Empirical Analysis of Sorting Algorithms

(Worst-Case Scenario)

Introduction:

This report presents an empirical analysis of four sorting algorithms: **Bubble Sort**, **Selection Sort**, **Insertion Sort**, and **Merge Sort**, implemented in **C++ language**. The objective is to study how their execution time grows as input size increases, especially under **worst-case scenarios**.

In sorting, the *worst-case* means the input is arranged in such a way that the algorithm takes the maximum number of steps. For all algorithms used (except Merge Sort), the worst-case happens when the input is in **reverse order** (e.g., 5, 4, 3, 2, 1). For Merge Sort, worst-case and average-case are the same because of its divide-and-conquer structure.

➢ Methodology:

Each sorting algorithm was implemented **manually in separate C++ files** to maintain full control over logic and ensure fair testing. The arrays used were:

- `Arr1` = {5, 4, 3, 2, 1}
- `Arr2` = {10, 9, ..., 1}
- `Arr3` = {50, 49, ..., 1}
- `Arr4` = {100, 99, ..., 1}

These arrays represent increasing input sizes, all in worst-case order (descending).

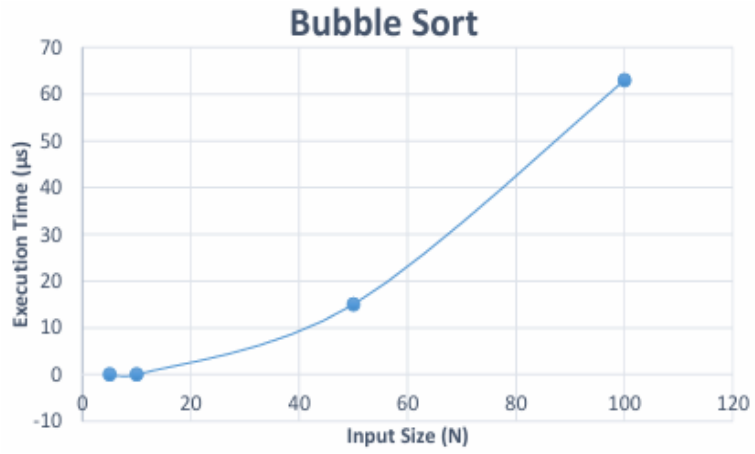
➢ Results & Graph:

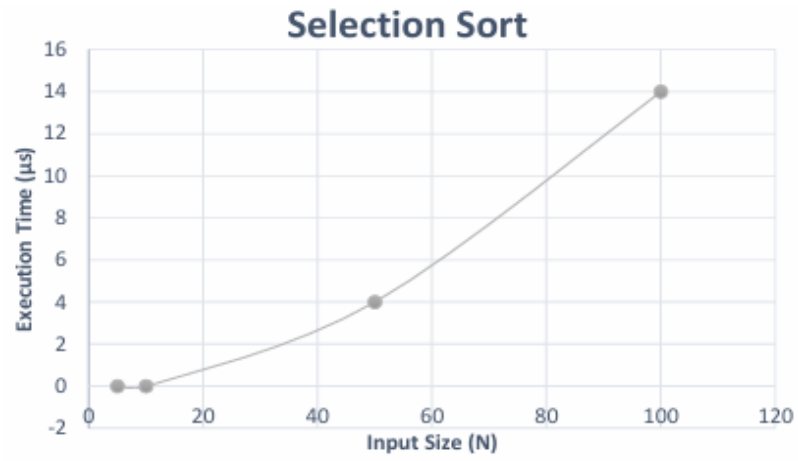
Average execution times (in **µs**) across each input size for all sorts are summarized below:

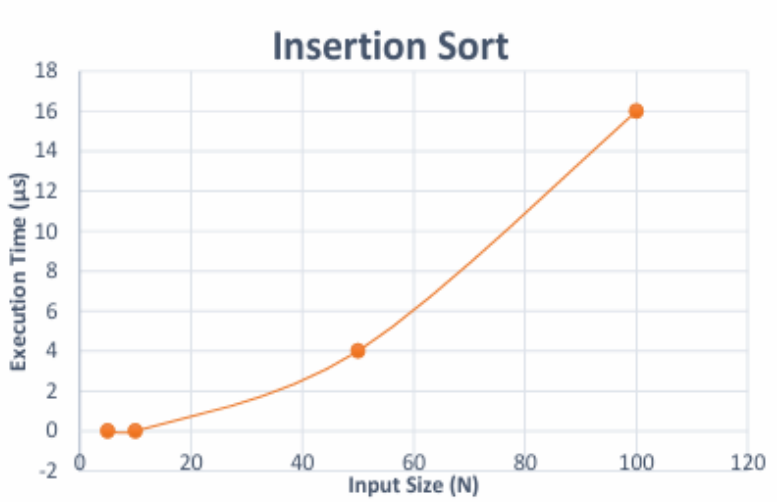
Input Size	*Bubble sort*	*Selection sort*	*Insertion sort*	*Merge sort*
5	0	0	0	0
10	0	0	0	0
50	16	4	4	3
100	57	14	16	5

