

Insertion Sort Algorithm

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion sort has various advantages such as -

- Simple implementation
- Efficient for small data sets
- Adaptive, i.e., it is appropriate for data sets that are already substantially sorted.

Working of Insertion sort Algorithm

Now, let's see the working of the insertion sort Algorithm.

To understand the working of the insertion sort algorithm, let's take an unsorted array. It will be easier to understand the insertion sort via an example.

Let the elements of array are -

12	31	25	8	32	17
----	----	----	---	----	----

Initially, the first two elements are compared in insertion sort.

12	31	25	8	32	17
----	----	----	---	----	----

Here, 31 is greater than 12. That means both elements are already in ascending order. So, for now, 12 is stored in a sorted sub-array.

12	31	25	8	32	17
----	----	----	---	----	----

Now, move to the next two elements and compare them.

12	31	25	8	32	17
----	----	----	---	----	----

12	31	25	8	32	17
----	----	----	---	----	----

Here, 25 is smaller than 31. So, 31 is not at correct position. Now, swap 31 with 25. Along with swapping, insertion sort will also check it with all elements in the sorted array.

For now, the sorted array has only one element, i.e. 12. So, 25 is greater than 12. Hence, the sorted array remains sorted after swapping.

12	25	31	8	32	17
----	----	----	---	----	----

Now, two elements in the sorted array are 12 and 25. Move forward to the next elements that are 31 and 8.

12	25	31	8	32	17
----	----	----	---	----	----

12	25	31	8	32	17
----	----	----	---	----	----

Both 31 and 8 are not sorted. So, swap them.

12	25	8	31	32	17
----	----	---	----	----	----

After swapping, elements 25 and 8 are unsorted.

12	25	8	31	32	17
----	----	---	----	----	----

So, swap them.

12	8	25	31	32	17
----	---	----	----	----	----

Now, elements 12 and 8 are unsorted.

12	8	25	31	32	17
----	---	----	----	----	----

So, swap them too.

8	12	25	31	32	17
---	----	----	----	----	----

Now, the sorted array has three items that are 8, 12 and 25. Move to the next items that are 31 and 32.

8	12	25	31	32	17
---	----	----	----	----	----

Hence, they are already sorted. Now, the sorted array includes 8, 12, 25 and 31.

8	12	25	31	32	17
---	----	----	----	----	----

Move to the next elements that are 32 and 17.

8	12	25	31	32	17
---	----	----	----	----	----

17 is smaller than 32. So, swap them.

8	12	25	31	17	32
---	----	----	----	----	----

8	12	25	31	17	32
---	----	----	----	----	----

Swapping makes 31 and 17 unsorted. So, swap them too.

8	12	25	17	31	32
---	----	----	----	----	----

8	12	25	17	31	32
---	----	----	----	----	----

Now, swapping makes 25 and 17 unsorted. So, perform swapping again.

8	12	17	25	31	32
---	----	----	----	----	----

Now, the array is completely sorted.

Insertion sort complexity

Now, let's see the time complexity of insertion sort in best case, average case, and in worst case. We will also see the space complexity of insertion sort.

1. Time Complexity

Case	Time Complexity
Best Case	$O(n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity** - It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of insertion sort is **$O(n)$** .
- **Average Case Complexity** - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of insertion sort is **$O(n^2)$** .
- **Worst Case Complexity** - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of insertion sort is **$O(n^2)$** .

2. Space Complexity

Space Complexity	$O(1)$
Stable	YES

- The space complexity of insertion sort is $O(1)$. It is because, in insertion sort, an extra variable is required for swapping.

Insertion Sort Algorithm

```
package Sorting;

import java.util.Arrays;
public class InsertionSort {
    public static void main(String[] args) {
        int[] arr = {4,3,2,5,1};
        sort(arr);
    }

    public static void sort(int[] arr) {
        for (int i = 0; i < arr.length-1; i++) {
            for (int j = i+1; j > 0 ; j--) {
                if (arr[j] < arr[j-1]) {
                    swap(arr, j, j-1);
                }
                else{
                    break;
                }
            }
            System.out.println(Arrays.toString(arr));
        }

        public static void swap (int[] arr, int start, int end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
        }
    }
}
```