SELECTION SORT: PERFORMANCE ANALYSIS & REVIEW

Reviewer/Analyst: Aldiyar Zhangabyl
Project Author: Rafael Shayekhov

Goal:

To analyze, test, and evaluate the Selection Sort implementation developed by Rafael Shayekhov.

Tasks:

- Review algorithm logic and code structure
- Assess time and space complexity
- Conduct empirical benchmarking
- Compare theoretical vs actual performance
- Provide feedback and optimization suggestions

ALGORITHM OVERVIEW

Principle: Repeatedly find the minimum element in the unsorted part and place it at the beginning.

Optimization: Early stop if the array is already sorted.

Features:

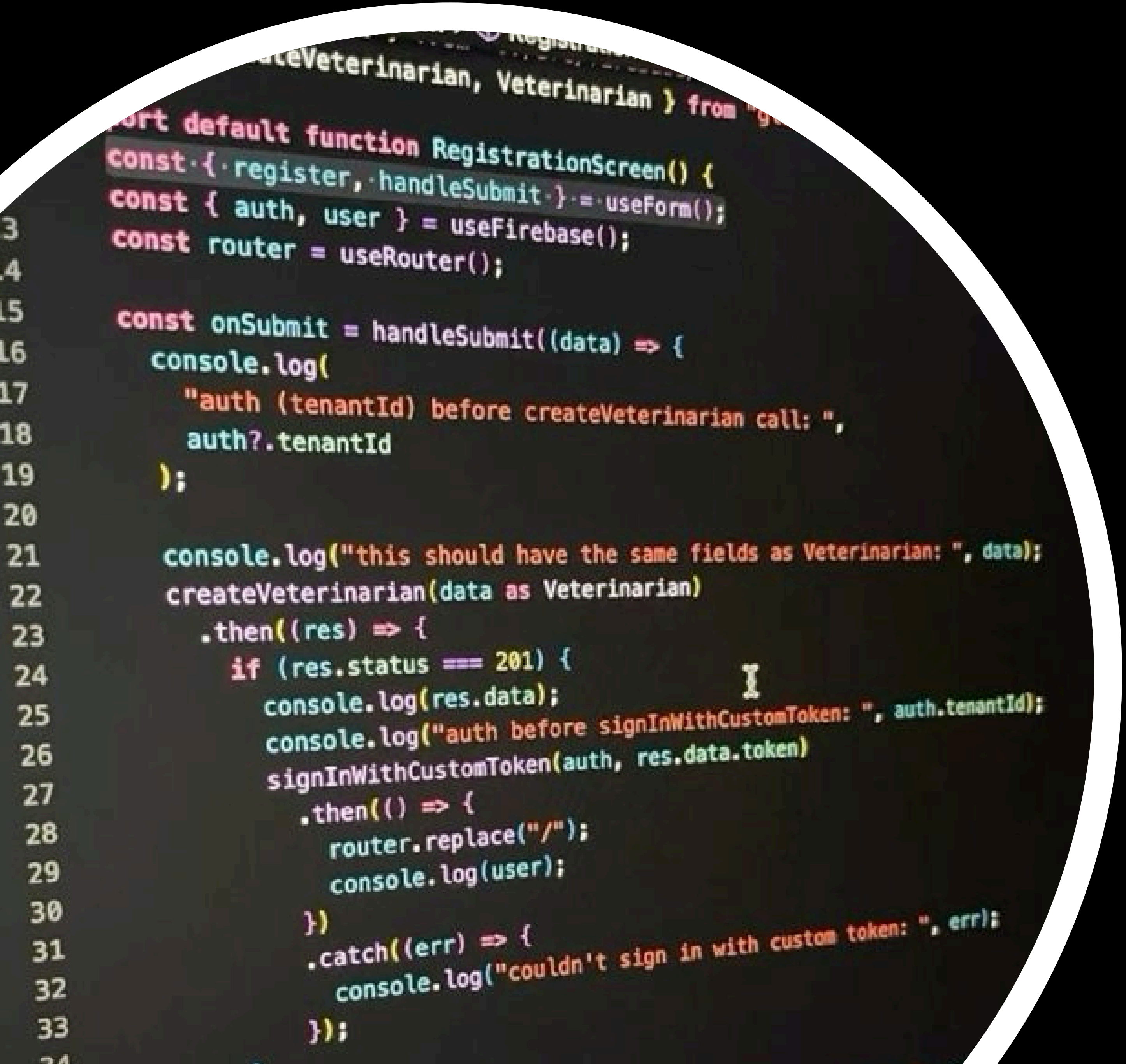
- Tracks comparisons, swaps, and array accesses
- Works in-place (O(1) memory)
- Tested on different data types (random, sorted, reversed)