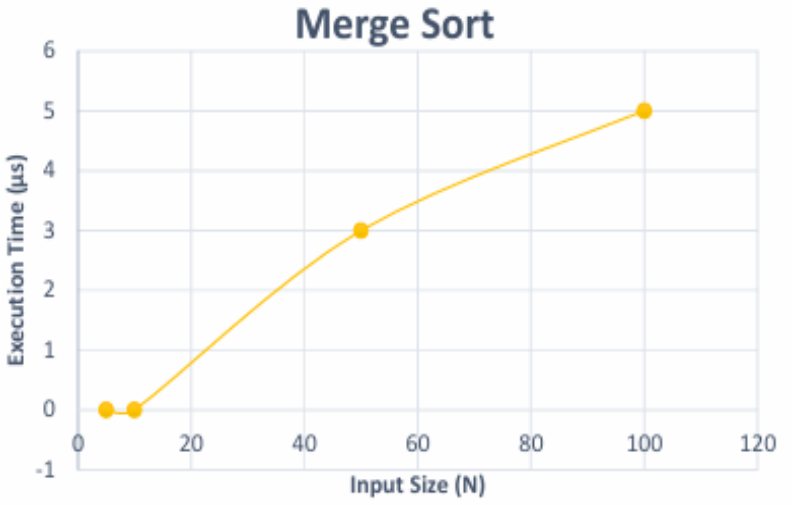
For input sizes 5 and 10, execution time shows **0 microseconds**. This is because these arrays are too small, and the sorting completes so fast that the average time rounds down to zero

Graphical plots were created for each algorithm individually (4 graphs) **and then** a combined comparison (1 graph).

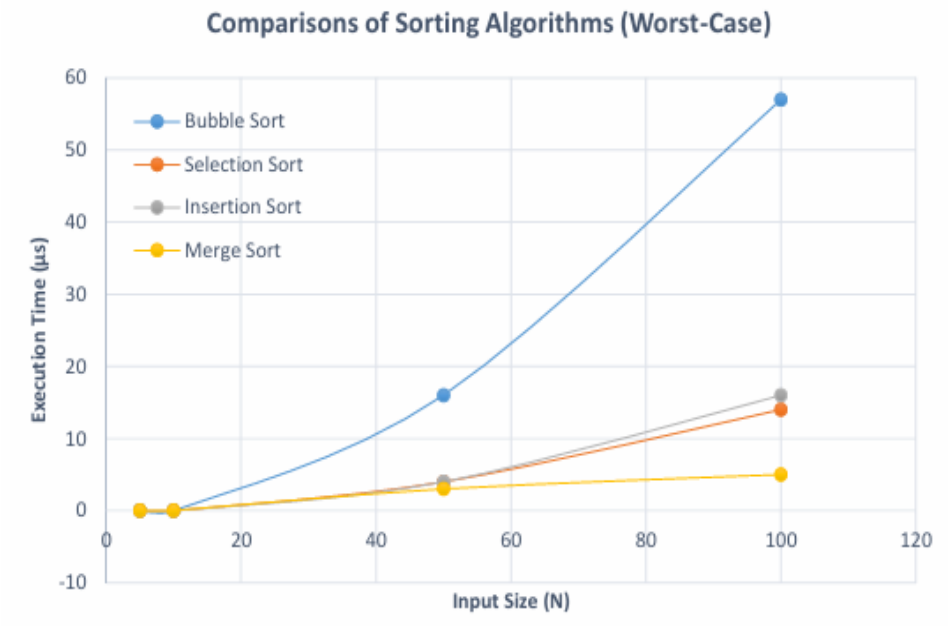








Combined Graph:



➢ Analysis:

The results closely match theoretical time complexities:

- **Bubble Sort**: Time grew rapidly with input size, confirming its **$O(n^2)$** complexity in worst-case.
- **Selection Sort**: Also showed a quadratic pattern, matching its **$O(n^2)$** behavior.
- **Insertion Sort**: Gradual increase confirms **$O(n^2)$** in worst-case.
- **Merge Sort**: Very efficient; time grew slowly, confirming its **O(n log n)** complexity.

These observations demonstrate that:

- **Merge Sort** is the most efficient in all cases.
- **Bubble Sort** is the least efficient, especially as input grows.
- Execution time is influenced by not just algorithm logic, but also compiler performance, hardware, and system load. Therefore, times can vary slightly between different environments and languages.

Conclusion

This empirical study validates the theoretical time complexities of common sorting algorithms under worst-case conditions. The practical results emphasize the importance of choosing the right algorithm for larger datasets and highlight the performance gap between basic quadratic sorts and optimized divide-and-conquer algorithms like Merge Sort.