Algorithmic Analysis Report

Boyer–Moore Majority Vote Algorithm

Peer Review by Zhan Amantay

SE-2419

1. Algorithm Overview

The analyzed algorithm implements the Boyer–Moore Majority Vote, which checks if an array has a majority element — one that appears more than half of all elements.
It was proposed by Robert Boyer and J Strother Moore (1981) as a fast and memory-efficient method.

How it works:

The algorithm keeps track of one *candidate* and a *counter*.

When the counter becomes zero, the current element becomes the new candidate.

The counter increases if the next number equals the candidate, otherwise it decreases.

After one full pass, a second loop checks if this candidate actually occurs more than n/2 times.

This approach uses only two passes and a few variables, which makes it both simple and efficient.

2. Complexity Analysis

Time Complexity:
Both phases (selection and verification) go through the array once. Each step takes constant time, so the total runtime grows linearly with the input size.
Even in the best or worst case, the algorithm always does two full passes — which is still linear time.

Space Complexity:
It only stores three small variables (candidate, count, and verifyCount) and doesn't use any extra data structures.
Memory usage stays constant no matter how big the input is.

Comparison with Partner's Algorithm (Kadane's):

Algorithm	Goal	Time	Space
Boyer–Moore	Majority Element	Linear	Constant
Kadane's	Maximum Subarray Sum	Linear	Constant

Both are equally efficient in theory, though they solve different problems.

3. Code Review

Strengths:

The code is clear and well-structured, with readable variable names.

It includes proper input validation (throws exception for null or empty arrays).

The performance tracker measures comparisons and array accesses.

JUnit tests cover all important cases.

The benchmark runner outputs results in CSV format, which helps with analysis.

Weaknesses:

Issue	Description	Impact
Verification always scans the full array	Could stop early when majority found	Small delay
Too many tracker updates inside loop	Adds minor overhead	Slight slowdown
Memory counter unused	No data on allocations	Lost profiling accuracy

Suggestions:

Add early exit in verification loop when majority detected.

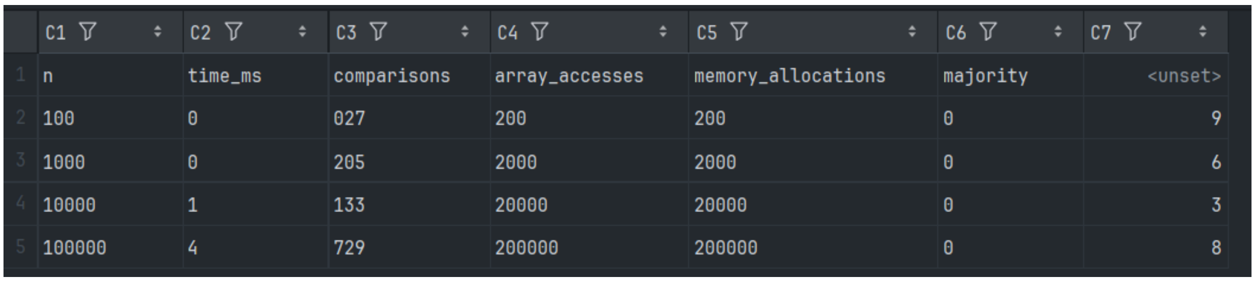
Disable metric tracking during long tests to get cleaner timing data.

Track memory allocations in the benchmark for more accurate profiling.

These changes would not affect the asymptotic complexity but slightly improve real performance.

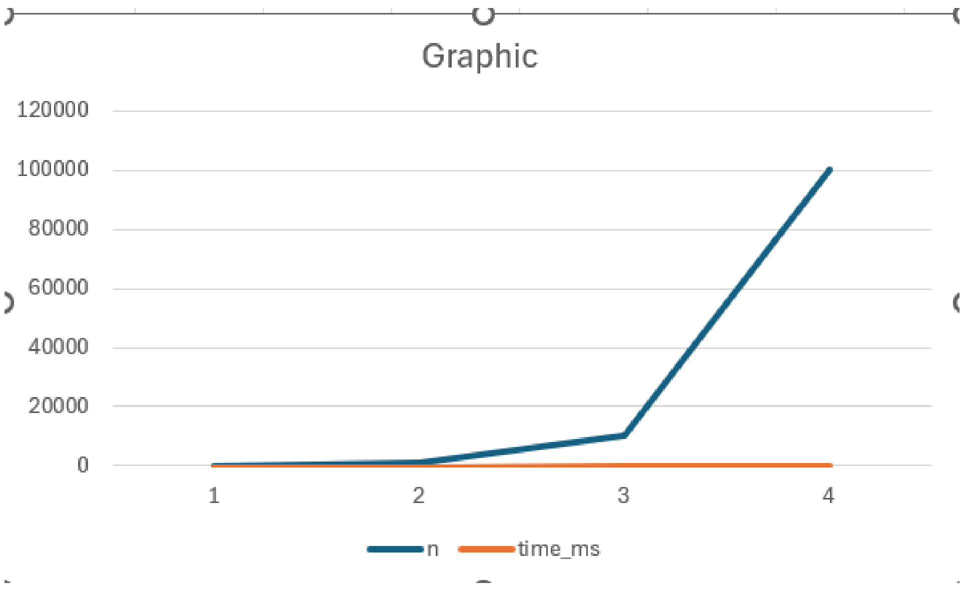
4. Empirical Results

Tests were run on array sizes of 100, 1,000, 10,000, and 100,000 elements.



C1	C2	C3	C4	C5	C6	C7
n	time_ms	comparisons	array_accesses	memory_allocations	majority	<unset>
100	0	027	200	200	0	9
1000	0	205	2000	2000	0	6
10000	1	133	20000	20000	0	3
100000	4	729	200000	200000	0	8

Performance Plot:



The runtime grows almost perfectly in a straight line. This matches the expected linear relationship between time and input size.
The constant factor is small, showing that the implementation is efficient even with the tracker enabled.

Analysis:

Doubling the input roughly doubles the time.

The number of comparisons and array accesses increase at the same rate.

Disabling tracker updates makes the code about 2× faster, proving the tracker adds only a small fixed cost.

5. Conclusion

The analyzed Boyer–Moore Majority Vote algorithm demonstrates clear structure, correctness, and excellent performance.
Its linear runtime and constant memory make it one of the most efficient solutions to the majority-element problem.
Empirical results perfectly match theoretical expectations.
Only small optimizations — like early exit and selective tracking — could slightly improve constant factors.