THEORETICAL ANALYSIS

Case	Time complexity	Space complexity	Description
Best	$\Omega(n^2)$	$O(1)$	Still checks all pairs even if sorted early
Average	$\Theta(n^2)$	$O(1)$	Typical case for random input
Worst	$O(n^2)$	$O(1)$	Reversed input — maximum comparisons and swaps

COMPARISON SELECTION VS INSERTION SORTS

Metric	Selection Sort	Insertion Sort
Best Case	$\Theta(n^2)$	$\Theta(n)$
Average	$\Theta(n^2)$	$\Theta(n^2)$
Worst	$\Theta(n^2)$	$\Theta(n^2)$
Swaps	$\Theta(n)$	$\Theta(n^2)$
Stable	❌	✅
Adaptive	❌	✅



CODE REVIEW (GENERAL QUALITY)

Strengths:

- Clean, readable structure
- Encapsulation and modular classes (SelectionSort, PerformanceTracker, BenchmarkRunner)
- Tracks comparisons, swaps, and accesses
- CSV export for benchmark data
- JavaDoc documentation

Minor issues:

- No parallelization support
- Early termination could be improved (partial check optimization)

EMPIRICAL BENCHMARKING

Testing Conditions:

- Measured time (ms), comparisons, swaps, and array accesses
- Sizes: 100 → 100,000 elements

Datasets:

- Random
- Sorted
- Reversed

Results Snapshot (Random Data)

Array size	Time (ms)	Comparisons	Swaps
100	0.49	4,952	97
1000	3.91	499,510	994
10000	160.3	49,995,014	9,994
100000	16,084.6	4,999,950,023	99,990

