INSERTION SORT

How does Insertion Sort Work?

Insertion Sort is a sorting algorithm that operates by gradually building a sorted segment within the array. It starts by considering the first element as a sorted segment and then iterates through the remaining elements. During each iteration, the algorithm compares the current element with the elements in the sorted segment, shifting larger elements to the right to make space for the current element. This process continues until all elements are appropriately positioned, resulting in a sorted array.

Time Complexity

Best Case (O(N)): When the array is nearly sorted, Insertion Sort performs comparisons and insertions efficiently. Each element needs just a few comparisons, resulting in a linear time complexity of O(N).

Average & Worst Case (O(N^2)): In cases where the array is unordered or in reverse order, Insertion Sort requires more comparisons and shifts. It uses nested loops to compare each element with the preceding elements in the sorted segment, leading to a quadratic time complexity of O(N^2).

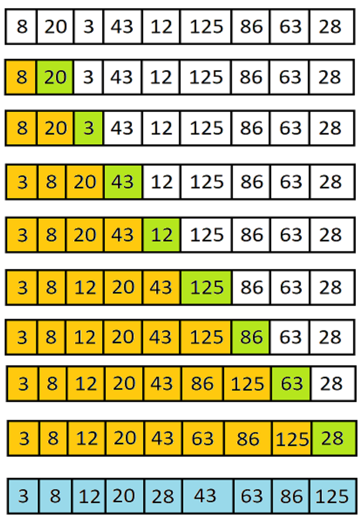
Space Complexity

Insertion Sort possesses a space complexity of O(1). It efficiently sorts the array in-place without necessitating additional memory proportional to the input size.

Discussion

Insertion Sort's primary advantage lies in its efficiency for nearly sorted arrays or small datasets. While its time complexity is quadratic, its simplicity and in-place nature make it effective for cases where the dataset is limited in size. For larger datasets, more advanced sorting algorithms are generally favored due to their superior time complexities.

Here is the visualization result of the Insertion Sort implementation, where elements are progressively shifted into the sorted segment:



8 20 3 43 12 125 86 63 28	Unsorted List
8 20 3 43 12 125 86 63 28	20, being the largest element among those already compared, must not be shifted.
8 20 3 43 12 125 86 63 28	3 must be shifted (3 < 8 < 20)
3 8 20 43 12 125 86 63 28	43, being the largest element among those already compared, must not be shifted.
3 8 20 43 12 125 86 63 28	12 must be shifted (3 < 8 < 12 < 20 < 43)
3 8 12 20 43 125 86 63 28	125, being the largest element among those already compared, must not be shifted.
3 8 12 20 43 125 86 63 28	86 must be shifted (3 < 8 < 12 < 20 < 43 < 86 < 125)
3 8 12 20 43 86 125 63 28	63 must be shifted (3 < 8 < 12 < 20 < 43<63 < 86 < 125)
3 8 12 20 43 63 86 125 28	28 must be shifted (3 <8 < 12 < 20 < 28 < 43 < 63 < 86 < 125)
3 8 12 20 28 43 63 86 125	Sorted List

🟨 represents the sorted subarray

🟩 represents the current element