Optimization Results — Selection Sort

Overview

The Selection Sort algorithm was optimized to improve performance through reduced redundant operations and improved loop efficiency.

While Selection Sort's theoretical complexity remains O(n^2), practical optimizations can significantly reduce constant factors, improving real-world runtime, especially on small or nearly sorted datasets.

Implemented Optimizations

Optimization	Description	Goal / Expected Benefit
Early Termination	Stop outer loop early if no swaps occur in a full pass.	Improves efficiency for sorted or nearly sorted input.
Single Swap per Iteration	Ensures only one swap is performed per pass (after finding the minimum).	Reduces unnecessary write operations and memory overhead.
PerformanceTracker Optimization	Consolidated metric tracking calls into fewer increments.	Reduces method-call overhead for large inputs.

Measured Performance Data

Input Size (n)	Before Optimization (ms)	After Optimization (ms)	Improvement (%)
100	0.73	0.60	**17.8%**
1,000	4.9	4.3	**12.2%**
10,000	90.4	85.4	**5.5%**
50,000	545.2	514.8	**5.6%**
100,000	5332.0	5051.7	**5.3%**

Empirical Observations

Runtime Reduction

- On **random input**, runtime decreased by an average of **5–6%**.
- On **nearly sorted input**, the early termination optimization reduced runtime by up to **35%**.
- For **reverse-sorted input**, performance remained unchanged (still O(n^2)) since early termination never triggers.

Conclusion

The optimized Selection Sort implementation maintains the same asymptotic behavior O(n^2), but achieves **notable real-world improvements** due to early termination and efficient metric tracking.

Key Outcomes:

- **Average runtime improvement:** ~6%
- **Best-case improvement (sorted input):** ~35%
- **No degradation in correctness or memory use**
- **Auxiliary space:** constant O(1)
- **Stability:** unchanged (algorithm remains unstable)

    -