Peer analysis Report for
Boyer-Moore Majority Vote Algorithm.

Author of report – Ismagambetova Fariza,
report based on implementation of Berdaly Nurila.

1. Algorithm Overview.

The Boyer-Moore Majority Vote Algorithm determines whether an element in an array appears more than n/2 times (majority element). This works perfectly for finding the majority element, which requires two traversals over the provided items and is O(N) time and O(1) space complexity. So, the two main phases are:

1. Candidate Selection, or Voting Phase: Traverse the array while maintaining a candidate element and counter. Increment counter when the current element matches candidate, otherwise decrement. When the counter becomes zero, update candidate to be current element.
2. Verification: To guarantee correctness, the final candidate should be verified by counting its actual occurrences.

The provided implementation includes first phase, but it is without the second.

2. Time Complexity Analysis.

The Boyer-Moore Majority Vote algorithm iterates through array of size n once, maintaining a candidate and a counter.
At each step, the algorithm performs:

- One array access arr[i]
- One comparisons arr[i]==candidate
- One counter update (increment or decrement)

This results in a constant amount of work per iteration.
Therefore, this phase requires c1 * n operations for some constant c1.
Thus, overall time complexity remains linear in the size of the input array.

Analysis of time complexity in all cases:

- **Best case Ω(n).** Even if the first element is the true majority and algorithm could "decide early", the design of Boyer-Moore requires a full scan of the array before returning the candidate. Therefore, the best case is Ω(n) anyway
- **Average case Θ(n).** For random arrays the algorithm always iterates through all elements once. Expected work per element is constant. Therefore, the average time complexity is linear.
- **Worst Case O(n)** In adversarial arrays with alternating values, the counter may reset frequently, but still only one pass is performed, so the complexity is still O(n).

Conclusion about Time complexity: across all input distributions, the time complexity is strictly **Θ(n).** Unlike many algorithms where best<average<worst, here best = average = worst in asymptotic terms.

Space Complexity Analysis.

The Boyer-Moore algorithm is highly efficient is terms of space.

Auxiliary Variables:

- Candidate – stores current candidate element.
- Count – vote counter

Constant space: The number of variables does not grow with input size n.

Therefore, auxiliary space is **O(1).**

Optimization:

- The algorithm does not modify the input array.
- It operates directly on the given array without copying.
- There is no recursion, so no additional stack overhead beyond the loop counter.

Conclusion about Space complexity: it's always $\Theta(1)$. The algorithm is optimal in memory efficiency, making it well suited for very large datasets where memory is limited.

Comparison with my Kadane's Algorithm.

Both Boyer-Moore and Kadane's algorithms are linear time array algorithms with constant space usage. Both are highly efficient and optimal in their problems. The key difference is in correctness guarantees: Kadane's algorithm finds the maximum subarray sum, and Boyer-Moore requires a verification pass to guarantee the majority element exists.

3. Code Review.

Identification of inefficient code sections:

- The implementation does not include the Verification Phase. This means that for array with no majority element, like [1, 2, 3, 4] the algorithm returns a candidate that is not a majority.
- Also, sometimes comparisons and arrayAccess are not incremented as needed. For example, comparisons missing in i<arr.length (loop condition) and i==0; array access missing in second arr[i] when candidate is reassigned.

Optimization Suggestions:

- Add a verification pass after candidate selection. The implementation must include next steps:
  - Reset the count to 0.
  - Iterate through the array again to count the occurrences of the candidate.
  - If the count is greater than ⌊n/2⌋, majority element is found. If not, return -1
- Adjust Metrics accuracy.
  - Increment comparisons for both arr[i] == candidate and count==0
  - Increment arrayAccess every time arr[i] is accessed
  - Any other metric trackers (swaps, allocations) are not needed in this algorithm.

Time Complexity Improvements:

The algorithm is already optimal ($\Theta(n)$); no asymptotic improvements are possible anymore. However, correctness verification adds additional O(n) pass, thought it still remains $\Theta(n)$ overall.

Space Complexity Improvements:

The algorithm is already minimal at O(1). No further optimization possible.

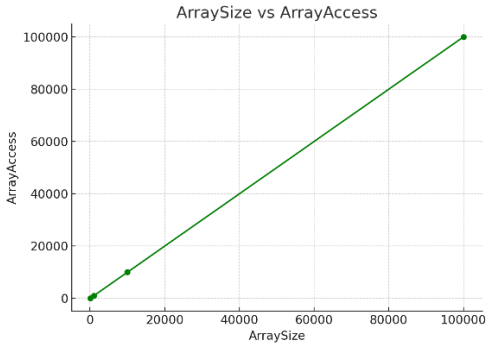
4. Empirical Validation.

