Detailed Comparison of the Two Algorithm

1. Algorithm Overviews

- **Boyer-Moore Majority Vote (Implemented by Bekesh Dastan, Reviewed by Nurtay Aibek)**: This algorithm identifies the majority element in an array (appearing more than n/2 times, where n is the array length) in O(n) time and O(1) space. It's implemented in Java with clean code, performance metrics (comparisons and array accesses), and a CLI for benchmarking. The process has two phases:
    1. **Candidate Selection**: Iterate through the array, maintaining a candidate and count. Set a new candidate when count is zero; increment if matching, decrement otherwise. This leverages the majority element's dominance.
    2. **Verification**: Count the candidate's occurrences in a second pass; throw IllegalArgumentException if not > n/2. A Result class encapsulates the element and metrics. The BenchmarkRunner generates test arrays (majority in first n/2+1 positions) and logs to CSV. The report compares it to Kadane's for shared efficiency traits.

- **Kadane's Algorithm (Implemented by Nurtay Aibek, Reviewed by Bekesh Dastan)**: This finds the maximum sum contiguous subarray in O(n) time, outperforming O(n^2) naive approaches. It's a dynamic programming method using local-to-global optimization: track curSum (current subarray sum) and bestSum (maximum found).
    1. **Initialization**: Start with the first element as both curSum and bestSum.
    2. **Iteration**: For each element, choose to extend the current subarray or start new (max(a[i], curSum + a[i])). Update bestSum and indices if improved.
    3. **Tracking**: Maintain start/end indices for current and best subarrays. It discards negative curSum as it can't contribute positively. Implemented in Java with a Result class (maxSum, start, end) and tracking for comparisons/array accesses. The report emphasizes its efficiency for large datasets like financial analysis.

Similarities: Both use single-pass (or two-pass) iteration, O(n) time, O(1) space, and a Result class for outputs/metrics.

Differences: Boyer-Moore requires verification for correctness; Kadane's is single-pass with index tracking.

2. Complexity Analyses

Both confirm O(n) time and O(1) space, with no recursion (no recurrence relations).

Aspect	Boyer-Moore	Kadane's
Time Complexity	$\Theta(n)$ overall (two passes: ~n each for candidate selection and verification). Best/Worst/Average: $\Theta(n)$, as passes are fixed regardless of input (e.g., random, sorted). Total: $\Theta(n) + O(1)$.	$\Theta(n)$ (single pass with constant work per element). Best (all positive): $\Omega(n)$; Worst (frequent updates): $O(n)$; Average: $\Theta(n)$. No nested loops or division. $T(n) = \Theta(n) = O(n) = \Omega(n)$.
Space Complexity	O(1) (variables: count, candidate, comparisons, arrayAccesses; Result object with 3 ints). No scaling structures.	O(1) (variables: curSum, bestSum, indices (curL, bestL, bestR); temporary extend/ai; Result with maxSum/start/end; Counter object). In-place on input array.
Recurrence	None; iterative linear structure.	None; sequential processing, unlike MergeSort.

Similarities: Input-agnostic performance; constant space for all cases.

Differences: Boyer-Moore's two passes double operations (~2n comparisons/accesses); Kadane's single pass is ~n.

3. Code Reviews

Both implementations are well-crafted but have inefficiencies; suggestions focus on readability, efficiency, and modularity without changing asymptotics.

- **Boyer-Moore Inefficiencies**: Redundant full verification pass (could optimize for dominant majors); no early checks for empty/single-element arrays; metrics overhead in loops; biased benchmark inputs (majority front-loaded). Optimizations: Early termination in verification (stop at >n/2, saving ~n/2 iterations in best cases); add edge-case checks; combine metrics counters; diversify inputs (random/sorted/reverse). Quality: Good Java style, readable two-pass structure, modular (separate classes). Add comments for logic; refactor BenchmarkRunner for input/benchmark/output separation.

- **Kadane's Inefficiencies**: Poor readability (multi-operations per line in if/else); memory-heavy argument parsing (switch/streams create temporaries); unnecessary extend variable adds instructions. Optimizations: Separate operations for readability; replace switch/streams with loops/if-else for space; inline comparisons without extend for minor time gains. Quality: Strong design (Result class); straightforward. Add formatting/indentation; comments for conditions. Refactoring improves maintainability/debugging.

Similarities: Metrics tracking adds minor overhead; emphasize modularity and comments. Differences: Boyer-Moore focuses on runtime reductions (passes); Kadane's on style and memory (parsing/temporaries).

4. Empirical Results

Benchmarks on n = 10 to 100,000 show linear scaling, matching theory. Times in nanoseconds via System.nanoTime(); comparisons/accesses ~2n for Boyer-Moore, ~n for Kadane's. Tests assume majority/max sum exists.

- **Boyer-Moore Performance Table** (Majority = 1; front-loaded arrays):

n	Time (ns)	Comparisons	Array Accesses
10	6,700	20	20
100	16,100	180	200
1,000	133,800	1,740	2,000
10,000	833,900	17,559	20,000
100,000	9,721,700	174,958	200,000

Verification: Log-log plot shows linear time vs. n; array accesses area chart confirms ~2n. Optimization (early termination) reduced time by ~40% in favorable cases (e.g., 1.72M ns to 1.03M ns for n=100k). Constant ~17-18 ns/element; space constant via profiling.

- **Kadane's Performance Table** (Random/positive/negative mixes):

n	Time (ns)	Comparisons	Array Accesses
10	4,400	18	10
100	14,400	198	100
1,000	170,400	1,998	1,000

n	Time (ns)	Comparisons	Array Accesses
10,000	1,085,900	19,998	10,000
100,000	7,845,500	199,998	100,000

Verification: Log-log plot shows linear growth; ratios approach 10x as n grows (e.g., 14,400/4,400 ≈ 3.27; 7,845,500/1,085,900 ≈ 7), confirming O(n) for large n. Small n overhead masks linearity.

Similarities: Linear scaling; ~n operations per pass; validates Θ(n) with minor constants. Differences: Boyer-Moore has higher constants (~2n); Kadane's single pass is faster. Both note input biases and suggest diverse tests.

5. Conclusions

Boyer-Moore: Robust, efficient for constrained memory. Optimizations yield gains; empiricals match theory. Future: Comments, modular benchmarks, diverse inputs. Compares well to Kadane's in linear/constant traits.

Kadane's: Optimal over naive; ideal for real-world (e.g., signal processing). Readability/space improvements enhance maintainability. Empiricals confirm O(n);