1. Insertion Sort

Best Case: The best case for Insertion Sort occurs when the data is already sorted. The table shows that the time is 0 ms for sorted data across all set sizes, indicating a very efficient performance.

Average Case: For random data, the time increases with the set size. For example, with 10,000 elements, it takes 253 ms, while with 25,000 elements, it takes 1486 ms.

Worst Case: The worst case occurs with reverse-sorted data, where the time increases significantly. For instance, it takes 479 ms for 10,000 elements and 2928 ms for 25,000 elements.

Conclusion: Insertion Sort is efficient for small or nearly sorted data sets but performs poorly with larger or reverse-sorted data due to its O(n^2) time complexity.

2. Merge Sort

Best Case, Average Case, and Worst Case: Merge Sort's time complexity is O(n log n), which is consistent across different data types. The time increases linearly with the set size but is relatively unaffected by the initial data order.

For 10,000 elements, it takes 16-17 ms, while for 25,000 elements, it takes 40-43 ms.

Conclusion: Merge Sort is a stable and efficient choice for large datasets and works well with both random and ordered data due to its consistent time complexity.

3. Quick Sort

Best Case: Quick Sort performs best with random data, where the time is minimal. For example, it takes 5 ms for 10,000 elements and 11 ms for 25,000 elements.

Worst Case: Quick Sort's worst-case performance occurs with sorted or reverse-sorted data, where it exhibits O(n^2) time complexity. For example, it takes 1859 ms for 10,000 sorted elements and 11302 ms for 25,000 sorted elements.

Conclusion: Quick Sort is generally efficient, especially for random data. However, it can perform poorly with already sorted or reverse-sorted data unless optimizations like randomized pivot selection are applied.

Final Thoughts

Best for Small/Nearly Sorted Data: Insertion Sort is optimal for small or nearly sorted datasets.

Best Overall Performance: Merge Sort provides consistent performance across all data types and sizes.

Best for Random Data: Quick Sort is efficient for random data but needs optimization for sorted data to avoid worst-case scenarios.