Peer Analysis Report on Majority Element Algorithm

This report analyzes the Majority Element algorithm implemented by my partner. It covers asymptotic complexity, code review and optimization, and empirical validation.
The time complexity in the best case (Ω) is Ω(n), as the algorithm requires a single pass through the array to find the candidate, even in the best case. In the average case (Θ), the time complexity is Θ(n), as two passes are made over the array: one to find the candidate and another to verify it. In the worst case (O), the time complexity is O(n) since the algorithm still performs two passes even if no majority element exists. Therefore, the time complexity summary is Ω(n) for the best case, Θ(n) for the average case, and O(n) for the worst case.
Regarding space complexity, the algorithm uses constant space (O(1)), requiring only a few variables like candidate, count, and occurrences. This ensures that the space complexity is O(1) with in-place operations. Since there is no recursion in the algorithm, recurrence relations do not apply.
During the code review, an inefficiency was identified with the redundant second pass for verification. The algorithm performs two passes over the array, which could be optimized by combining both steps into one pass. By merging candidate search and verification, the time complexity can be reduced from O(2n) to O(n). The code snippet below demonstrates how both steps can be merged into a single pass:

```
public static int majorityElementOrNone(int[] nums) {
    int candidate = 0, count = 0, occurrences = 0;
    for (int i = 0; i < nums.length; i++) {
        if (count == 0) {
            candidate = nums[i];
            count = 1;
        } else if (candidate == nums[i]) {
            count++;
        } else {
            count--;
        }

        if (count > 0) {
            occurrences++;
        }
    }

    if (occurrences > nums.length / 2) {
        return candidate;
    } else {
        return Integer.MIN_VALUE;
    }
}
```

No further improvements are needed for space complexity as it is already O(1). The code is clear but would benefit from additional comments explaining key steps, especially the logic behind resetting count and choosing a candidate. The code is modular, but combining the candidate search and verification will make it simpler and more maintainable.
For empirical validation, benchmarking on input sizes ranging from n = 100 to n = 100,000 should confirm the O(n) time complexity. A time vs input size plot should show a linear relationship, confirming the O(n) complexity. Measured performance should align with theoretical predictions. The impact of optimization should be measured by combining candidate search and verification, as this should reduce execution time, especially for larger arrays.
The Majority Element algorithm is efficient with O(n) time complexity and O(1) space complexity. However, combining the candidate selection and verification into one loop will further optimize the algorithm.

Recommendations:

1. Combine candidate search and verification into one loop to reduce time complexity.
2. Add comments to improve readability.
3. Benchmark the algorithm to validate its performance.
4. Measure the impact of the optimization.