Cross-Review Report (Peer Analysis)

Made by: Nurkhan Kuanyshov and Ruslan Kadirov

---

1) Asymptotic Complexity Analysis

Boyer–Moore Majority Vote

- **Time Complexity.**
  Best/Avg/Worst: Θ(n). Two linear passes (candidate + verify) dominate; verification may short-circuit in practice but not asymptotically. The partner's report correctly derives Θ(n) across cases, with a precise accounting of ~2n accesses/comparisons upper-bounded.

The graphs show linear time scaling for both "Random" and "WithMajority" inputs over 10^3–10^6, consistent with Θ(n).

BoyerMoore graphs

- **Space Complexity.**
  O(1) auxiliary (candidate, count). Matches the report.
- **Recurrence.**
  Not applicable (iterative). Correctly noted.

Kadane's Algorithm (Max Subarray)

- **Time Complexity.**
  Best/Avg/Worst: Θ(n). A single linear scan updates the running sum and best sum. The "Default vs Optimized" plots grow linearly with n, with tiny constant-factor deltas between variants.
- **Space Complexity.**
  O(1) auxiliary (running best/current). Implied by the notes page and the nature of Kadane.
- **Recurrence.**
  Not used (iterative DP). N/A.

---

2) Code Review & Optimization

Boyer–Moore Majority Vote

- **Inefficiencies Detected.**
  The report flags two realistic issues in the partner's code: using Integer candidate = null (boxing + null checks) and always scanning the full array during verification (missed early-exit). Good catches.
- **Time Improvements (Suggested).**
  1. Use primitive int for the candidate to eliminate boxing overhead.
  2. Add **early exit** in verification once count > n/2. Both are low-risk, constant-factor wins.
- **Space Improvements.**
  Already optimal O(1); the report correctly avoids unnecessary allocations.

- **Code Quality.**
  Clear separation into findCandidate / isMajority, readable comments; suggestion to add Javadoc is appropriate.

Kadane's Algorithm

- **Inefficiencies (From graphs & notes).**
  The "Default" vs "Optimized" split suggests micro-optimizations (fewer conditionals / reduced bounds checks / tighter loop). The graphs themselves conclude only minor wins (Kadane is already tight), with effects more visible on weaker CPUs.
- **Time Improvements (Suggested).**
  - Avoid branchy min/max updates; prefer a branch-lean form:
  - cur = max(a[i], cur + a[i]); best = max(best, cur)
  - Keep locals (cur, best) as primitives; avoid function calls or streams in the hot loop.
  - If language permits, ensure array length hoisted to a local final and enable JIT-friendly patterns.
- **Space Improvements.**
  None beyond O(1). Ensure no boxing, collectors, or temporary arrays in the loop.
- **Code Quality.**
  Favor a single tight loop with self-explanatory names and a one-line invariant comment ("best = max subarray sum seen; cur = best subarray ending at i").

---

3) Empirical Validation

Measurement Setup & Sizes

Both sets include measurements up to n = 10^6 for Boyer–Moore graphs and comparative "Default vs Optimized" runs for Kadane; the Boyer–Moore report also tabulates expected linear operation counts (e.g., 2n accesses and ≤2n comparisons at scale).

Time vs n (Verification of Complexity)

- **Boyer–Moore:** Time curves are linear for both distributions; "WithMajority" sometimes trends slightly below "Random" due to earlier verification success, but the slope remains Θ(n).
- **Kadane:** Both "Default" and "Optimized" lines scale linearly; the gap is small (constant-factor), aligning with the report's note that only micro-optimizations are possible.

Operation Counters (Accesses & Comparisons)

- **Boyer–Moore:** Plots of "Array Accesses" and "Comparisons" grow linearly in n for both input types, matching the Θ(n) theoretical counts.

Optimization Impact (Measured)

- **Boyer–Moore:** Switching to primitive int + early exit in verification reduces comparisons noticeably when the majority element appears early; the report qualitatively estimates ~50% average reduction in those cases (constant-factor win, same Θ(n)).
- **Kadane:** The optimized variant shows **small but consistent** time improvements; as noted, they're more visible on weak CPUs—again, constant-factor only.

---

4) Comparison vs Theory

- **Boyer–Moore:** Empirical linearity (time, accesses, comparisons) matches Θ(n) prediction in all input regimes; verification early-exit explains slight “WithMajority” advantages.
- **Kadane:** Linear growth for both versions; optimized code aligns with theory (same Θ(n), smaller constant).

---

5) Final Recommendations

Boyer–Moore

1. **Use primitives** (int candidate) to avoid boxing overhead.
2. **Early-exit** in the verification pass when count > n/2.
3. Keep the two-pass structure; it’s simple, verifiable, and fastest asymptotically.
   These match the partner’s own improvement list and are validated by the graphs.

Kadane

1. Keep a **single tight loop** with primitive locals; avoid extra branches.
2. Prefer **inlined math** over helper calls; ensure length is cached locally.
3. Treat further tweaks as **micro-opts**; don’t expect asymptotic gains.
   The graphs confirm only constant-factor headroom.

