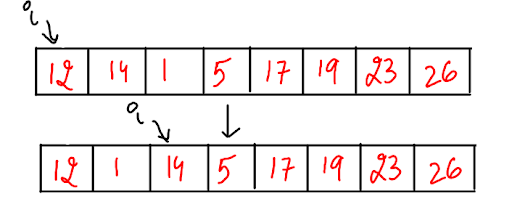
19 > 17, so these two will be swapped and the array becomes as follows:

****

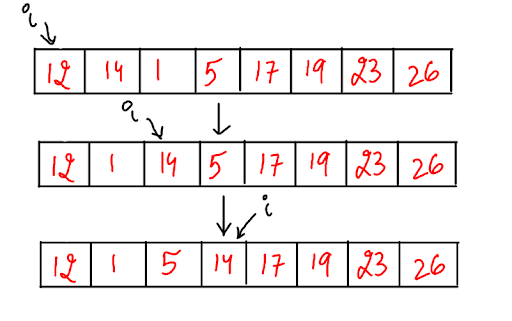
Now, 19 < 23, so these two will not be swapped. And i moves to 23. 23 < 26, so these two will also not be swapped. This will mark the end of second parse. The array is still not sorted, so we will parse it again.

****

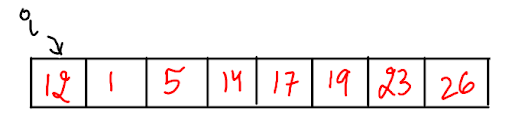
12 < 14, so i moves to 14. 14 > 1, so these two will be swapped, and the array becomes:

****

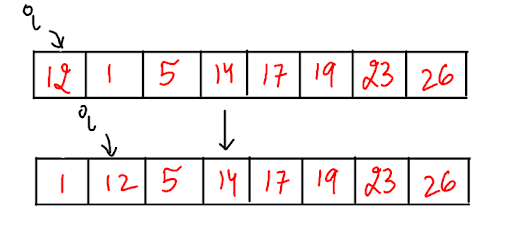
14 < 5, so these two will also be swapped, and the array becomes:

****

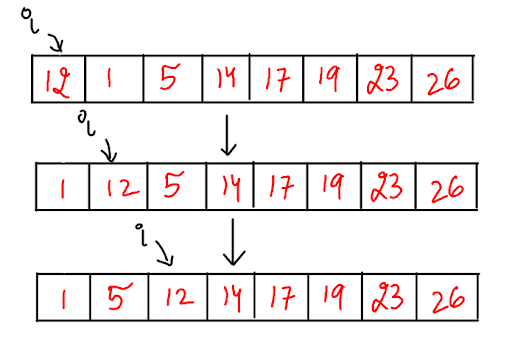
After this point, the array seems to be sorted, so i will reach the end of the array and this will end the third parse. But, the array is still not sorted. So, we will make another parse.

****

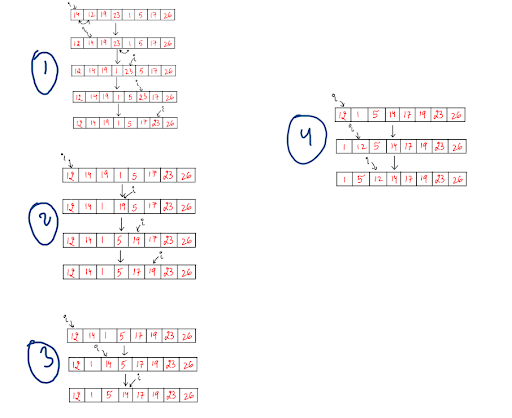
12 > 1, so the two will be swapped and the array becomes:

****

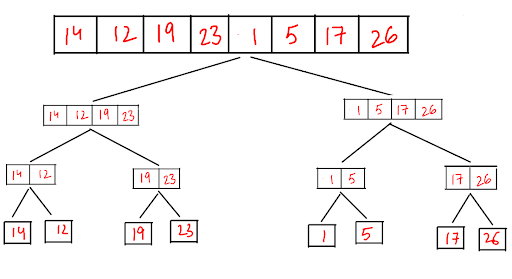
12 > 5, so these two will be swapped and the array becomes as follows:

****

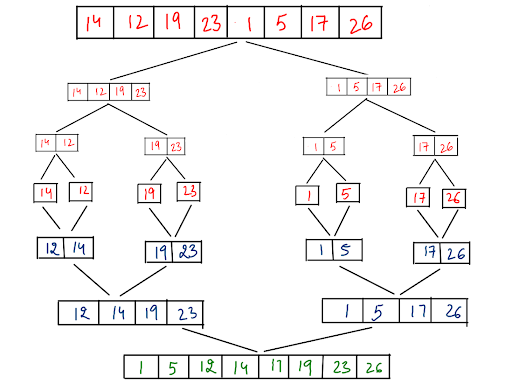
After this point, the array seems to be completely sorted. There will be another parse and now, the array is completely sorted after all these parses.

****

Now, let’s sort this using merge sort. So, this method follows the approach of dividing the array into two parts first, and further dividing those parts until we reach a point where each element exists as a single array. This looks as follows:

****

Now, we can use the algorithm of “merge two sorted arrays to merge these divided arrays in the following manner:

****

So, this is how merge sort helps us to sort and array by the divide and conquer method.

**3. Analysis for bubble sort:**

Time Complexity: O(n\*n)

Space Complexity: O(1)

**4. Analysis for merge sort:**

Time Complexity: O(logn)

Space Complexity: O(n)