**Design and Analysis of Algorithms**

Assignment 2: Algorithmic Analysis and Peer Code Review

1. Algorithm Overview

Kadane’s Algorithm is a linear-time algorithm used to find the maximum sum subarray within a one-dimensional array of numbers. It efficiently tracks the maximum cumulative sum while iterating through the array, resetting the temporary sum whenever it becomes negative. The algorithm is particularly efficient because it only requires a single pass through the array, making it O(n) in time complexity and O(1) in space.

In comparison, Boyer–Moore Majority Vote Algorithm (implemented by Daniyal) is another linear-time algorithm that identifies the majority element in an array — the element that appears more than n/2 times. It works by maintaining a candidate and a counter, updating them as it scans the array. Both algorithms share linear complexity and in-place computation, but they target different problems.

2. Complexity Analysis

Kadane’s Algorithm achieves O(n) time complexity since it processes each element of the array exactly once. In the best, worst, and average cases, the complexity remains Θ(n) because there are no nested loops or recursive calls. The space complexity is O(1), as it only uses a fixed number of variables (currentSum, maxSum, indices).

Similarly, Boyer–Moore Majority Vote Algorithm also has O(n) time complexity and O(1) space complexity. Both algorithms exemplify highly efficient linear array algorithms with minimal overhead.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| Kadane’s Algorithm | Θ(n) | Θ(n) | Θ(n) |
| Boyer–Moore Majority Vote | Θ(n) | Θ(n) | Θ(n) |

Both algorithms are linear-time, but Kadane’s focuses on cumulative sums, while Boyer–Moore focuses on frequency balance.

3. Code Review and Optimization

Adema’s implementation of Kadane’s Algorithm demonstrates clean and well-structured Java code. The PerformanceTracker class accurately records comparisons, array accesses, and operations, which helps quantify efficiency. Input validation is present, throwing an exception for null or empty arrays, ensuring robustness.

Possible optimizations include: (1) minimizing redundant comparisons by restructuring conditions; (2) using early termination when the remaining array cannot yield a larger sum; and (3) modularizing code for better testing and reusability. However, the existing implementation already performs optimally.

Code quality is strong: proper naming conventions, clear structure, and effective separation between computation, tracking, and user interface (BenchmarkRunner).

4. Empirical Results

The benchmark results confirm Kadane’s theoretical linear complexity. When tested with input sizes of 100, 1,000, 10,000, and 100,000 elements, the execution time increases proportionally with n. This validates the expected O(n) performance. The PerformanceTracker metrics show that the number of comparisons and array accesses scale linearly with array size.

Compared to Boyer–Moore, Kadane’s algorithm exhibits similar scaling behavior. Both maintain near-constant memory usage and show stable performance across various input distributions. Any time differences are primarily due to the type of operations performed — sums in Kadane’s vs. equality checks in Boyer–Moore.

5. Conclusion

In summary, Adema’s Kadane’s Algorithm is a well-implemented, efficient, and readable solution for the maximum subarray problem. It adheres to algorithmic efficiency principles and demonstrates excellent performance consistency. Both Kadane’s and Boyer–Moore algorithms exemplify elegant linear-time solutions with minimal space usage. Future enhancements could include visualization tools or integration with CSV-based performance logging to further support empirical analysis.