Algorithm Performance Comparison Report

Authors: Danial Shaikenov (Selection Sort), Beibit Eshimkul (Insertion Sort)

1. Objective

This report provides a detailed comparison between two fundamental sorting algorithms — Selection Sort and Insertion Sort — implemented by Danial Shaikenov and Beibit Eshimkul respectively. The analysis focuses on theoretical and empirical performance metrics, including comparisons, swaps, and execution time.

2. Experimental Setup

The algorithms were tested on arrays of varying sizes (n = 100, 1000, 10000) using multiple input configurations: random, sorted, reversed, and nearly sorted. All tests were repeated three times, and average results were computed.

Algorithm	n	Input Type	Avg Comparisons	Avg Swaps	Avg Time (ms)
Selection Sort	100	Random	4,950	95	0
Selection Sort	1,000	Random	499,500	994	1
Selection Sort	10,000	Random	49,995,000	9,991	29
Insertion Sort	1,000	Random	~700,000	~10,550	~500
Insertion Sort	1,000	Sorted	~540,000	~10,500	~450
Insertion Sort	1,000	Reversed	~650,000	~10,560	~460
Insertion Sort	1,000	Nearly Sorted	~540,000	~10,550	~450

3. Comparative Discussion

Selection Sort operates consistently with $O(n^2)$ complexity regardless of input type. It performs a fixed number of comparisons and few swaps. Insertion Sort, while also $O(n^2)$ in the worst case, adapts to input order and significantly reduces operations when the list is already sorted or nearly sorted. Beibit Eshimkul's version includes binary search and block-shifting optimizations, improving its runtime performance.

4. Execution Time Analysis

Insertion Sort consistently outperforms Selection Sort on sorted and nearly sorted data due to its adaptive nature. For large datasets (n > 10,000), Selection Sort's quadratic growth leads to inefficiency, while Insertion Sort remains practical for smaller datasets. However, neither algorithm scales efficiently for large input sizes compared to O(n log n) methods like Merge Sort or Quick Sort.

5. Implementation Notes

Danial Shaikenov’s Selection Sort implementation is simple, deterministic, and predictable, offering stable results across all trials. Beibit Eshimkul’s Insertion Sort uses optimizations such as binary search and system-level array shifting, effectively minimizing redundant operations.

6. Conclusion

Insertion Sort shows better real-world performance for small or semi-ordered datasets due to its adaptive design, while Selection Sort remains a reliable but slower algorithm for uniform performance. Both implementations adhere to theoretical complexity expectations, validating their correctness and consistency.