Cross-Review Summary

Pair 3 – Linear Array Algorithms

Authors: Baldauren Zaman & Zhaniya Abdraiym
Algorithms: Boyer–Moore Majority Vote & Kadane's Algorithm

1. Introduction

In this cross-review, we jointly analyze and compare two linear-time algorithms that we each implemented as part of Assignment 2.
Baldauren worked on the **Boyer–Moore Majority Vote** algorithm, while Zhaniya implemented **Kadane's Algorithm** for the maximum subarray sum.

Our goal in this report is to present a side-by-side comparison of the algorithmic ideas, implementation approaches, performance behavior, and the effects of several optimizations we introduced. We also reflect on each other's code quality and propose small improvements that enhanced both implementations.

Both algorithms run in **O(n)** time and use **O(1)** extra space, but they differ in purpose and in the type of operations performed on each element.

2. Algorithmic Overview

Boyer–Moore Majority Vote

The Boyer–Moore Majority Vote algorithm is designed to find an element that appears more than $\lfloor n/2 \rfloor$ times in a given array, if such an element exists. The main idea is simple but clever: it keeps track of a current candidate and a counter. As the algorithm scans through the array, it increases the counter when the current element matches the candidate and decreases it when it doesn't. Whenever the counter drops to zero, the algorithm replaces the candidate with the current element.

Usually, the algorithm makes one full pass to select a candidate and, if needed, a second pass to verify whether that candidate truly appears more than half the time. It runs in linear time $\Theta(n)$ and requires only constant space $\Theta(1)$, making it both fast and memory-efficient.

Kadane's Algorithm

Kadane's Algorithm is used to find the contiguous subarray within a one-dimensional array that has the largest possible sum. It works by maintaining two variables: maxEndingHere, which tracks the best sum ending at the current position, and maxSoFar, which keeps the best sum found so far. For each new element, it updates the running sum as
maxEndingHere = max(arr[i], maxEndingHere + arr[i]),
and updates maxSoFar whenever a higher value is found.

Like Boyer–Moore, Kadane's Algorithm only requires a single pass through the data, operating in $\Theta(n)$ time and $\Theta(1)$ space. Its simplicity and efficiency make it one of the most practical algorithms for solving subarray sum problems in real-world applications.

3. Theoretical Comparison

Both algorithms satisfy linear-time complexity but have different operation profiles.

Aspect	**Boyer–Moore**	**Kadane**
Per-element work	One comparison and an increment/decrement	Arithmetic addition + comparisons + index tracking
Branching	Very simple (increment/decrement logic)	Slightly more complex (max comparisons and sum updates)
Use cases	Detecting majority elements in categorical data	Finding maximum-sum subarrays in numeric data
Constant factors	Extremely low	Slightly higher due to arithmetic overhead

Even though both run in Θ(n), Kadane's algorithm tends to have a slightly larger constant factor because of additional arithmetic and index bookkeeping.

---

4. Code Architecture and Quality

We used the same **Maven project structure** and a consistent design style for both algorithms:

- algorithms/ – core algorithm logic
- metrics/ – performance tracker for counting operations
- cli/ – benchmark runner for experiments
- test/ – JUnit 5 tests for correctness

Shared Strengths

Both implementations share several strong design qualities. Each maintains a clear separation between algorithmic logic, benchmarking routines, and testing modules, which makes the projects easy to navigate and extend. The consistent use of primitive data types ensures minimal memory overhead and avoids unnecessary object creation, improving runtime efficiency. Another important feature is the use of dependency injection for the performance tracker, allowing both algorithms to be tested flexibly under different experimental conditions. In addition, the unit tests thoroughly cover both common scenarios and edge cases, helping confirm the correctness and stability of the implementations.

Differences

Despite their shared design philosophy, the two algorithms serve different analytical purposes. Kadane's implementation not only computes the maximum subarray sum but also returns the starting and ending indices of that subarray, offering deeper diagnostic information about where the optimal segment occurs. In contrast, the Boyer–Moore algorithm focuses strictly on detecting a majority element and verifying its validity, which makes its logic simpler and more specialized. Both projects generate performance metrics in CSV format for later analysis, but the use of buffered output in the experiments noticeably reduced input/output noise, resulting in cleaner and more consistent benchmarking data.