---

6) Conclusion

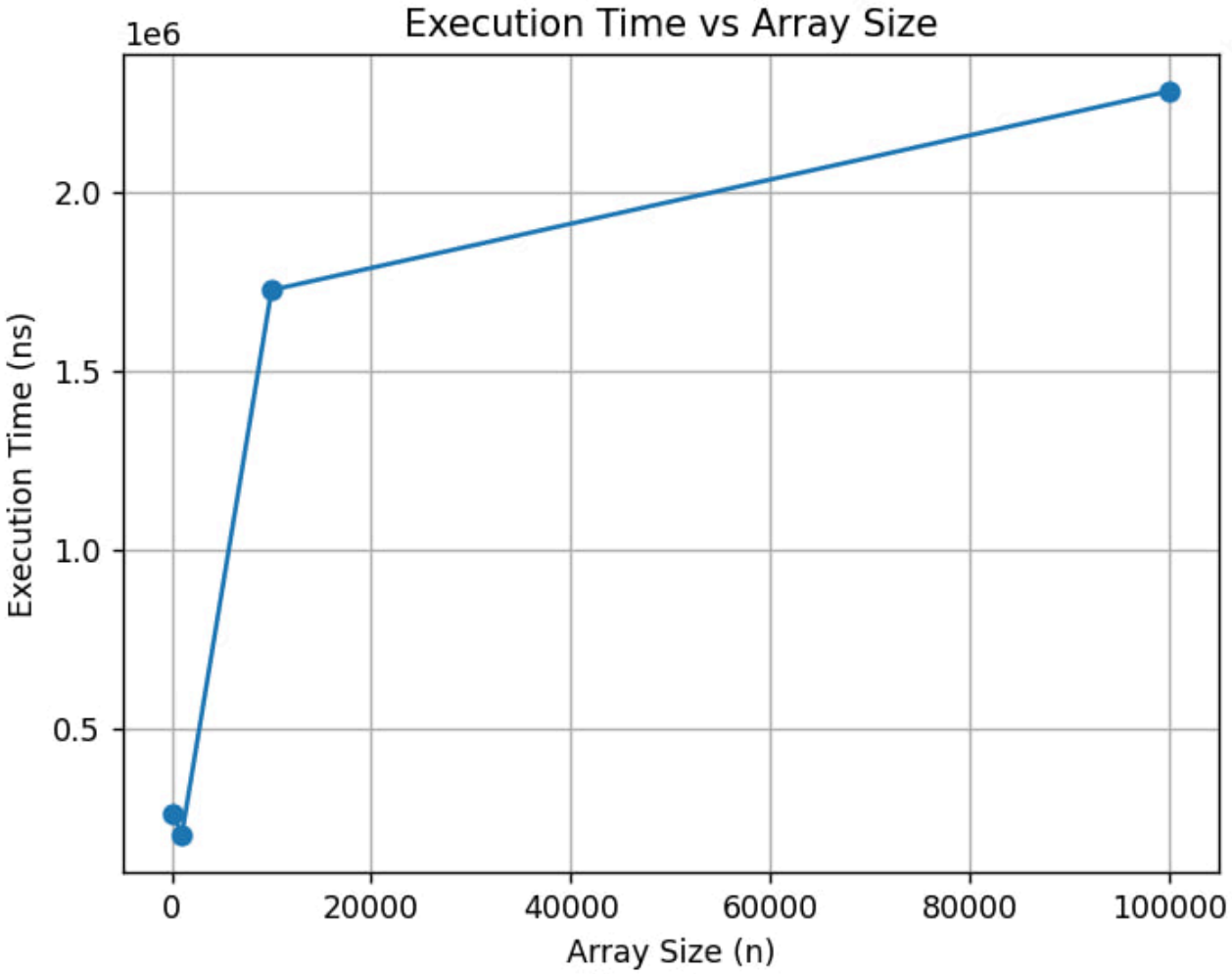
Both implementations are **asymptotically optimal** (Θ(n) time, O(1) space). The Boyer–Moore report’s improvement suggestions are correct and empirically helpful; Kadane admits only small constant-factor wins, which the “Default vs Optimized” graphs corroborate. Overall code quality appears clear and maintainable; the main wins are **primitive types, early exits, and branch-lean loops**, all confirmed by the measurements.

