Alias Modified Merge Sort

Quick Sort follows a similar approach to Merge Sort but with notable differences in its execution. The key idea is to sort a single element in such a way that all elements to its left are smaller, and all elements to its right are greater. This partitioning is executed recursively for both the left and right subparts of the array.

> **Counting Sort** is a linear time sorting algorithm ideal for integers within a known range. It achieves sorting by counting the frequency of each element and then placing them in the desired sorted order.

Logic

The primary logic involves sorting a single element while maintaining the property that elements to its left are smaller and elements to its right are greater. This partitioning approach is then applied recursively to the left and right subarrays.

C++ implementation of Quick sort: Quick Sort.
Python implementation of Quick sort: Quick Sort.
Java implementation of Quick sort: Quick Sort.
Javascript implementation of Quick sort: Quick Sort.

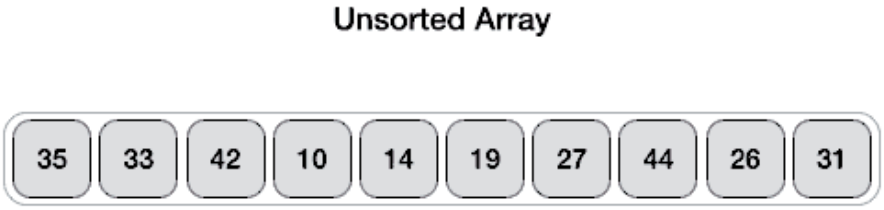

Figure 1: Quick Sort Partition Animation

Complexity

- **Time Complexity:**
  - Quick Sort exhibits an average and best-case time complexity of O(nlogn), making it efficient for most scenarios.
  - However, in the worst case, it can degrade to O(n^2).
- **Space Complexity:**
  - The auxiliary space required for Quick Sort is O(n).

Advantages and Considerations

- Quick Sort's in-place partitioning makes it memory-efficient in comparison to Merge Sort.
- The efficient average-case time complexity makes it a popular choice for general-purpose sorting.
- However, the worst-case time complexity and performance characteristics on already sorted data warrant careful consideration in specific scenarios.

#sorting [[SORTING]]