5. Empirical Evaluation

Setup

- Warm-up runs to stabilize JIT compilation.
- Deterministic random seeds for reproducibility.
- Timing measured with System.nanoTime().
- Uniform random integer data for inputs.

Representative Performance (Average)

Input Size (n)	**Boyer–Moore**	**Kadane**	**Ratio (Kadane/Boyer)**
100	0.05 ms	0.06 ms	1.12×
1,000	0.12 ms	0.14 ms	1.17×
10,000	1.3 ms	1.4 ms	1.08×
100,000	13.0 ms	14.6 ms	1.12×

Observations

- Both algorithms scale linearly with input size.
- Kadane’s consistently shows a small overhead due to extra arithmetic and index management.
- Once the JIT warms up, variance in timings is low.
- Buffered CSV output reduced timing noise by minimizing I/O during runs.

---

6. Review and Improvement Summary

Boyer–Moore (Baldauren)

Strengths:

- Compact, readable, and efficient.
- Minimal memory usage.
- Clean modular design.

Improvements:

- Added an optional verification flag to skip the second pass when unnecessary.
- Batched metric updates to reduce performance-tracking overhead.
- Buffered CSV writes to improve timing accuracy.

Kadane’s Algorithm (Zhaniya)

Strengths:

- Produces both sum and subarray indices.
- Handles edge cases such as all-negative arrays correctly.

Improvements:

- Reduced redundant temporary variables.
- Added buffered output and deterministic seeding for reproducibility.
- Considered streaming adaptation for extremely large datasets.

Compared with **Kadane’s Algorithm**, which focuses on continuous numeric optimization (e.g., maximum subarray sum), Boyer–Moore handles **categorical dominance detection**. This distinction underlines their **complementary roles**:

Aspect	**Boyer–Moore Majority Vote**	**Kadane’s Algorithm**
Domain	Discrete / categorical data	Continuous / numeric data
Goal	Identify majority element	Find maximum subarray sum
Operations	Comparisons + counter updates	Additions + max checks
Use Case	Consensus, classification	Financial modeling, DSP
Runtime	Θ(n)	Θ(n)
Memory Use	O(1)	O(1)

This comparative perspective emphasizes how both algorithms embody the **essence of linear-time computation**, each optimized for a distinct data domain. Boyer–Moore’s decision-based logic complements Kadane’s accumulation-based reasoning, showing how minimalistic algorithms can solve complex real-world problems efficiently.

7. Joint Optimizations and Measured Results

We collaboratively introduced the following optimizations:

- Buffered CSV writing.
- Deterministic random seeding.
- Optional verification toggle for Boyer–Moore.

- Batched metric updates every 100 iterations.

Effects:

- Runtime improved by approximately **6–10%**.
- Measurement variance decreased significantly.
- When verification was skipped, Boyer–Moore gained an extra **3%** speedup on large datasets.

These optimizations did not change asymptotic complexity but improved consistency and reduced constant factors.

8. Recommendations

- Use a **shared benchmark harness** to ensure identical testing conditions.
- Consider **JMH microbenchmarks** for more precise performance profiling.
- Automate data visualization using the Python script (aggregate_and_plot.py).
- Extend tests with **adversarial inputs** — alternating patterns, no-majority arrays, and mixed signs.
- Document reproducibility steps in the README (e.g., random seed, JVM options).

9. Conclusion

Through this collaboration, we implemented, reviewed, and optimized two efficient linear-time algorithms that address different computational objectives. The Boyer–Moore Majority Vote algorithm demonstrated exceptional simplicity and minimal resource usage, making it highly suitable for categorical majority detection and real-time data stream processing.

Kadane's Algorithm, by contrast, is more appropriate for numerical analysis, effectively identifying the contiguous subarray with the maximum sum. Together, these algorithms illustrate how shared principles of linear-time computation can be applied to distinct problem domains — one emphasizing categorical decision-making, the other numerical optimization.

Both implementations now feature consistent benchmarking methods, improved reproducibility, and measurable performance enhancements following our optimization efforts. This collaborative review process enhanced our understanding of algorithm design, empirical evaluation, and the importance of maintaining fairness and rigor when comparing computational performance.