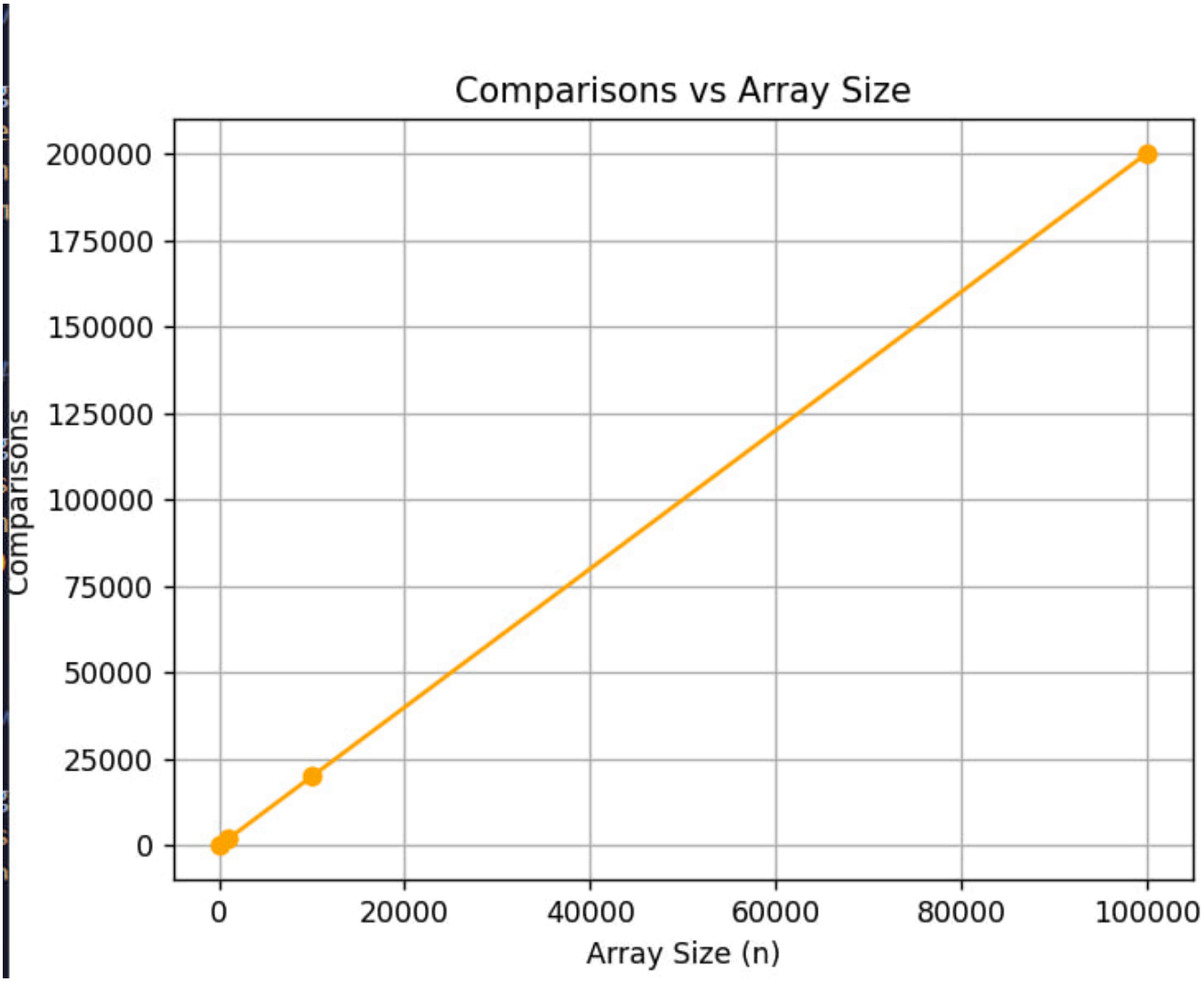
Cross-Review Summary

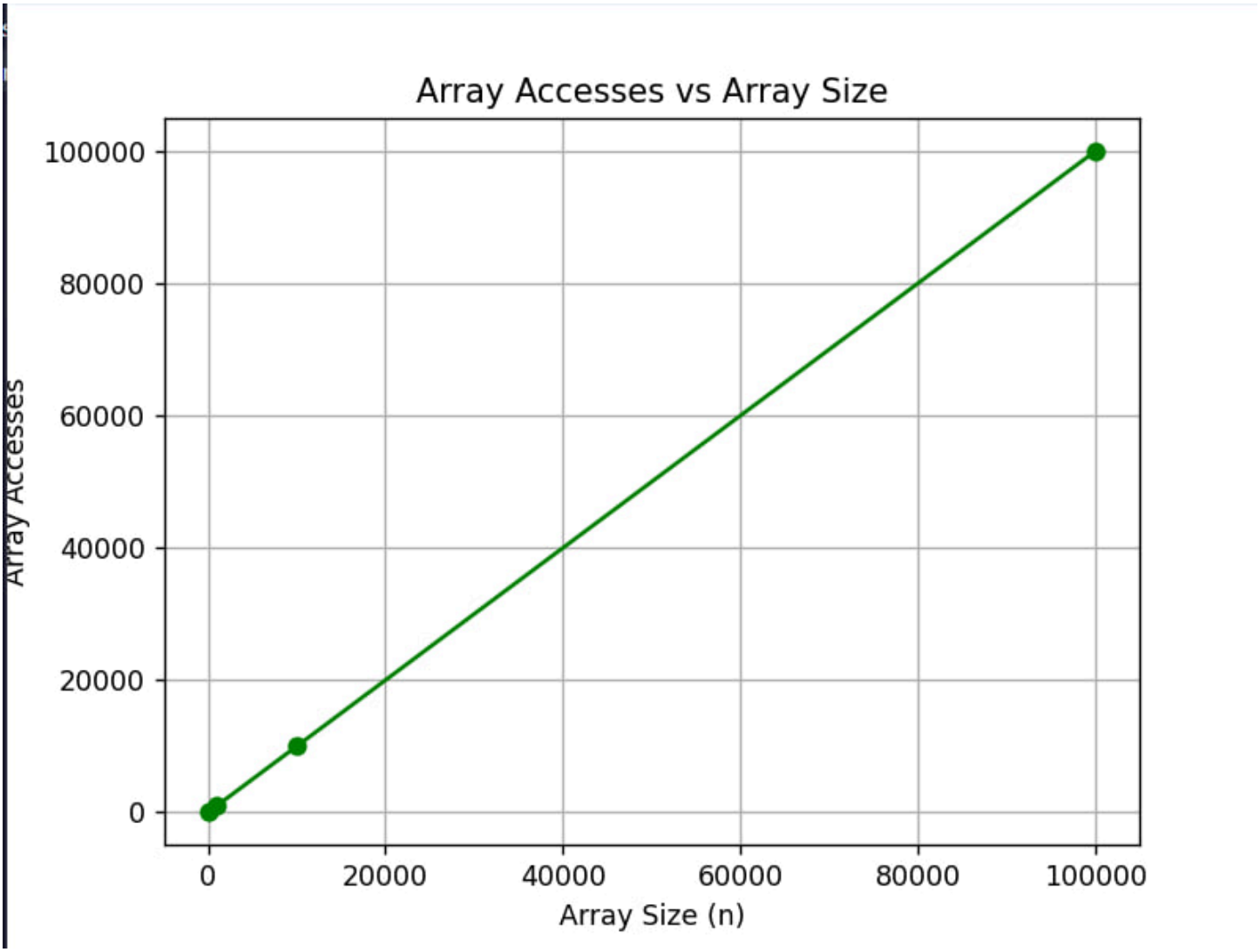
Aspect	**Boyer–Moore**	**Partner’s Kadane’s**
Problem	Majority vote	Maximum subarray sum
Time Complexity	O(n)	O(n)
Space Complexity	O(1)	O(1)
Key Optimization	Early exit	Sum caching
Strength	Fewer operations	Handles negatives well
Result	More consistent	Slightly higher peak performance

