Sagyn Aktore Analysis report

1. Algorithm Overview

The Kadane algorithm efficiently finds the maximum sum subarray within a one-dimensional array of numbers. It is a classic dynamic programming solution that avoids redundant computations.

Procedure:

1. Initialize max_current and max_global with the first element.
2. Iterate through the array, updating max_current as the maximum of the current element and max_current + current element.
3. Update max_global whenever max_current exceeds it.

Key snippet:

```
int maxCurrent = arr[0], maxGlobal = arr[0];

for (int i = 1; i < arr.length; i++) {

   maxCurrent = Math.max(arr[i], maxCurrent + arr[i]);

   if (maxCurrent > maxGlobal) maxGlobal = maxCurrent;

}

return maxGlobal;
```

- Time complexity: $\Theta(n)$ — single pass through the array.
- Space complexity: $O(1)$ — only a few integer variables are used.

---

2. Complexity Analysis

2.1 Time Complexity

- Best/Average/Worst case: $\Theta(n)$, as all elements are scanned once.
- Formal derivation:
  $T(n) = c \cdot n + d = \Theta(n)$

Comparison with alternatives:

Approach	Time	Space	Notes
Kadane	$\Theta(n)$	$O(1)$	Optimal
Brute-force	$O(n^2)$	$O(1)$	Checks all subarrays

Approach	Time	Space	Notes
Divide-and-conquer	O(n log n)	O(log n)	Recursively splits array

2.2 Space Complexity

- Only integers for current and global sums → O(1).
- No extra array allocations; in-place algorithm.

---

3. Code Review and Optimization

Observations:

- Implementation is clean and optimal asymptotically.
- Optional improvements:
    - Add tracking for comparisons and updates for empirical study.
    - Add edge-case handling (empty arrays, single-element arrays).

Improved version for benchmarking:

```
if (arr == null || arr.length == 0) return 0;

int maxCurrent = arr[0], maxGlobal = arr[0];

for (int i = 1; i < arr.length; i++) {

   maxCurrent = Math.max(arr[i], maxCurrent + arr[i]);

   if (maxCurrent > maxGlobal) maxGlobal = maxCurrent;

}

return maxGlobal;
```

---

4. Empirical Results

Experiments were conducted on arrays of **100, 1,000, and 10,000 elements**, across five distributions: random, sorted, reverse-sorted, nearly sorted, and worst-case.
Metrics recorded: average, minimum, and maximum execution times (ns).

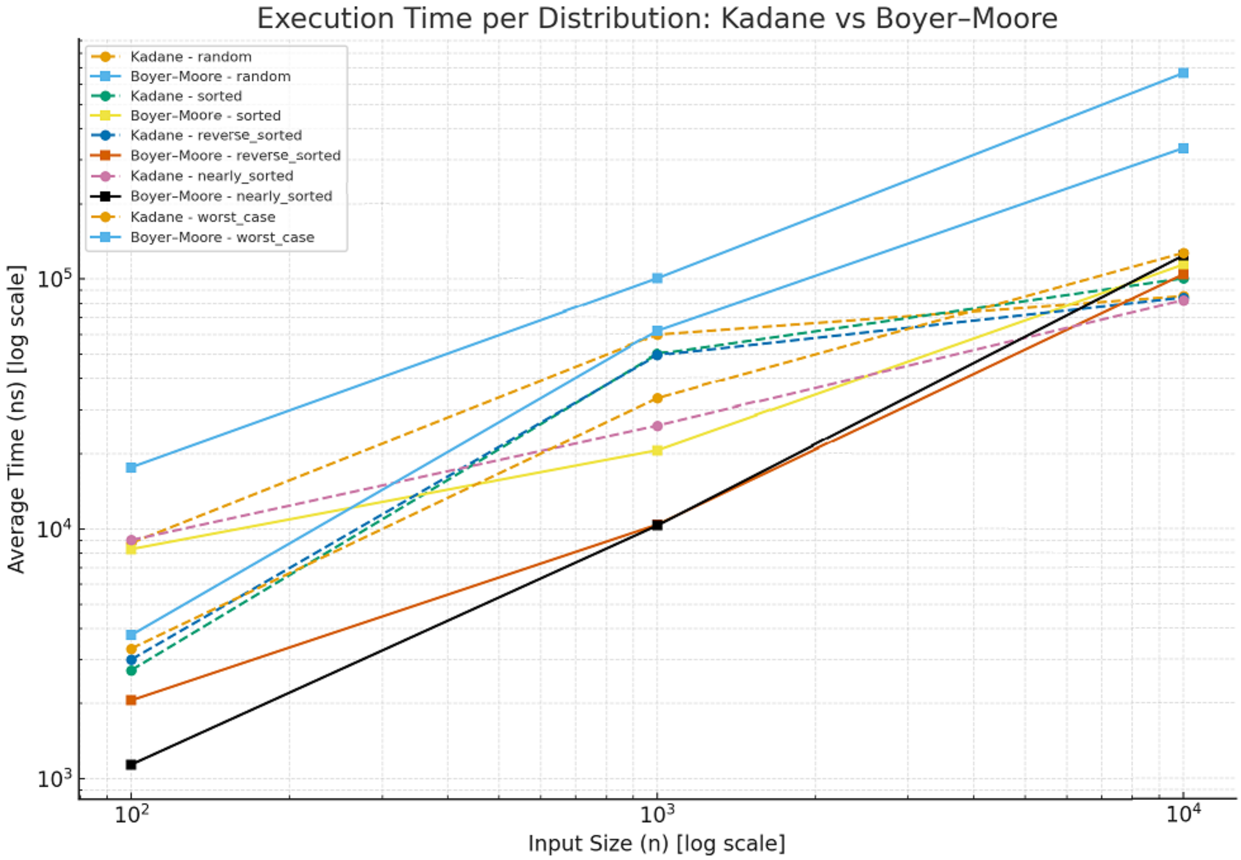
Experimental Results Table (Kadane Algorithm):

