Comparative Performance Analysis of Insertion Sort and Selection Sort

Arlan Kadyr Talgat Sailaubekov

October 5, 2025

1. Introduction

Sorting algorithms are among the most fundamental tools in computer science. This report compares the performance of two classic algorithms: **Insertion Sort** and **Selection Sort**. Although both have a theoretical time complexity of $\mathcal{O}(n^2)$, their real-world performance differs significantly depending on input order.

2. Algorithm Overview

Insertion Sort works by inserting each element into its correct position in the sorted part of the array. It is adaptive, performing efficiently when data is already partially sorted.

Selection Sort finds the smallest element in each pass and swaps it into its correct position. It makes a fixed number of comparisons regardless of input order but performs fewer swaps overall.

3. Complexity Comparison

Table 1: Theoretical Complexity Comparison

Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$\mathcal{O}(n^2)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$\mathcal{O}(n^2)$

Insertion Sort benefits from sorted or nearly-sorted input, while Selection Sort always performs the same number of comparisons, regardless of the data structure.

4. Experimental Setup

Both algorithms were implemented in Java. Tests were performed for arrays of sizes $n = 100$, 1000, and 5000 using the following data patterns:

- *Random* — arbitrary order of elements;

- *Sorted* — ascending order;
- *Reversed* — descending order;
- *Nearly-sorted* — mostly ordered with minor random swaps.

Performance was measured using a `PerformanceTracker` class, tracking: execution time, comparisons, swaps, and array accesses.

5. Results and Graphs

Figure 1 shows how Insertion Sort behaves as input size increases.