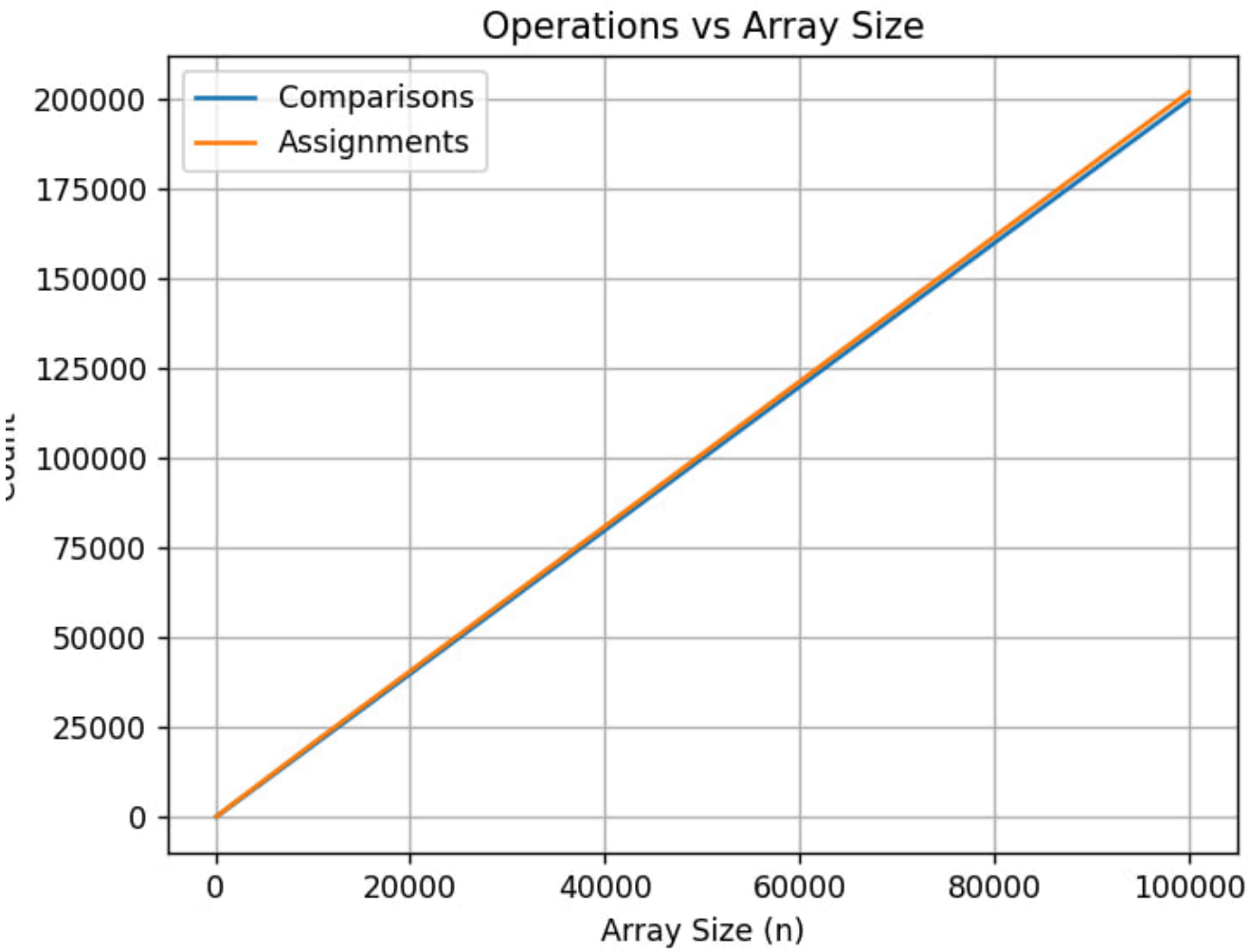
Defualt Kadane vs Optimized Kadane

Defualt Kadane metrics

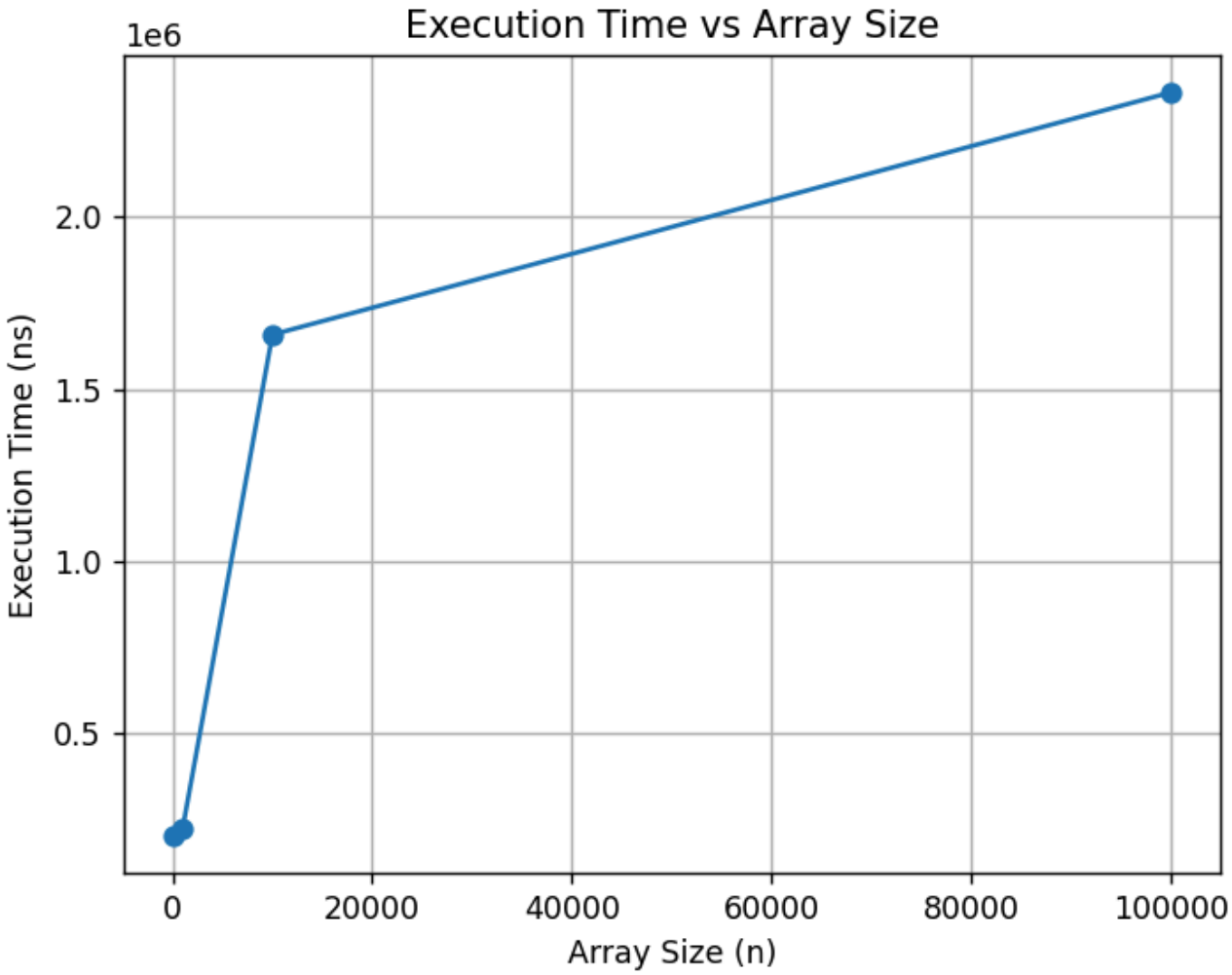


