Algorithm Analysis Report — Boyer–Moore Majority Vote (Bekzat’s Algorithm)

Pair 3 — Student B
Name: Farkhad Imanbayev
Teammate: Bekzat Duesenbek (Student A)
Group: SE-2438

1. Algorithm Overview

The Boyer–Moore Majority Vote algorithm determines whether an array contains a majority element — an element that appears more than floor(n/2) times. It operates in a single linear pass and requires constant auxiliary space. The algorithm maintains a candidate element and a counter: when the counter is zero, the current element becomes the new candidate; otherwise, the counter is incremented if the same element is encountered or decremented otherwise. The final candidate, if any, is then verified through a second pass.

2. Correctness and Invariant

Invariant: After processing any prefix of the array, the counter equals the difference between the count of the candidate element and the count of all other elements “paired away”. The majority element, if it exists, can never be fully canceled. Therefore, after the first pass the candidate is a valid majority candidate. A second pass confirms whether it appears more than floor(n/2) times.

3. Complexity Analysis

• **Time Complexity:** Θ(n) for candidate selection and an additional Θ(n) for verification, resulting in

Θ(n) overall.

• **Space Complexity:** O(1), as only two variables (candidate and counter) are maintained.

This efficiency makes Boyer–Moore one of the most optimal deterministic algorithms for this problem.

4. Code Review and Implementation Style

The partner’s implementation follows the canonical Boyer–Moore pattern and is structurally sound.

Code readability is generally high due to clear variable names (e.g., candidate, count). However,

some areas can be improved:

- No explicit verification step in the main function (should count occurrences of the candidate).

- Lacks defensive handling for empty or null input arrays.

- No unit tests verifying edge cases (e.g., no majority, all identical elements).

- Missing inline comments describing invariant logic.

Adding these refinements would enhance maintainability and clarity for future reviewers.

5. Optimization Opportunities

The algorithm is already optimal in asymptotic terms. However, micro-optimizations include:

- Early exit during verification if count exceeds n/2.

- Using primitive types instead of boxed wrappers to reduce overhead.

- Merging both passes (selection and verification) in scenarios where streaming input is used,

though this may reduce code clarity.

6. Empirical Validation

To empirically validate Boyer–Moore's linear performance, multiple arrays of varying sizes were tested under two scenarios: (A) majority present and (B) no majority. The runtime increases linearly

with n, confirming the expected Θ(n) behavior. Both scenarios demonstrate identical asymptotic growth,

with minimal constant-factor differences due to verification overhead.

7. Comparison with Kadane's Algorithm

While both Kadane's and Boyer–Moore algorithms run in Θ(n) and use O(1) space, they differ conceptually.

Kadane's algorithm solves an optimization problem (finding the subarray with the maximum sum), while

Boyer–Moore addresses a decision problem (existence of a majority element). Both exemplify linear-time

algorithms using iterative invariants and constant memory, aligning with the assignment’s learning goals.

8. Conclusion

The Boyer–Moore Majority Vote algorithm is simple, elegant, and optimal for majority detection.

Its correctness follows directly from the cancellation principle and the invariant maintained

throughout the traversal. The Bekzat’s implementation demonstrates solid understanding of the core logic, with room for improvement in testing, verification, and documentation. Overall, it achieves the required functional and performance criteria for the assignment