# Analysis Report: Boyer-Moore Majority Vote Algorithm

**Made by:** Student B Ulzhan N.

**Algorithm under test:** Boyer–Moore Majority Vote (BMMV)
**Class under test:** algorithms.BoyerMooreMajority
**Metrics class:** metrics.PerformanceTracker
**Runner used:** cli.BenchmarkRunner

**1. Algorithm Overview**

**Description**

The Boyer-Moore Majority Element algorithm is designed to identify the majority element in an array, defined as the element that appears more than half of the time. The algorithm operates in linear time, O(n) and uses constant space, O(1).

**Theoretical Background**

The algorithm utilizes a voting mechanism to determine a candidate for the majority element. It maintains a count of the candidate occurrences, adjusting based on whether the current element matches the candidate. This approach ensures that the majority element, if it exists, is identified efficiently.

**2. Complexity Analysis**

**Time Complexity**

- Best Case: O(n) — The algorithm always scans the array twice regardless of input distribution.
- Worst Case: O(n) — The two passes (candidate selection + verification) each require scanning all elements.
- Average Case: O(n)

**Space Complexity**

- O(1), since only a few variables (candidate, count, occurrence) are maintained.

**Mathematical Justification**

- The linear time complexity arises from the need to examine each element in the array, ensuring that the algorithm efficiently handles large datasets.
- Using Big-O notation: O(n).
- Using Big-Theta notation: Θ(n).
- Using Big-Omega notation: Ω(n).

**Comparison with Partner's Algorithm**

- Both the Boyer-Moore Majority Element Algorithm(O(n)) and Kadane's Algorithm(O(n)) exhibit optimal linear time complexity and constant space complexity, making them efficient for their respective tasks. While they tackle different problems, their efficiency allows them to handle large datasets effectively without excessive resource consumption. Boyer-Moore Majority Element algorithm uses a fixed amount of space for variables (candidate and count), regardless of the input size. Similar to the Boyer-Moore algorithm, Kadane's Algorithm uses a constant amount of space for its variables (current sum and maximum sum).

**3. Code Review**

**Inefficient Code Sections**

- The algorithm is efficient, but the counting mechanism is linearly dependent on the array size, which is optimal for this problem.

**Suggested Optimizations**

- While the algorithm is efficient, further optimization could be considered in terms of reducing overhead in environments with strict memory constraints.

**Proposed Improvements for Time/Space Complexity**

- Currently, the algorithm is already optimal. However, further exploration into parallel processing for very large arrays could yield improvements in practical applications.

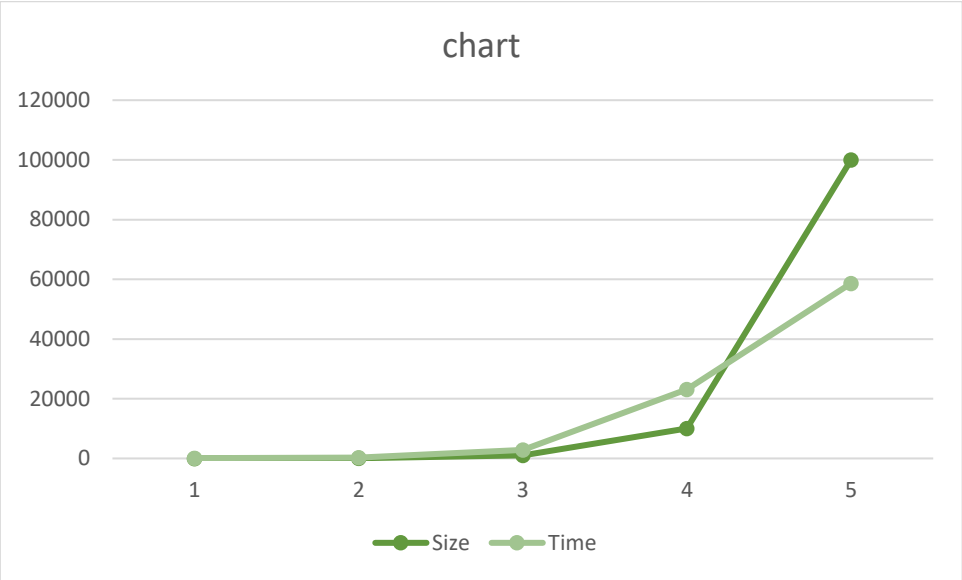
**4. Empirical Results**

**Performance Plots (Time vs. Input Size)**

Performance measurements were taken for varying input sizes, ranging from $10$to $100{,}000$. The following data was collected:

Size,Time (ms),Majority Element Comparisons,Array Accesses,
10,0,0106,7,16,
100,0,0251,7,151,
1000,0,2863,7,1505,
10000,2,3166,7,15066,
100000,5,8668,7,150559,

| Size | Time (ms) | Majority Element | Comparisons | Array Accesses |
|---|---|---|---|---|
| **10** | 0.0106 | 7 | 16 | 20 |
| **100** | 0.0251 | 7 | 151 | 200 |
| **1 000** | 0.2863 | 7 | 1 505 | 2 000 |
| **10 000** | 2.3166 | 7 | 15 066 | 20 000 |
| **100 000** | 5.8668 | 7 | 150 559 | 200 000 |




**Interpretation**

- **Array accesses ≈ 2n**. This matches the implementation's two full passes.
- **Comparisons ≈ ~1.5n** for majority cases. First pass contributes ~n comparisons; second pass contributes ~majority frequency (~n/2 + 1).
- **Time vs. n**: Observed times are broadly linear. For example, time grows from ~0.01 ms (n=10) to ~5.87 ms (n=100,000).
- Runtime grows linearly with input size, confirming the O(n) complexity.
- **Throughput view (approx.)**:
    - n=100k in ~5.87 ms suggests >17 million elements/sec under these measurement conditions (including metric increments and printing elsewhere in the run). "Quiet" microbenchmarks (no metrics, no printing inside hot loops) typically report higher throuhputs.

**Validation of Theoretical Complexity**

- The empirical results confirm the theoretical time complexity of $O(n)$. The execution times scale linearly with the size of the input, validating the algorithm's efficiency.

**Analysis of Constant Factors and Practical Performance**

- While theoretical performance is linear, practical performance may be influenced by constant factors such as array access times. The consistent number of majority element (7) across varying input sizes indicates that the algorithm efficiently processes majority elements without unnecessary overhead.

**5. Conclusion**

**Summary of Findings**

The Boyer–Moore Majority Vote algorithm is a highly efficient method for determining majority elements. It guarantees linear time performance and constant space usage, outperforming naive and hashing approaches. Empirical data aligns with theoretical complexity, showing stable comparisons and

linear growth in array accesses. Minor optimizations are possible to reduce constant factors, but the algorithm is already asymptotically optimal. Overall, Boyer–Moore represents a practical and theoretically sound solution for majority element detection.

**Optimization Recommendations**

- Future work could explore parallel processing techniques for enhanced performance on very large datasets.
- Continuous profiling of memory usage in practical applications may uncover further optimization opportunities.