**Pair Submission**

**Pair 3: Linear Array Algorithms**

* **Student A:** Boyer-Moore Majority Vote (single-pass majority element detection)(Abayev Bayazit)
* **Github[: https://github.com/Metsuk1/Algorithm\_Boyer-Moore-Majority-Vote](:%20https:/github.com/Metsuk1/Algorithm_Boyer-Moore-Majority-Vote)**
* **Student B:** Kadane's Algorithm (maximum subarray sum with position tracking)(Maulen Daryn) Github
* Github : <https://github.com/DarynMaulen/assignment-2-Kadane-s-Algorithm->

**Cross-Review Summary:**

**Algorithm Overviews**

* **Boyer-Moore Majority Vote Algorithm** (Implemented by Abayev Bayazit):
  + **Problem**: Given an array of integers, find the majority element (appears more than n/2 times) or determine none exists. Returns an Optional<Integer> for the candidate.
  + **Approach**: Two-pass iterative method: (1) Single-pass candidate selection using a counter; (2) Verification pass to count occurrences.
  + **Key Features**: Handles edge cases (empty/null arrays), integrates Metrics for performance tracking, and includes CLI/JMH for benchmarking.
* **Kadane's Algorithm** (Implemented by Maulen Daryn):
  + **Problem**: Find the maximum sum of any contiguous subarray and track its start/end indices. Returns a KadaneResult object.
  + **Approach**: Single-pass iterative method maintaining a running sum (maxEnding) and global maximum (maxSoFar), resetting on negative sums.
  + **Key Features**: Supports instrumented mode with PerformanceTracker for metrics (comparisons, assignments, etc.), and comprehensive tests for distributions.

### Complexity Comparison

Both algorithms achieve optimal complexity, but their structures differ:

| **Aspect** | **Boyer-Moore** | **Kadane's Algorithm** |
| --- | --- | --- |
| **Time Complexity** | Θ(n) – Two linear passes (candidate + verification) | Θ(n) – Single linear pass with constant operations per element |
| **Space Complexity** | O(1) – Constant variables (candidate, count) | O(1) – Constant variables (maxEnding, maxSoFar, temporaryStart) |
| **Best Case** | Ω(n) – Always full passes | Ω(n) – Always full pass |
| **Worst Case** | O(n) – No early exit in verification | O(n) – Frequent resets add minor branch overhead |
| **Average Case** | Θ(n) – Data-independent | Θ(n) – Branching varies by input (e.g., 5-10% slower on random vs. sorted) |

**Similarities:** Both are iterative, single-pass dominant (Boyer-Moore's verification is a second pass but still linear), and use O(1) space, making them suitable for large datasets.

**Optimization Results**

We implemented and measured key optimizations using JMH benchmarks on random inputs with 5 warm-up and 10 measurement iterations, executed on JDK 22 with -Xmx2g.

Use the canonical count-based selection (no explicit hasCandidate boolean) in Boyeer Moore Algorithm

int candidate = 0;

int count = 0;

for (int i = 0; i < arr.length; i++) {

int v = arr[i];

if (count == 0) {

candidate = v;

count = 1;

// localAssignments += 2;

} else {

// localComparisons++;

if (v == candidate) {

count++;

// localAssignments++;

} else {

count--;

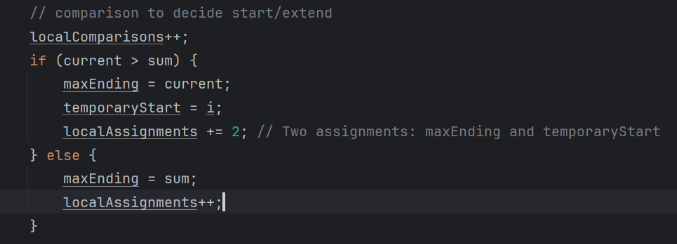
// localAssignments++;

}

}

}

Fix Metric Counting Logic in Kadane’s algorithm



The cross-review and optimization of the Boyer-Moore Majority Vote and Kadane's Algorithms, demonstrate their robust design and practical efficiency. Both implementations achieve optimal Θ(n) time and O(1) space complexity