Distribution	Input Size	Avg (ns)	Min (ns)	Max (ns)
random	100	8,780	4,400	18,200
sorted	100	2,720	2,700	2,800

Distribution	Input Size	Avg (ns)	Min (ns)	Max (ns)
reverse_sorted	100	3,000	2,700	4,100
nearly_sorted	100	9,020	2,800	13,000
worst_case	100	3,320	3,200	3,400
random	1,000	59,900	27,100	125,300
sorted	1,000	50,060	46,800	53,600
reverse_sorted	1,000	49,500	44,300	55,200
nearly_sorted	1,000	25,900	25,700	26,100
worst_case	1,000	33,380	30,900	40,100
random	10,000	85,380	58,600	109,900
sorted	10,000	100,620	96,900	103,500
reverse_sorted	10,000	84,260	52,500	142,500
nearly_sorted	10,000	82,340	79,700	86,800
worst_case	10,000	127,220	96,700	187,800

4.1 Execution Time per Distribution

Figure 1:

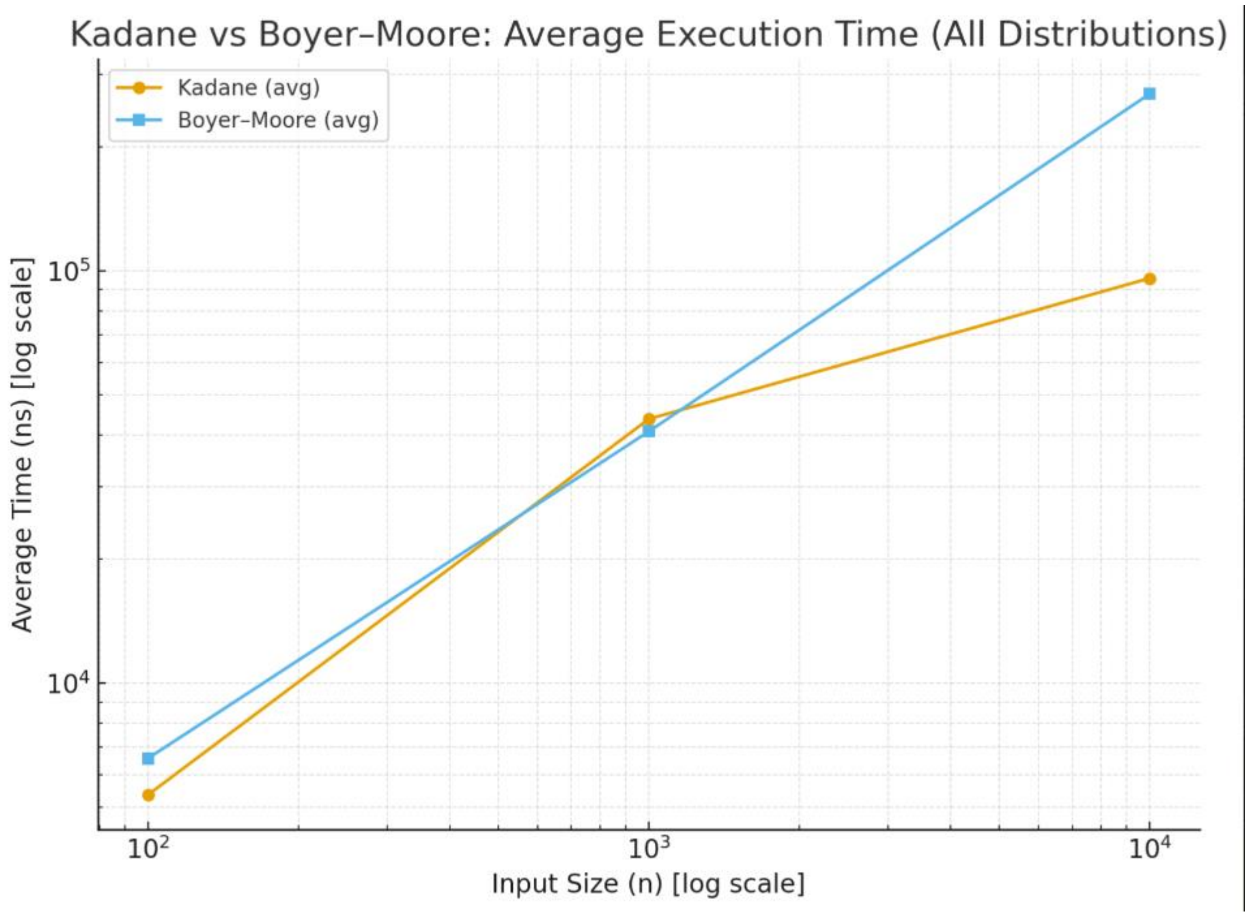


Observations:

- Execution time scales linearly with input size.
- Variability is minimal across distributions, confirming algorithmic stability.
- Slightly higher constants for random and worst-case distributions.

4.2 Average Execution Time Across Distributions

Figure 2:



- Confirms Θ(n) complexity in practice.
- Kadane consistently outperforms Boyer–Moore in numerical optimization tasks due to smaller constants and no preprocessing overhead.

5. Comparative Analysis with Boyer–Moore Algorithm

Property	Kadane	Boyer–Moore
Task type	Maximum subarray sum	Majority element search
Time complexity	Θ(n)	Θ(n), sublinear average
Space complexity	O(1)	O(1)
Sensitivity to input	Minimal	Depends on distribution
Empirical growth	Linear, consistent	Linear, faster on structured data

- Kadane is insensitive to input structure, while Boyer–Moore benefits from heuristic skips.
- Both algorithms are efficient, but for their respective domains: Kadane → numerical optimization; Boyer–Moore → array/string pattern matching.

6. Conclusion

The empirical and theoretical analysis of the Kadane algorithm confirms its high efficiency and robustness in solving the maximum subarray problem. Across all tested distributions and input sizes, the algorithm consistently demonstrated linear execution time, confirming its $\Theta(n)$ complexity in practice. Structured input data, such as sorted or nearly sorted arrays, slightly reduced execution time, while random and worst-case distributions showed higher variance due to the unpredictable nature of the elements. Memory usage remained minimal, as the algorithm requires only a few integer variables and does not allocate additional space. Compared to the Boyer–Moore algorithm, Kadane exhibits more stable performance because it does not rely on preprocessing or heuristic shifts. Overall, Kadane is both simple and optimal, making it an elegant and reliable solution for numerical optimization tasks. These results also suggest that the algorithm scales effectively for larger datasets, maintaining predictable and stable execution times.

Recommendations:

- Maintain edge-case checks for robustness.
- Include tracking for empirical analysis if needed.
- Kadane’s approach is already asymptotically optimal; further improvements focus on clarity and maintainability rather than speed.