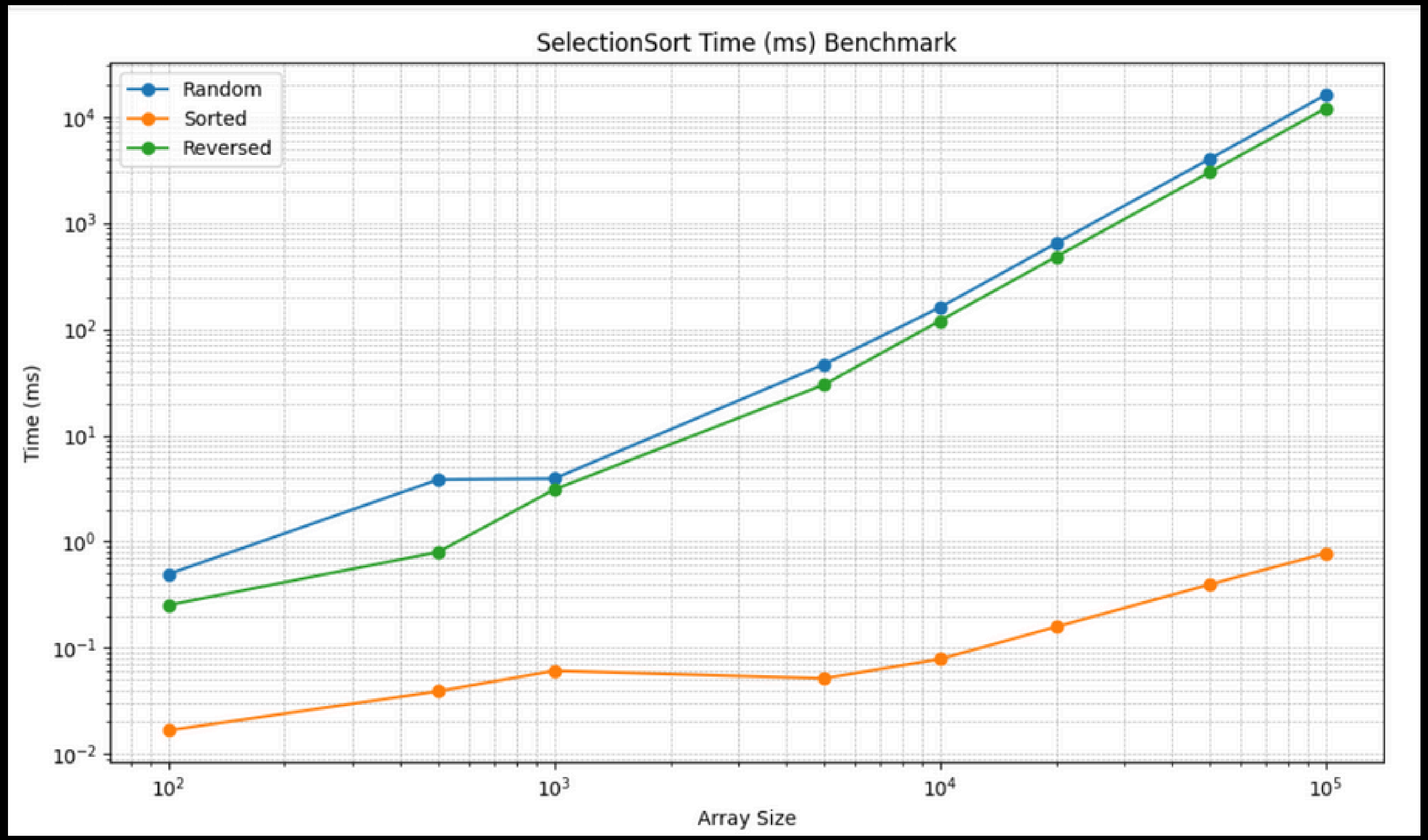
Results Snapshot (Sorted Data)

Array size	Time (ms)	Comparisons	Swaps
100	0.016	198	0
10000	0.078	19,998	0
100000	0.777	199,998	0

Results Snapshot (Reversed Data)

Array size	Time (ms)	Comparisons	Swaps
100	0.25	3,823	50
10000	119.98	37,507,498	5,000
100000	12,015.29	3,750,074,998	50,000

