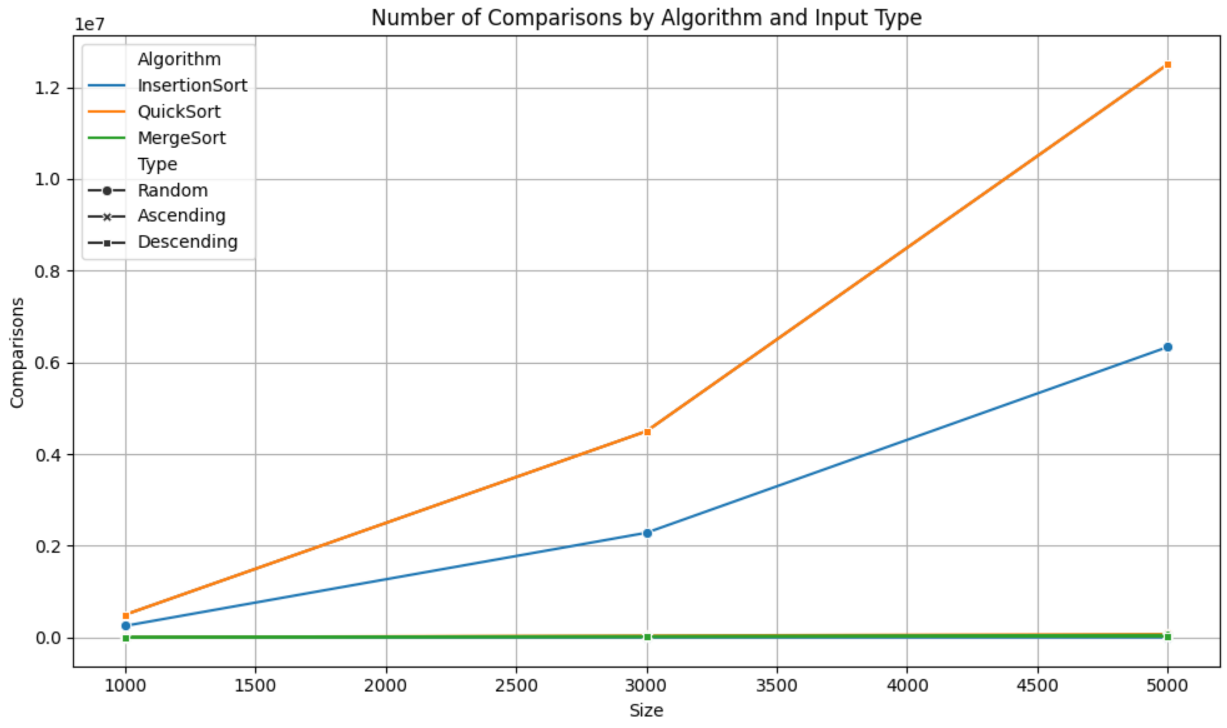
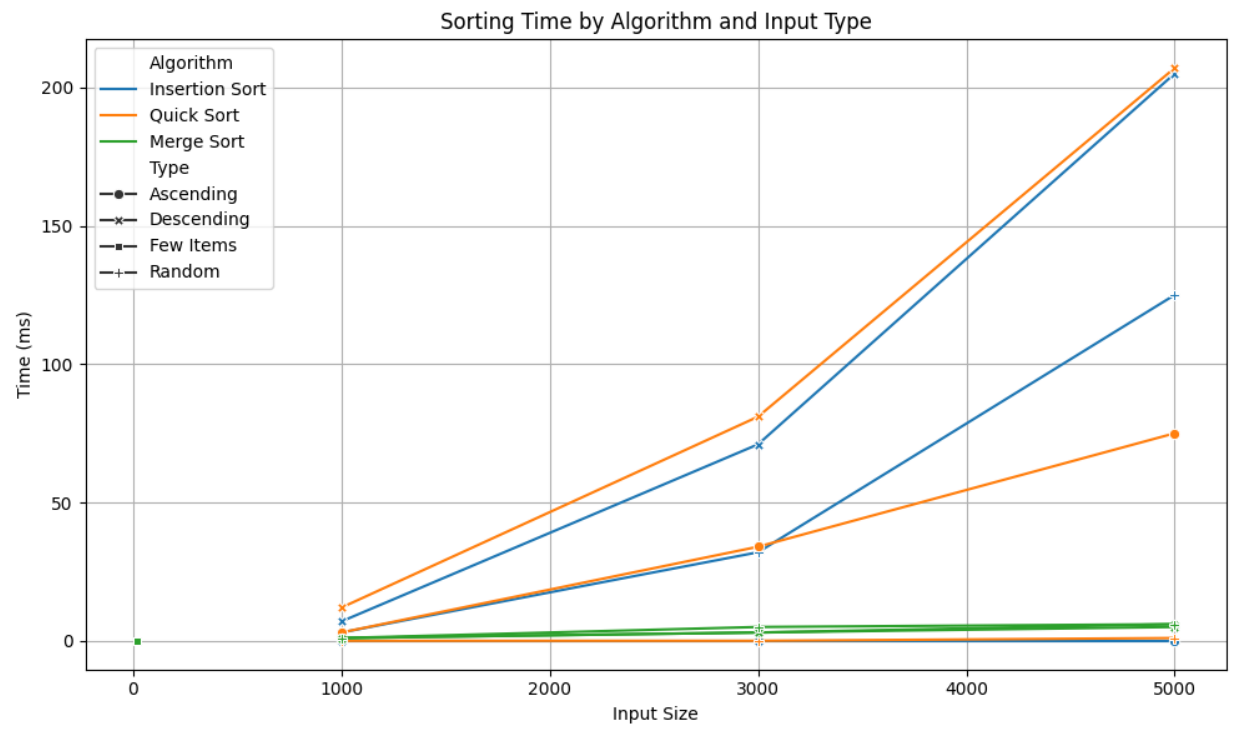
