Categories of Sorting Algorithms

Sorting algorithms can be categorized as stable or non-stable, depending on how they handle equal (duplicate) elements in a collection. The key distinction lies in whether or not the relative order of equal elements is preserved after sorting.

Stable Sorting Algorithms:

A stable sorting algorithm maintains the relative order of equal elements in the sorted output as they appeared in the original, unsorted input. In other words, if two elements are equal, the one that appeared first in the original list will still appear before the other in the sorted list.

Example of Stable Sorting:

Consider the following list of student records sorted by age:

Name Age

John 25

Jane 20

Bob 25

Alice 20

If we sort this list by age using a stable sorting algorithm, the relative order of students with the same age will be preserved:

Original order (sorted by insertion):

Jane (20), Alice (20), John (25), Bob (25)

Stable sort by age:

Jane (20), Alice (20), John (25), Bob (25)

Since Jane came before Alice with the same age (20) in the original list, she remains before Alice after sorting.

Examples of Stable Sorting Algorithms:

- Bubble Sort
- Merge Sort
- Insertion Sort

Non-Stable Sorting Algorithms:

A non-stable sorting algorithm does not guarantee preserving the relative order of equal elements. After sorting, two equal elements might end up in a different relative order than they were originally.

Example of Non-Stable Sorting:

Using the same list of students:

Name Age

John 25

Jane 20

Bob 25

Alice 20

If we sort this list by age using a non-stable sorting algorithm, the relative order of students with the same age might change:

Non-stable sort by age:

Alice (20), Jane (20), Bob (25), John (25)

Notice that now Alice (20) appears before Jane (20), even though Jane appeared first in the original list.

Non-stable sort by age:

Alice (20), Jane (20), Bob (25), John (25)

Notice that now Alice (20) appears before Jane (20), even though Jane appeared first in the original list.

Examples of Non-Stable Sorting Algorithms:

- Quick Sort
- Heap Sort
- Selection Sort
- Shell Sort

Sorting algorithms can also be classified as adaptive or non-adaptive, based on how efficiently they handle partially sorted input data.

Adaptive Sorting Algorithms:

- They recognize patterns in the input data (such as elements that are already in order).
- They perform fewer operations (like comparisons or swaps) when the data is partially sorted.
- Time complexity improves when the input is close to sorted.

Example: Insertion Sort: If the input is already sorted or nearly sorted, Insertion Sort performs in O(n) time because it only checks each element once.

Merge Sort (Modified): Adaptive versions of Merge Sort can adjust based on the sortedness of the input.

Non-Adaptive Sorting Algorithms:

- They do not adjust based on the initial order of the input.
- The algorithm follows its sorting process regardless of whether the input is sorted or not.
- Time complexity remains the same regardless of the degree of sortedness in the input data.
  Example: Selection Sort: Always performs the same number of comparisons and swaps regardless of how sorted the input is. Its time complexity is $O(n^2)$, even if the list is almost sorted.
  Heap Sort: Even if the input is partially sorted, Heap Sort does not exploit that information, and it still builds a heap and performs the same operations.
  Quick Sort: Although efficient on average, Quick Sort is generally considered non-adaptive because it doesn't adjust based on the sortedness of the input.