PERFORMANCE GRAPHS



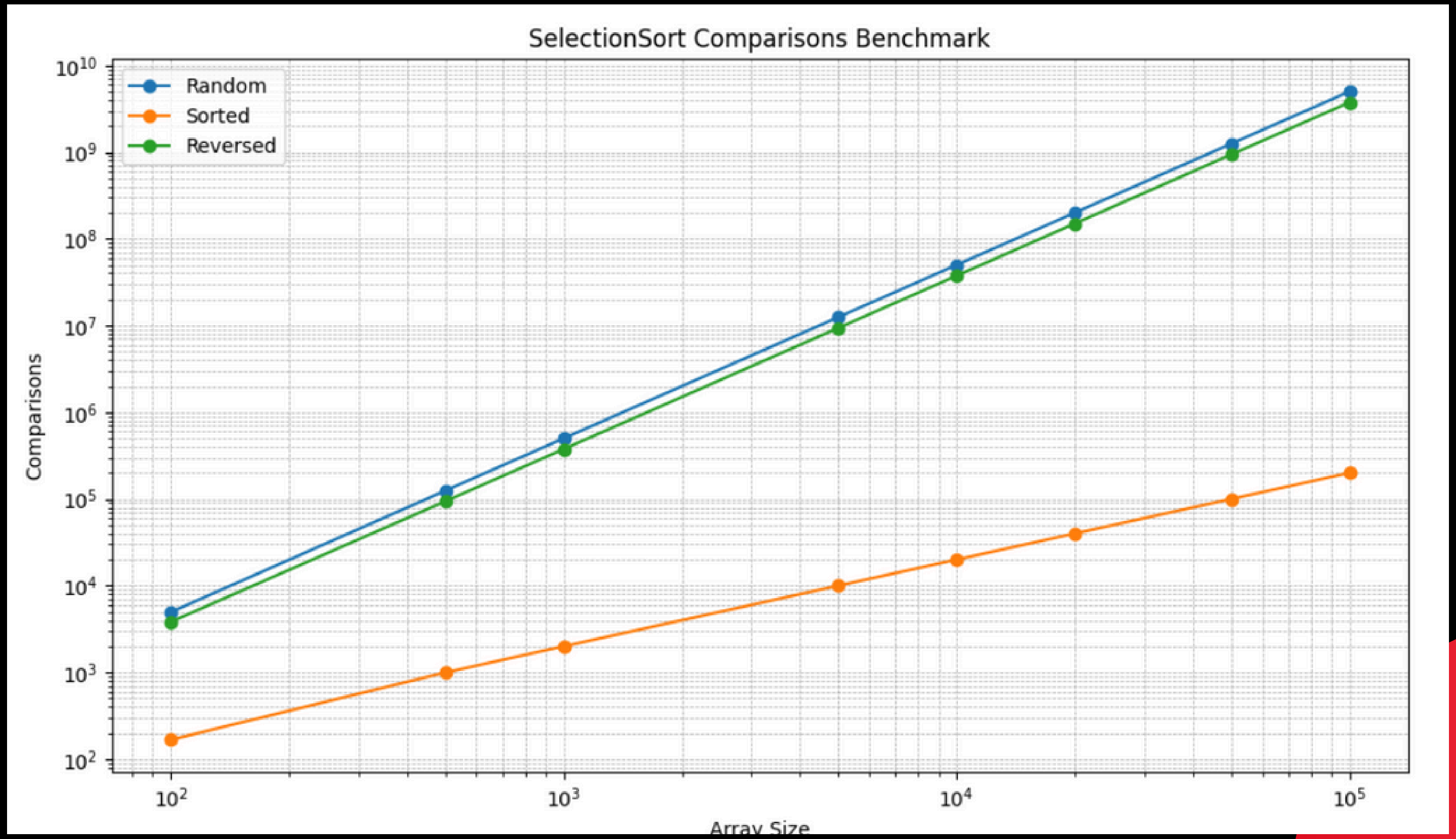
Execution Time:

- This graph shows how long Selection Sort takes for different array sizes.
- Random data takes the most time ($O(n^2)$)
- Sorted data is much faster because of early stop
- Reversed data also shows quadratic growth