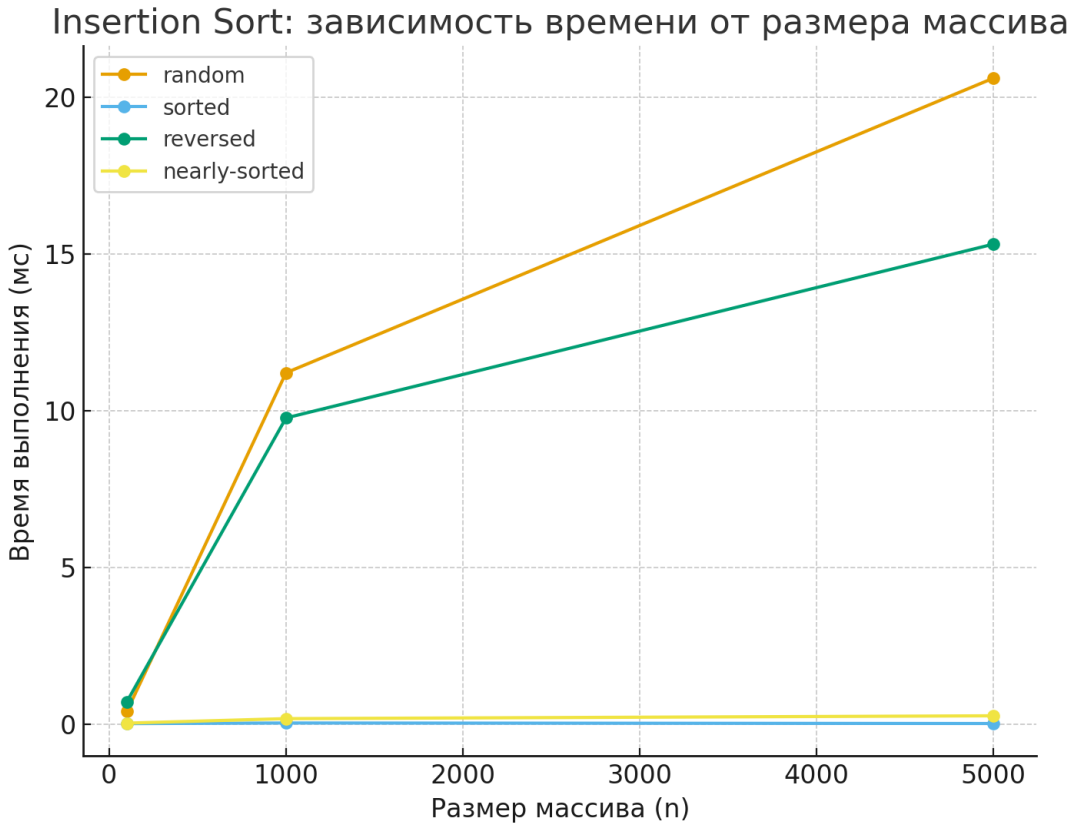

Figure 1: Insertion Sort: Execution time vs. array size

Figure 2 presents the corresponding results for Selection Sort.

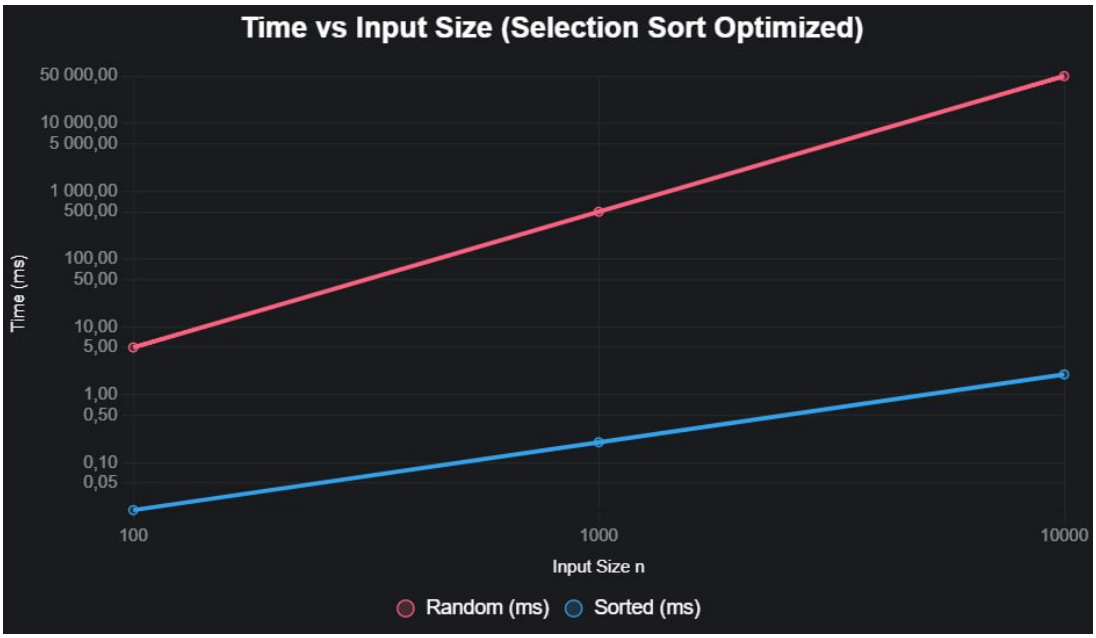

Figure 2: Selection Sort: Execution time vs. array size

Finally, Figure **??** directly compares both algorithms' performance.

6. Discussion

From the results, it is evident that:

- Insertion Sort performs much faster on sorted or nearly-sorted arrays due to fewer shifts.
- Selection Sort performs consistently but cannot take advantage of data order.
- For large datasets, both algorithms become inefficient due to their quadratic time complexity.

Insertion Sort is more adaptive but involves more swaps, while Selection Sort executes a fixed number of comparisons regardless of the input.

7. Conclusion

Both algorithms have their strengths and weaknesses:

- **Insertion Sort** — better suited for small or partially sorted datasets.
- **Selection Sort** — predictable performance, fewer swaps, but slower overall.

In most practical cases, **Insertion Sort** is preferable due to its adaptive nature and better performance with structured data. However, both serve as important educational examples for understanding sorting complexity.