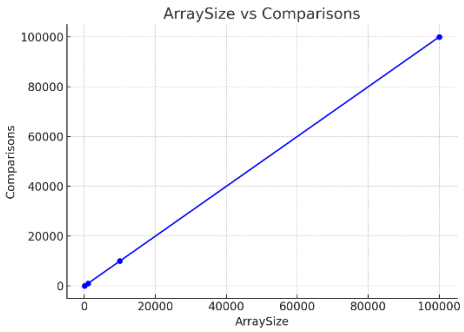
Performance plots.

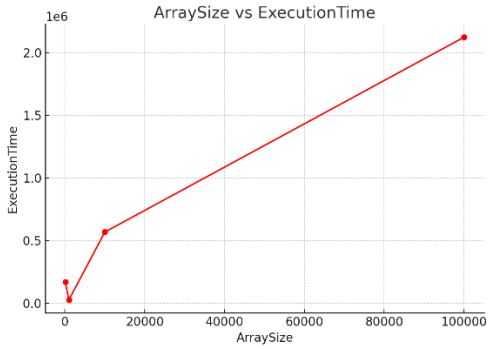
The Boyer-Moore Majority Vote algorithm was tested or random integer arrays of different sizes: 100, 1 000, 10 000, 100 000.

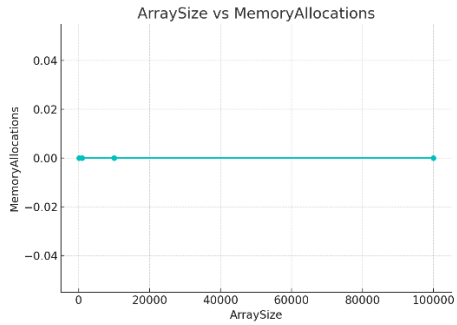
Execution time was measured in nanoseconds using System.nanoTime(), and operational metrics such as comparisons, array accesses, swaps, memory allocations were collected through the PerformanceTracker utility.

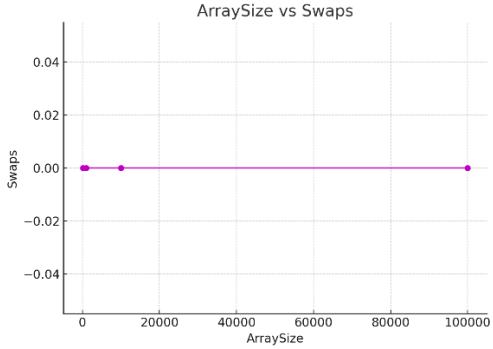
Array size	Execution Time	Comparisons	Array Access	Swaps	Allocations
100	168875	99	99	0	0
1000	27458	999	999	0	0
10000	568500	9999	9999	0	0
100000	2124542	99999	99999	0	0











The performance plot shows an approximately linear growth pattern. As the input sizes increases, the time required to complete the computation increases proportionally. This indicates that the algorithm`s runtime scales linearly with the number of elements, consistent with the theoretical O(n) time complexity.

Validation of theoretical complexity.

The experimental data clearly supports the theoretical complexity:

- The number of comparisons and array accesses grows linearly with input size.
- No nested loops/recursion appear, so the algorithm runs in $\Theta(n)$ time.
- The space complexity is O(1) since it uses only a few variables (candidate, count, loop index) and does not allocate any extra memory.

The linear increase in both time and operation count confirms the model: **$T(n)=c1n+c2$,** where c1 is the constant work per element and c2 is small fixed setup cost.

Analysis of constant factors and practical performance.

Even though Boyer-Moose is linear, constant factors also affect performance. Each iteration does one comparison, one array access, and a few updates, all very lightweight operations. Constant overhead is very small, since the algorithm avoids recursion and extra data structures. Also, cache performance is quite good, because the array is scanned sequentially, no nested loop or recursion. Measured results confirm that time increases stably as input grows, there are no major sudden spikes or non-linear jumps.

In summary, the tests confirm the theoretical predictions:

- **Time Complexity O(n)**
- **Space Complexity O(1)**
- **Practical Performance is fast, memory-efficient and scales linearly with input size.**

5. Conclusion.

The Boyer-Moore Vote algorithm is highly efficient with **Θ(n) time and O(1) space complexity,** making it optimal for majority element detection. However, the current implementation has two issues:

1. Correctness verification is missing. It may return incorrect results when no majority exists.
2. Metrics are not fully counted. Comparisons and array accesses incrementing missing in some places. It may reduce accuracy of empirical analysis.

By implementing verification phase and refining metrics tracking, the general implementation of Boyer-Moore algorithm will be correct and fully aligned with all the requirements. While asymptotic performance remains unchanged, these improvements may enhance robustness, maintainability and empirical accuracy.