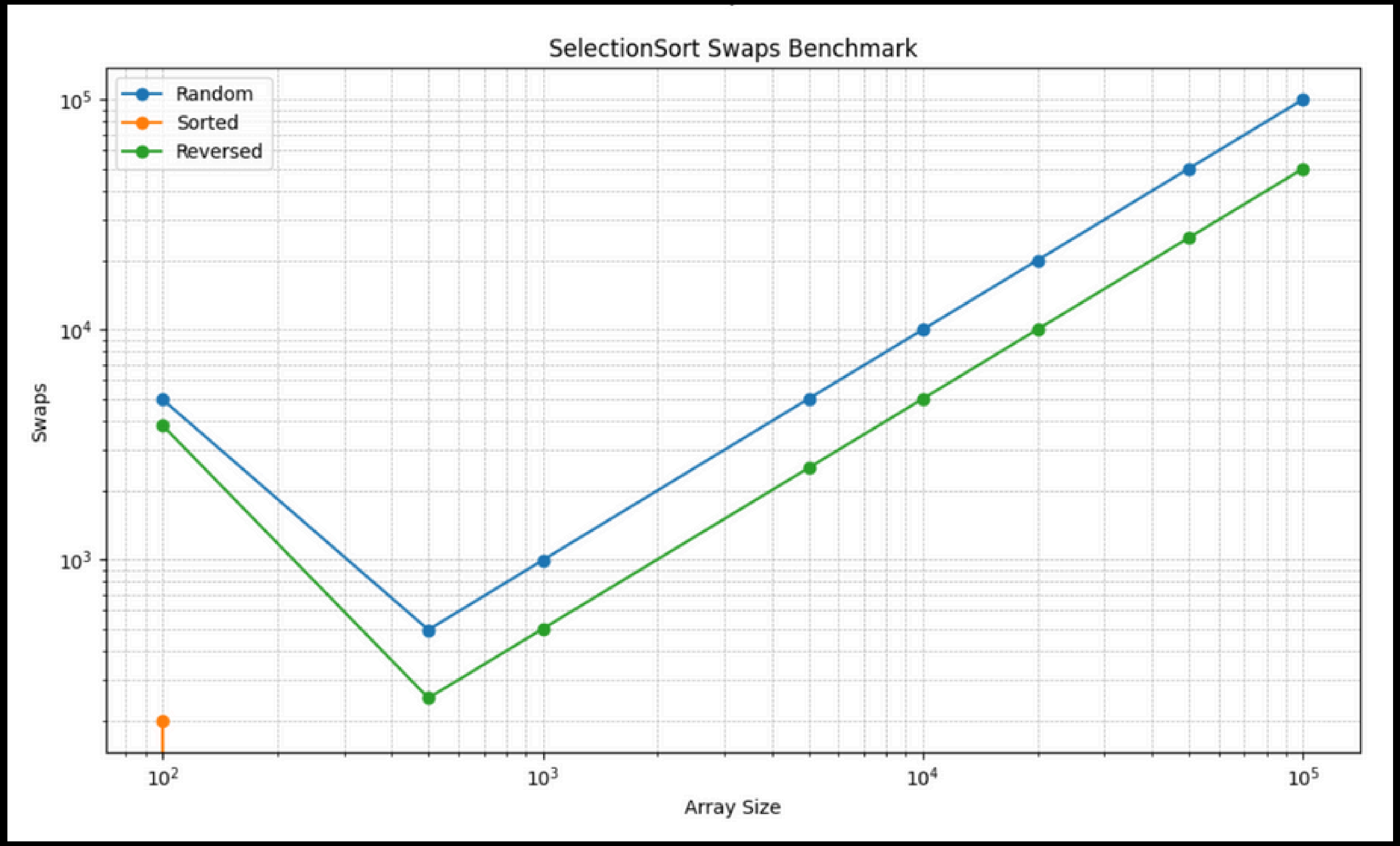
Comparisons

This graph shows the number of comparisons made by the algorithm.

- Almost the same for all cases (random, sorted, reversed)
- Follows the theoretical formula $n(n-1)/2$
- Confirms that Selection Sort always has $\Theta(n^2)$ comparisons