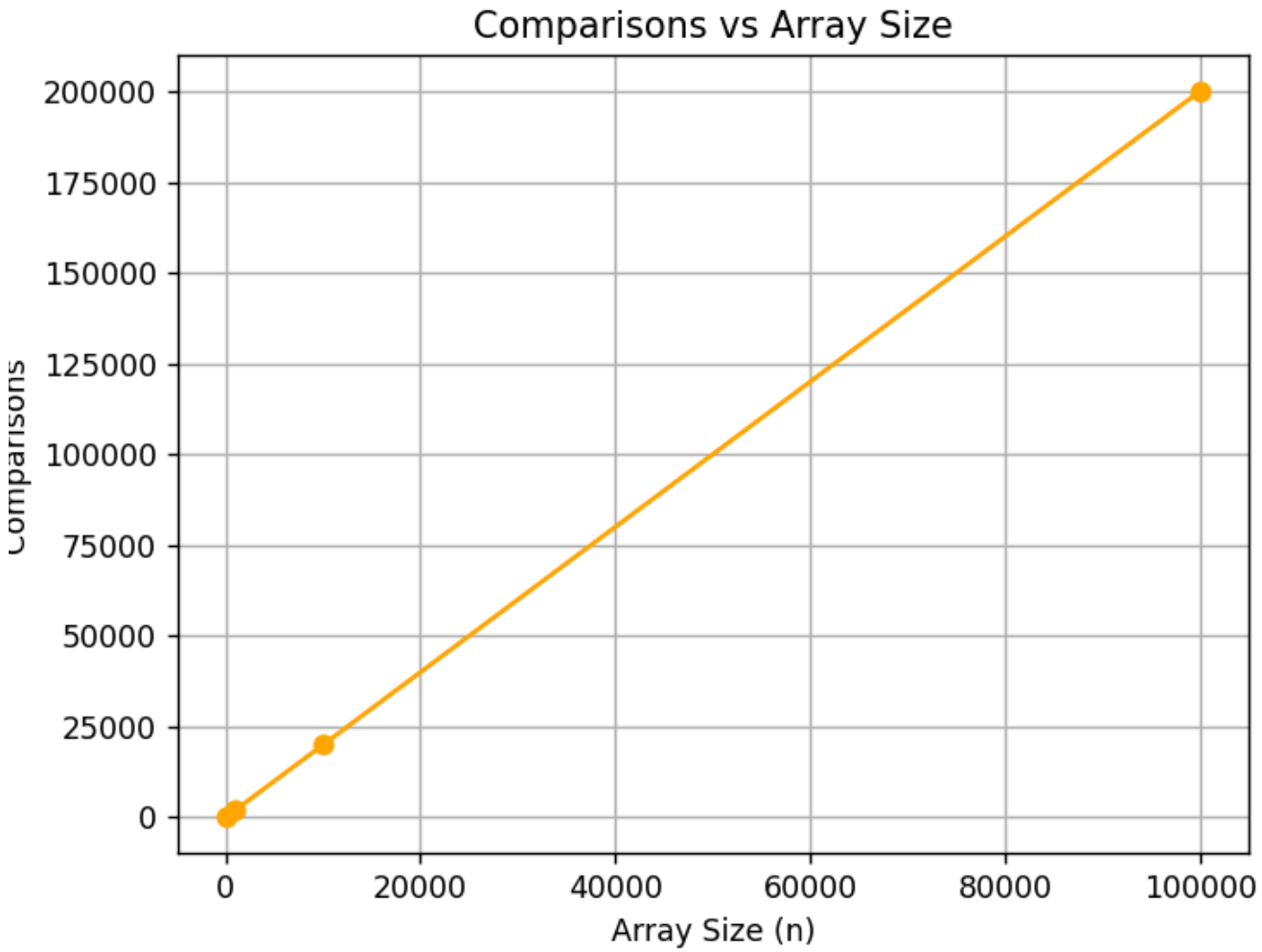

Comparisons vs Array Size
Comparisons
200000
175000
150000
125000
100000
75000
50000
25000
0
0
20000
40000
60000
80000
100000
Array Size (n)

