Peer Algorithm Analysis Report – Boyer–Moore Majority Vote

Student Name: Prince Sharma, Kartik Jindal
Partner's Algorithm: Boyer–Moore Majority Vote Algorithm
My Algorithm: Kadane's Algorithm (Maximum Subarray Problem)
Course: Algorithmic Analysis and Peer Code Review
Language Used: Java

1. Algorithm Overview

The Boyer–Moore Majority Vote Algorithm is designed to find the element that appears more than n/2 times in an array. It works by maintaining a candidate and a counter. As it scans the array once, it adjusts the counter based on matches and mismatches.

Steps:
1. Initialize count = 0, candidate = None.
2. For each element:
  • If count == 0 → set current element as candidate.
  • If element == candidate → increment count.
  • Else → decrement count.
3. The remaining candidate is the potential majority element.

Example: Input: [2, 2, 1, 2, 3, 2, 2] → Output: 2 (appears 5 times out of 7)

2. Complexity Analysis

Case	Description	Time Complexity	Space Complexity
Best Case	All elements are same	Θ(n)	O(1)
Average Case	Random values	Θ(n)	O(1)
Worst Case	Alternating values	O(n)	O(1)

• Time Complexity: Linear time since the array is traversed once.
• Space Complexity: Constant (uses only a few variables).
• Recurrence Relation: $T(n) = T(n-1) + O(1) \rightarrow T(n) = O(n)$

3. Code Review & Optimization

Strengths:
• Clean single-pass structure.
• Very efficient with minimal space.
• Correct logic and simple implementation.

Weaknesses:
• Lacks input validation for empty or null arrays.
• No verification pass to confirm majority candidate.
• Limited comments for explanation.

Suggestions for Improvement:
1. Add a second pass to confirm the candidate truly appears > n/2 times.
2. Add input checks for edge cases.
3. Add benchmark tracking (comparisons, array accesses, etc.).

4. Empirical Validation

Algorithm	Input Size (n)	Input Type	Time (ms)
BoyerMoore	100	Random	1.4465
BoyerMoore	100	Sorted	0.0167
BoyerMoore	100	Reverse	0.0145
BoyerMoore	100	NearlySorted	0.0139
BoyerMoore	1,000	Random	0.144
BoyerMoore	1,000	Sorted	0.1182
BoyerMoore	1,000	Reverse	0.1168
BoyerMoore	1,000	NearlySorted	0.1181
BoyerMoore	10,000	Random	1.3012
BoyerMoore	10,000	Sorted	1.3196
BoyerMoore	10,000	Reverse	1.6663
BoyerMoore	10,000	NearlySorted	0.9386
BoyerMoore	100,000	Random	2.2615
BoyerMoore	100,000	Sorted	2.0466
BoyerMoore	100,000	Reverse	2.1549
BoyerMoore	100,000	NearlySorted	2.2069

Observations:
• Running time grows linearly with input size.
• Small variations are due to data distribution.
• Confirms the O(n) time complexity experimentally.

5. Comparison with My Algorithm (Kadane’s Algorithm)

Metric	Boyer–Moore	Kadane’s Algorithm
Problem Solved	Finds majority element	Finds maximum subarray sum
Time Complexity	O(n)	O(n)
Space Complexity	O(1)	O(1)
Core Operation	Counting / Voting	Summation tracking
Verification Step	Optional second pass	Not required
Best For	Identifying dominant value	Finding largest sum

Observation: Both algorithms are linear-time and space-efficient but solve very different problems. Kadane’s focuses on continuous sum optimization, while Boyer–Moore identifies frequency dominance.

6. Conclusion

The Boyer–Moore Majority Vote Algorithm is an elegant and highly efficient method to detect majority elements. Its linear-time performance and constant space make it ideal for large datasets. The theoretical and experimental results are consistent, confirming its O(n) complexity. With added validation and improved documentation, the code would be fully robust and professional.