**Time Complexity Analysis Of Quick-sort.**

* The best-case time complexity of Quick-sort is O(n log n), which occurs when the pivot chosen is the median element.
* The average-case time complexity is also O(n log n).
* The worst-case time complexity is O(n^2), which occurs when the pivot chosen is the smallest or largest element and the array is already sorted.
* However, in practice, Quick-sort tends to outperform other algorithms like Merge Sort due to its efficient partitioning.

**Comparison with Merge Sort:**

**Advantages of Quick-sort:**

* Quick-sort typically has better average-case performance compared to Merge Sort. It often requires fewer comparisons and swaps to sort an array.
* Quick-sort is an in-place sorting algorithm, meaning it does not require additional memory for temporary arrays, making it more memory-efficient.
* Quick-sort's cache performance is usually better due to its partitioning scheme, resulting in fewer cache misses.

**Disadvantages of Quick-sort:**

* The worst-case time complexity of Quick-sort is higher compared to Merge Sort. In certain scenarios, it may degrade to O(n^2) time complexity, which is undesirable.
* Quick-sort's performance heavily depends on the choice of pivot, and a bad pivot choice can lead to sub-optimal performance.
* Quick-sort is not stable, meaning it may change the relative order of equal elements in the input array.

**Performance Analysis:**

To perform the performance analysis, we created arrays of different sizes (20, 30, and 50 elements) with randomly generated integers. We measured the time it took to sort each array using the Quick sort algorithm. The results may vary based on the system's hardware and other running processes, but you can observe that Quick-sort should generally be faster for larger arrays compared to other O(n log n) sorting algorithms like Merge Sort.