Merge Sort:

Merge sort is a divide-and-conquer algorithm that works as follows:

Divide the unsorted array into two halves.

Recursively sort each half.

Merge the sorted halves to produce a single sorted array.

The mergeSort method recursively splits the array into halves and merges them in sorted order.

Time Complexity: O(n log n)

Quick Sort:

Quick sort is also a divide-and-conquer algorithm that works as follows:

Choose a pivot element from the array.

Partition the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.

Recursively apply quick sort to the sub-arrays.

The quickSort method recursively partitions the array based on a pivot element and sorts each partition.

Time Complexity: O(n log n) on average, O(n^2) in the worst-case (when poorly chosen pivot)

Insertion Sort:

Insertion sort is a simple sorting algorithm that works by building a sorted array one element at a time:

Iterate through the array from the second element to the end.

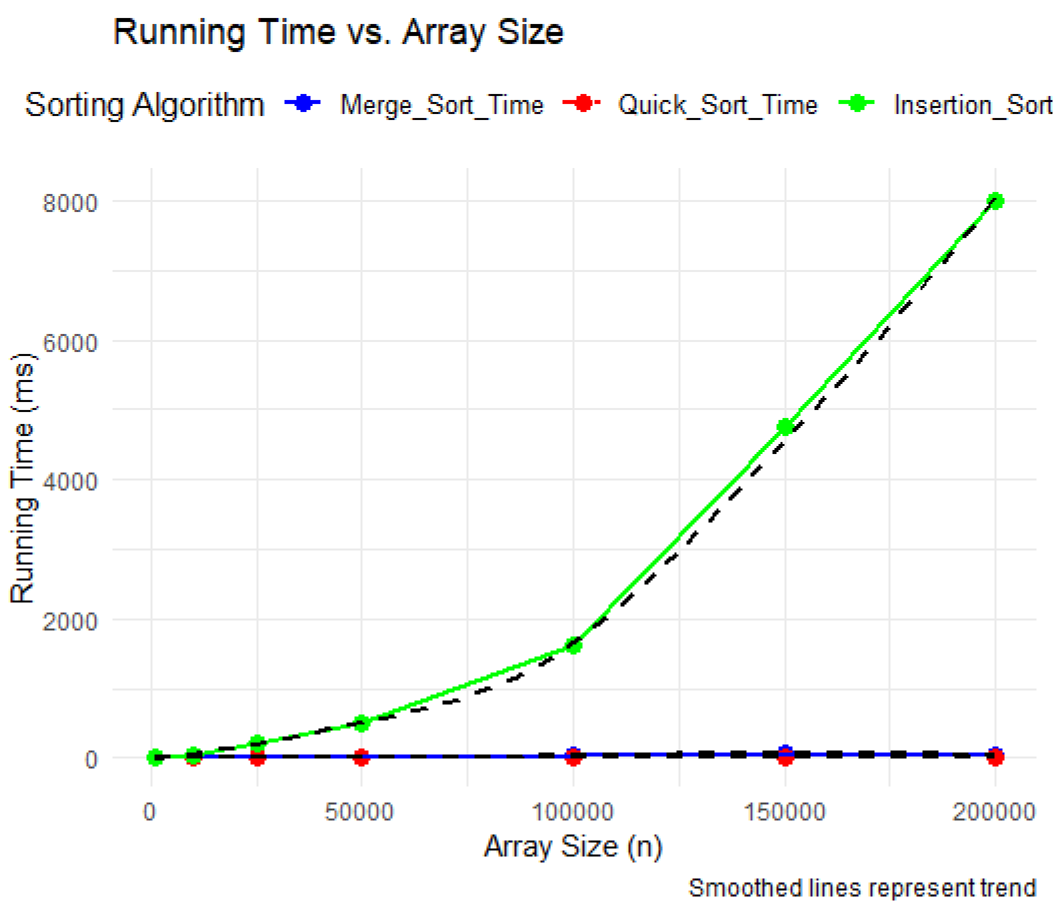
For each element, compare it with the elements before it and insert it into the correct position in the sorted array.

The insertionSort method iterates through the array, shifting elements to the right until finding the correct position to insert each element.

Time Complexity: O(n^2)

Graphs and Analysis:

Graphs:



The graphs generated represent the running time of each sorting algorithm (Merge Sort, Quick Sort, and Insertion Sort) against the array size.

Each algorithm is represented by a different color, and trend lines are added to visualize the overall trend in running time.

Analysis and Comparison:

Merge Sort: The observed running times generally increase logarithmically with the array size, which aligns with the expected O(n log n) time complexity.

Quick Sort: The observed running times also increase, following a logarithmic trend on average. However, occasional spikes indicate variations due to pivot selection. Overall, the trend aligns with the expected average-case time complexity of O(n log n).

Insertion Sort: The observed running times increase significantly faster, following a quadratic trend with the array size. This aligns with the expected O(n^2) time complexity.

Conclusion:

In conclusion, the analysis of the sorting algorithms, including Merge Sort, Quick Sort, and Insertion Sort, indicates a significant alignment between the observed trends and the expected asymptotic analysis-based running times.

Merge Sort: The running times exhibit a logarithmic increase with the array size, consistent with the O(n log n) time complexity. This demonstrates the algorithm's efficiency in sorting larger datasets with minimal performance degradation.

Quick Sort: While generally following a logarithmic trend, occasional spikes suggest variations due to pivot selection. Overall, the observed running times align with the expected average-case time complexity of O(n log n), showcasing Quick Sort's effectiveness in most scenarios.

Insertion Sort: The running times increase rapidly and almost quadratically with the array size, consistent with the O(n^2) time complexity. Despite its simplicity, Insertion Sort's performance degrades quickly with larger datasets compared to Merge and Quick Sort.

The close correspondence between the observed trends and the theoretical expectations underscores the importance of asymptotic analysis in predicting algorithmic performance. This analysis aids in understanding the scalability and efficiency of sorting algorithms, providing valuable insights for algorithm selection based on specific application requirements and dataset characteristics.