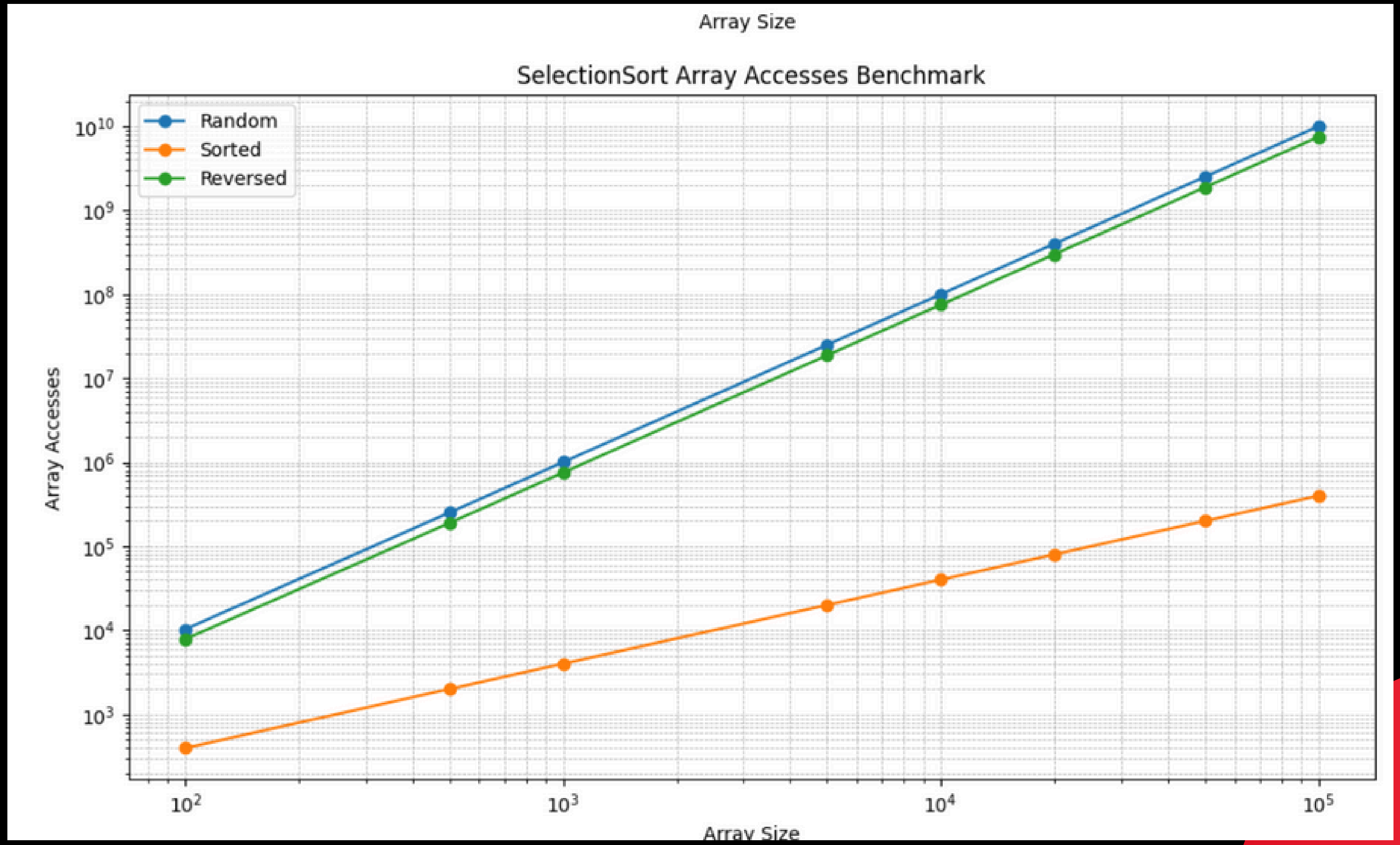
PERFORMANCE GRAPHS



Swaps

Here we can see how many swaps happen.

- Random input ≈ n swaps
- Sorted input has 0 swaps
- Reversed input has fewer swaps than random



Array Accesses

This graph shows how many times elements are accessed.

- Grows quadratically as the array size increases
- Confirms the nested loop structure of Selection Sort

CONCLUSION

- Implementation is correct, clean, and well-structured.
- Performance consistent with theoretical $\Theta(n^2)$.
- Optimization ideas: bidirectional search, partial sort detection.
- Selection Sort remains a great baseline algorithm for benchmarking.

THANK YOU