Summary of insertion sort

Insertion sort is the sorting algorithm where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. This can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

**Insertion sort works as follows:**

* The first step involves the comparison of the element in question with its adjacent element.
* And if at every comparison reveals that the element in question can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
* The above procedure is repeated until all the element in the array is at their apt position.

For example, Consider the following array: 12, 10, 2

* **First Iteration**: Compare 12 with 10. The comparison shows 10< 12. Hence swap 10 and 12.

The array now looks like: 10,12,2

* **Second Iteration**: Begin with the second element (12), but it was already swapped on for the correct position, so we move ahead to the next element.

Now hold on to the third element (31) and compare with the ones preceding it.

Since 2<12> 25, we swap 2 and 12. Also, since 2> 12, we swap 2 and 10

The array after the Second iteration looks like: 2,10,12

This is the final array after all the corresponding iterations and swapping of elements.

**Time complexity analysis**

Even though insertion sort is efficient, still, if we provide an already sorted array to the insertion sort algorithm, it will still execute the outer for loop, thereby requiring n steps to sort an already sorted array of n elements, which makes its best case time complexity a linear function of n.

Wherein for an unsorted array, it takes for an element to compare with all the other elements which mean every n element compared with all other n elements. Thus, making it for n x n, i.e., comparisons.

**Best Case Time Complexity: O(n)**

**Worst Case Time Complexity: O(n2)**