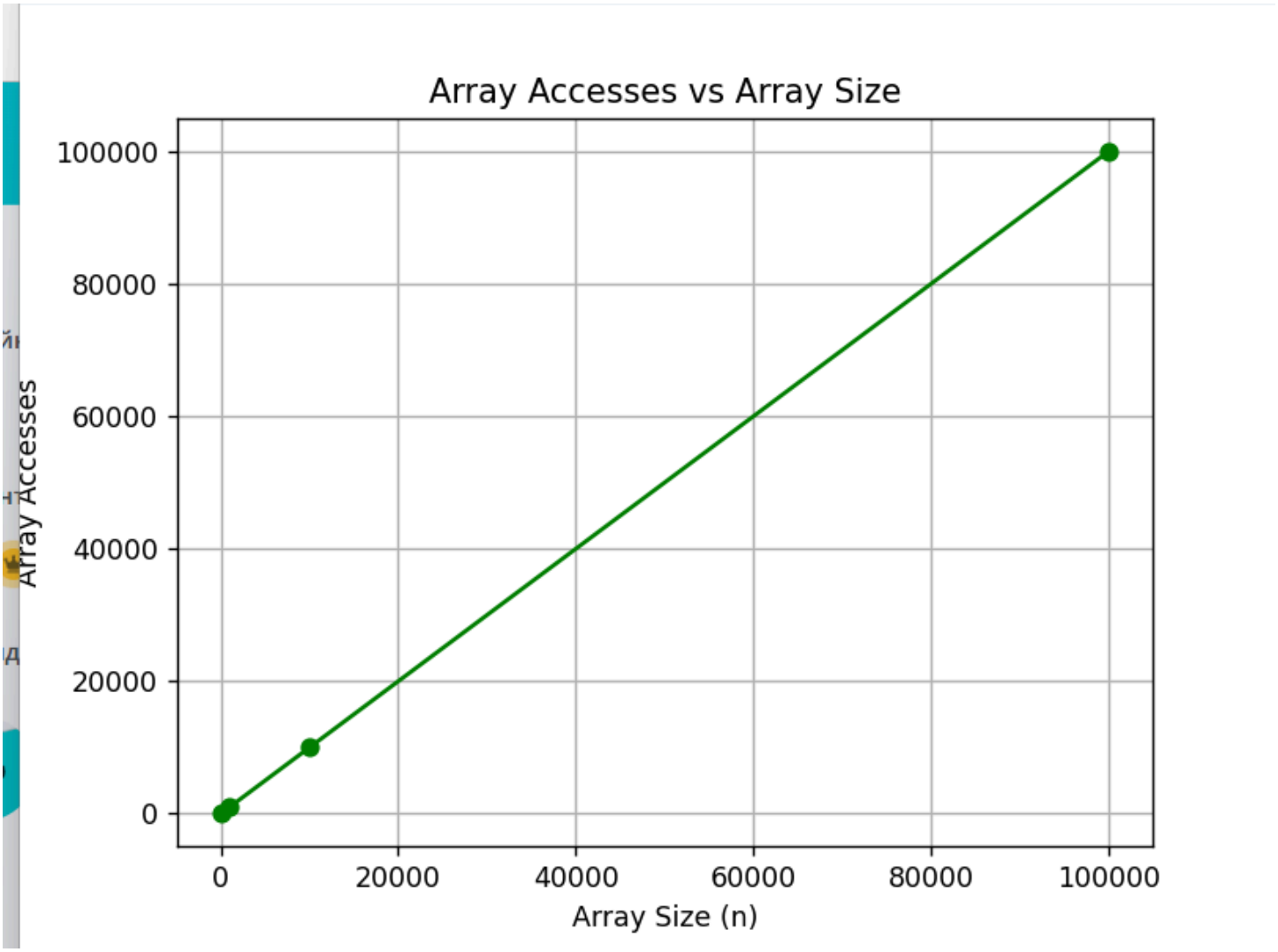

Array Accesses vs Array Size
Array Accesses
100000
80000
60000
40000
20000
0
0
20000
40000
60000
80000
100000
Array Size (n)

