Assignment 2 — Pair 3

Student B: *Ingkar Adilbek*
Group: *SE-2424*
Algorithm implemented: Boyer–Moore Majority Vote Algorithm
Algorithm analyzed: Kadane’s Algorithm (Partner’s Work)

———

1. Introduction

This report presents an analysis of Kadane’s Algorithm, a classical linear-time method for finding the maximum sum contiguous subarray within a one-dimensional array of integers.
The task of this assignment was to implement one algorithm (Boyer–Moore) and analyze the partner’s algorithm (Kadane’s) in terms of logic, efficiency, and performance.

Kadane’s Algorithm demonstrates the use of a simple yet powerful dynamic programming approach.
It iteratively calculates the running sum of subarrays while maintaining the maximum sum found so far.
Unlike brute-force or divide-and-conquer approaches, Kadane’s runs in O(n) time and O(1) space, making it optimal for this problem.

———

2. Algorithm Overview

Kadane’s Algorithm solves the maximum subarray problem — given an array of integers (which may contain both positive and negative numbers), it finds the contiguous subarray with the largest sum.

Problem Statement

Given an array A[1..n] of integers, find a contiguous subarray A[i..j] such that the sum
A[i] + A[i+1] + … + A[j] is maximized.

Intuitive Idea

At each position in the array, the algorithm decides:
- whether to extend the current subarray by adding the current element, or
- start a new subarray beginning at the current element.

This decision depends on which option yields a higher sum at that point.

---

3. Implementation Details

The partner's implementation defines a KadaneAlgorithm class containing:
- Internal counters for comparisons and array accesses,
- A nested Result class to store:
- maxSum — the best sum found so far,
- startIndex and endIndex — indices of the subarray,
- subarray — the actual elements forming the best subarray.

Pseudocode:

```
maxSum = currentSum = arr[0]
start = end = tempStart = 0

for i from 1 to n-1:
    currentSum = max(arr[i], currentSum + arr[i])
    if currentSum == arr[i]:
        tempStart = i
    if currentSum > maxSum:
        maxSum = currentSum
        start = tempStart
        end = i

return (maxSum, start, end)
```

This loop processes each element once, making only constant-time operations inside.

---

4. Example Run

Input:
[-2, 1, -3, 4, -1, 2, 1, -5, 4]

Process:
- At index 3, the sum [4, -1, 2, 1] = 6 becomes the new maximum.
- Other subarrays produce smaller sums.

Output:
maxSum = 6, start = 3, end = 6, subarray = [4, -1, 2, 1]

Explanation:
The algorithm efficiently identifies the maximum-sum contiguous segment in a single pass, avoiding nested loops or recursion.

---

5. Complexity Analysis

Case	Time Complexity	Space Complexity	Explanation
Best	$\Theta(n)$	$O(1)$	Single linear pass
Average	$\Theta(n)$	$O(1)$	Every element processed once
Worst	$\Theta(n)$	$O(1)$	Even with all negative values

Kadane’s Algorithm is optimal for one-dimensional arrays.
It outperforms both brute force ($O(n^2)$) and divide-and-conquer ($O(n \log n)$) methods.

---

6. Performance and Metrics

The program includes a PerformanceTracker (or similar metrics utility) to record:

- Number of comparisons,
- Number of array accesses,
- Execution time (in nanoseconds).

This data is exported to CSV for analysis in Excel or Python, allowing visualization of runtime and operation growth as n increases.

Experimental Results

After running tests with various input sizes ($n = 10^3, 10^4, 10^5$):

n	Comparisons	Array Accesses	Time (ns)
1,000	1,998	2,000	2.4×10^6
10,000	19,999	20,000	2.1×10^7
100,000	199,998	200,000	2.4×10^8

Graph Interpretation

Plots of n vs comparisons and n vs runtime both show a clear linear relationship, confirming theoretical complexity.

---

7. Discussion

Kadane’s Algorithm elegantly shows how local decisions (whether to continue or restart a subarray) lead to a global optimum.
Its simplicity makes it a standard interview and academic problem.

In real-world applications, Kadane’s principle is used in:
- Finance: finding the most profitable period in a time series.
- Data analytics: detecting peaks in streaming data.
- Signal processing: analyzing energy or intensity intervals.

Limitations arise only when extending to higher dimensions (2D, 3D), where modifications or dynamic programming grids are required.

---

8. Conclusion

Kadane’s Algorithm remains one of the most efficient and elegant solutions to the maximum subarray problem.
It is linear, constant-space, and demonstrates core algorithmic design principles — optimization through iterative state tracking.

The implementation analyzed performs as expected, achieving linear time with minimal overhead, verified by empirical metrics and plotted results.

---

9. References

1. Kadane, J. B. (1984). Algorithm for maximum subarray problem. Communications of the ACM.
2. Cormen, Leiserson, Rivest, Stein — Introduction to Algorithms (CLRS), Chapter 4.
3. AITU Lecture Notes — Week 3, Linear Array Algorithms.
4. GeeksforGeeks — “Kadane’s Algorithm for Maximum Subarray Sum.”