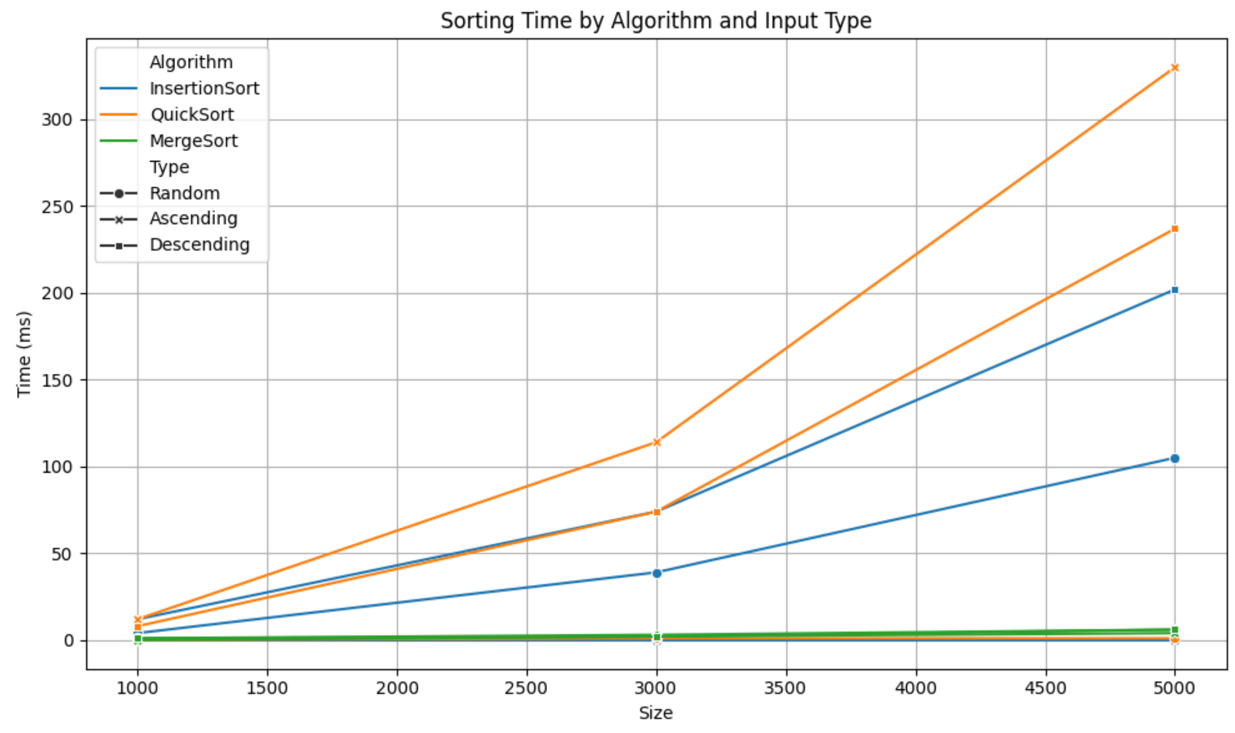
1. Graphs:

Comparison-count per size graph.





Sorting Time by input size for each sorting algorithm.



The time comparison.

2. Analysis:

Algorithm Efficiency by Time and Operations

Insertion Sort

- Best Case (Ascending list): Nearly instant (0ms) with minimal comparisons (n - 1). As expected, Insertion Sort performs best when data is already sorted.
- Worst Case (Descending list): Time and comparisons grow drastically. For 5000 items, ~12.5 million comparisons were made, confirming its $O(n^2)$ worst-case behavior.
- Few Items: Due to low overhead, it's very fast and efficient for small data sets (e.g., 20 or 30 items), often outperforming more complex algorithms.

Quick Sort

- Best Case (Random list): Performs very well, completing in 0–1ms on small to medium sizes. Even on large lists (5000 items), time is typically under 10ms.
- Worst Case (Descending or Ascending): Poor pivot choices result in $O(n^2)$ comparisons. In sorted inputs, comparisons ballooned to over 12 million, and time went up to 200+ ms for 5000 items.
- Key Insight: Pivot strategy is critical. Without randomized pivoting or median-of-three, Quick Sort can degrade significantly on sorted data.

Merge Sort

- Consistent Performer: Time increases smoothly with input size, regardless of order. Even in the worst case, it completed sorting in under 7ms for 5000 items.
- Operations: Slightly more comparisons than Quick Sort on random data, but far fewer than Quick Sort on already sorted inputs.
- Key Insight: The $O(n \log n)$ time is stable because it divides the problem regardless of data order.

3. Reflection:

- **Insertion Sort** mimics manual sorting: one item at a time. Fastest when data is nearly sorted but unscalable for large, unsorted inputs.
- **Quick Sort** is like intelligent decision-making — fast with good decisions (pivot), but poor strategy can make it inefficient.
- **Merge Sort** is systematic and machine-like: divides tasks evenly and merges consistently. Its performance is **reliable** and predictable.

Trade-Offs & Optimizations

Scenario	**Best Algorithm**	**Why?**
Small dataset (few items)	Insertion Sort	Simple, minimal overhead
Random large dataset	Quick Sort	Fast on average if pivot is well-chosen
Already sorted data	Insertion or Merge Sort	Quick Sort suffers due to poor pivot
Consistency needed	Merge Sort	Always O(n log n), no worst-case spike

Conclusion:
There's no single best algorithm for all cases.

- **Insertion Sort** is ideal for tiny or nearly-sorted data.
- **Quick Sort** is fastest on average but risky on sorted inputs without pivot optimization.
- **Merge Sort** It divides tasks evenly, ensuring each part is small and manageable. This makes it reliable and consistent even with large or disordered data, though at the cost of extra space for merging.