# Report A: Analysis of Boyer–Moore Majority Vote

Author: Student B (peer reviewer)  
Partner: Student A (implementation author)

## 1. Algorithm Overview

The Boyer–Moore Majority Vote algorithm finds a majority element (if one exists) in a sequence using a single pass and O(1) additional space. It maintains a candidate and a counter. The Java implementation in `BoyerMooreMajorityVote.java` mirrors this approach with a verification pass to confirm majority existence.

## 2. Complexity Analysis

Time Complexity:  
- Best/Average/Worst: Θ(n) — one pass to find candidate plus one pass for verification.  
  
Space Complexity:  
- O(1) auxiliary space (constant counters and candidate).  
  
Correctness:  
- The verification pass counts occurrences of candidate and checks >= floor(n/2)+1.

## 3. Code Review & Optimization Suggestions

Findings from code:  
- Implementation is clear and well-documented.  
- Uses Integer for majorityElement in result wrapper which allows null return when no majority.  
  
Suggestions:  
- Minor: counting threshold uses (n/2)+1; can be rewritten as > n/2 for clarity.  
- Consider combining passes only when approximate majority acceptable; but verification is standard.  
- Add generics to accept other types (T) instead of int[], to increase reusability.  
- Add unit tests for edge cases (empty array, single element) — tests exist in repo but ensure coverage.

## 4. Empirical Results

Benchmark setup: replicated algorithm in Python (behavior identical to Java logic). Each size run 3 trials; reported values are averages.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | time\_ms\_avg | time\_ms\_std | comparisons\_avg | array\_accesses\_avg |
| 100 | 0.0219 | 0.0032 | 200.0 | 250.0 |
| 1000 | 0.2765 | 0.0034 | 2000.0 | 2500.7 |
| 10000 | 12.2871 | 15.1598 | 20000.0 | 25002.7 |
| 100000 | 14.3005 | 0.5956 | 200000.0 | 250026.7 |

Observations:  
- Time scales linearly with n.  
- Comparisons and array accesses roughly proportional to n (≈2n due to verification pass).

## 5. Conclusion

Boyer–Moore Majority Vote implementation is asymptotically optimal: O(n) time, O(1) space. Code quality is good; suggested improvements are minor and mostly about API generalization and extra test coverage.