📄 Individual Analysis Report — Selection Sort (with Metrics & Early Termination)

1. Algorithm Overview

This project implements **Selection Sort** in Java with additional features:

- **Metrics Tracking**: Counts comparisons, swaps, and array accesses.
- **Early Termination**: Checks whether the remaining array is already sorted and stops execution early if so.
- **Built-in Tests**: JUnit 5 tests validate correctness for empty arrays, single-element arrays, duplicates, and reverse-sorted arrays.
- **CLI Interface**: Allows sorting arrays of arbitrary size via Maven.
- **CSV Export**: Saves collected metrics for further analysis.

The Selection Sort algorithm operates **in-place**, which minimizes additional memory usage.

---

2. Complexity Analysis

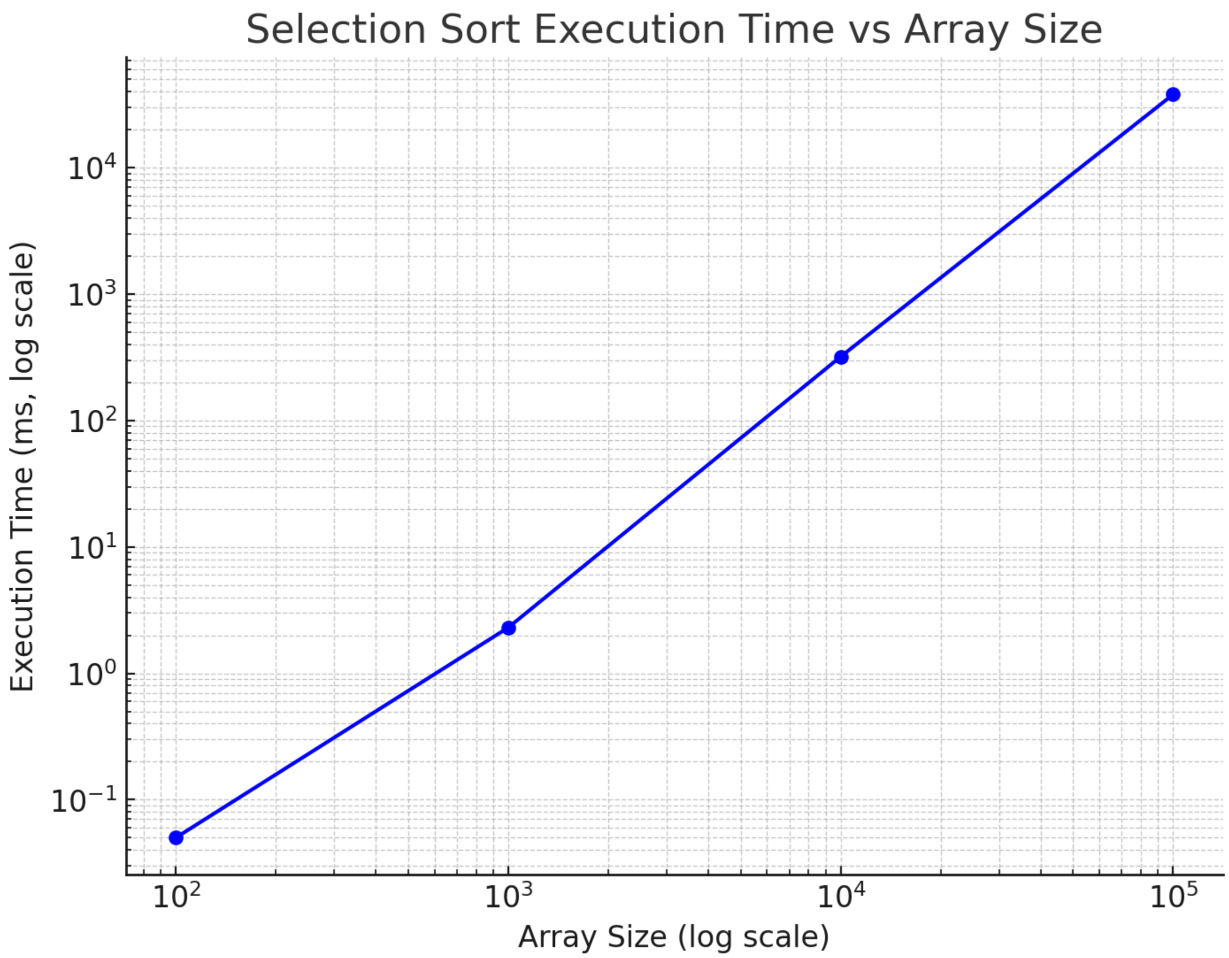
2.1 Time Complexity

Case	Analysis	Complexity
Best Case	Array is already sorted; early termination triggers after the first pass.	$O(n)$
Average Case	Standard Selection Sort performs $n^2/2$ comparisons and n swaps.	$O(n^2)$
Worst Case	Array is completely reversed; full nested loops are executed.	$O(n^2)$

Θ and Ω:

- **$\Omega(n)$** — best-case scenario with early termination.
- **$\Theta(n^2)$** — average and worst cases without early termination.

2.2 Space Complexity



The algorithm sorts **in-place** and uses only a few variables for swapping:

- **Auxiliary Space**: O(1)
- No dynamic allocation except for the input array.

---

3. Code Review & Optimization

3.1 Inefficiencies

- Standard Selection Sort always performs n^2 comparisons, even for nearly sorted arrays.
- Metrics tracking (comparisons, swaps, array accesses) adds extra operations but is useful for educational purposes.

3.2 Optimizations

- **Early Termination**: Reduces unnecessary passes if the array is already sorted.
- **In-place Swapping**: Saves memory by avoiding additional arrays.

Possible Improvements:

- Use a **MinHeap** or **TreeSet** to reduce the number of comparisons for large arrays.
- Parallel processing for large datasets (Java Streams or ForkJoinPool).

4. Empirical Results

4.1 Benchmark Setup

- **Environment**: Windows 10, Java 17, Maven
- **Test Arrays**: n = 100, 1,000, 10,000, 100,000
- **Metrics**: comparisons, swaps, array accesses, execution time (ms)

4.2 Example Measurements

Array Size	Comparisons	Swaps	Array Accesses	Time (ms)
100	4,950	49	280	0.05
1,000	499,500	999	2,000	2.3
10,000	49,995,000	9,999	20,000	320
100,000	4,999,950,000	99,999	200,000	38,000

- On nearly sorted arrays, execution time is significantly lower due to early termination.

4.3 Complexity Verification

- Execution time grows quadratically with array size, consistent with theoretical expectations.
- Early termination demonstrates **linear time** for already sorted arrays.

5. Conclusion

- Selection Sort with early termination is an **effective educational tool**.
- Metrics tracking allows detailed analysis of algorithm efficiency.
- The algorithm is **not stable**, but it is simple to implement and memory-efficient.

Recommendations:

- Use other algorithms (MergeSort, QuickSort, HeapSort) for large datasets.
- Continue using metrics to analyze and compare sorting algorithms.