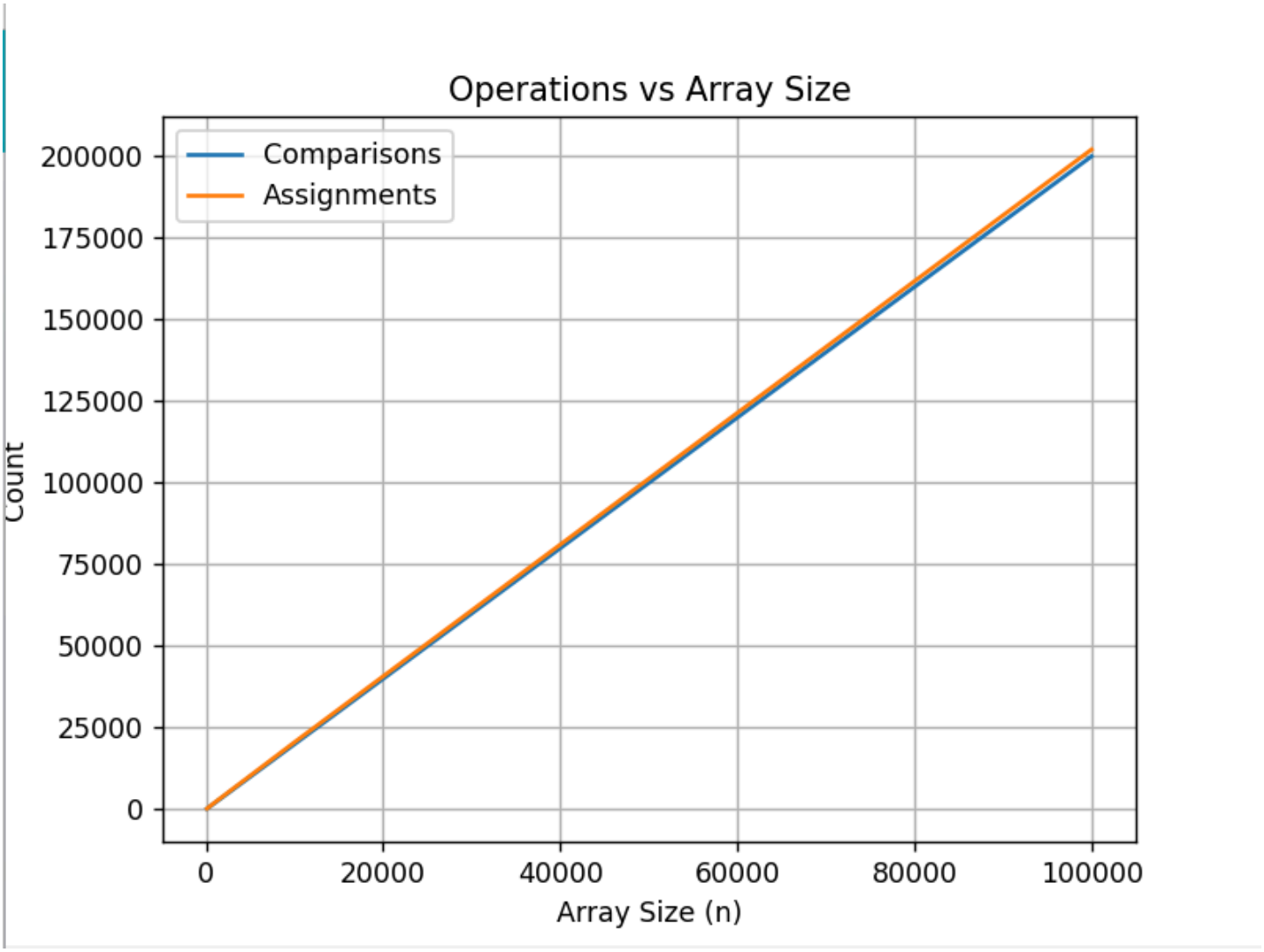


Optimized Kadane metrics

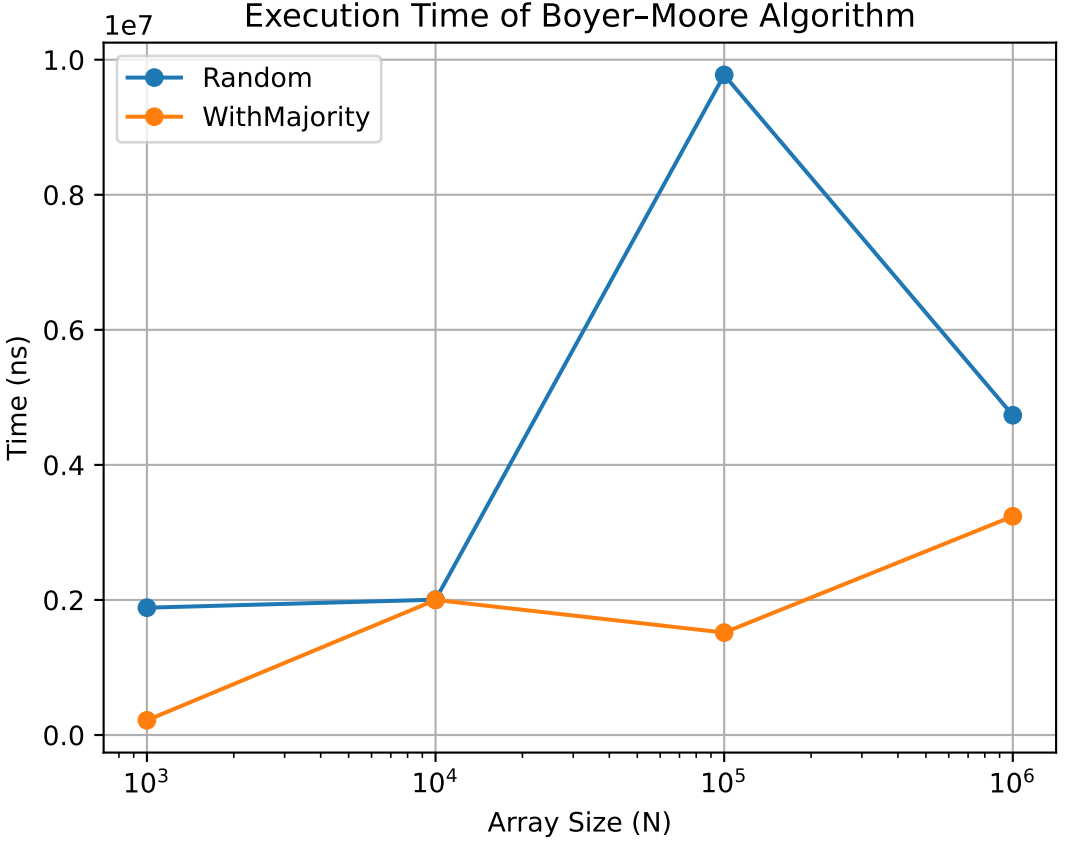

Execution Time vs Array Size
1e6
2.0
1.5
1.0
0.5
Execution Time (ns)
0
20000
40000
60000
80000
100000
Array Size (n)

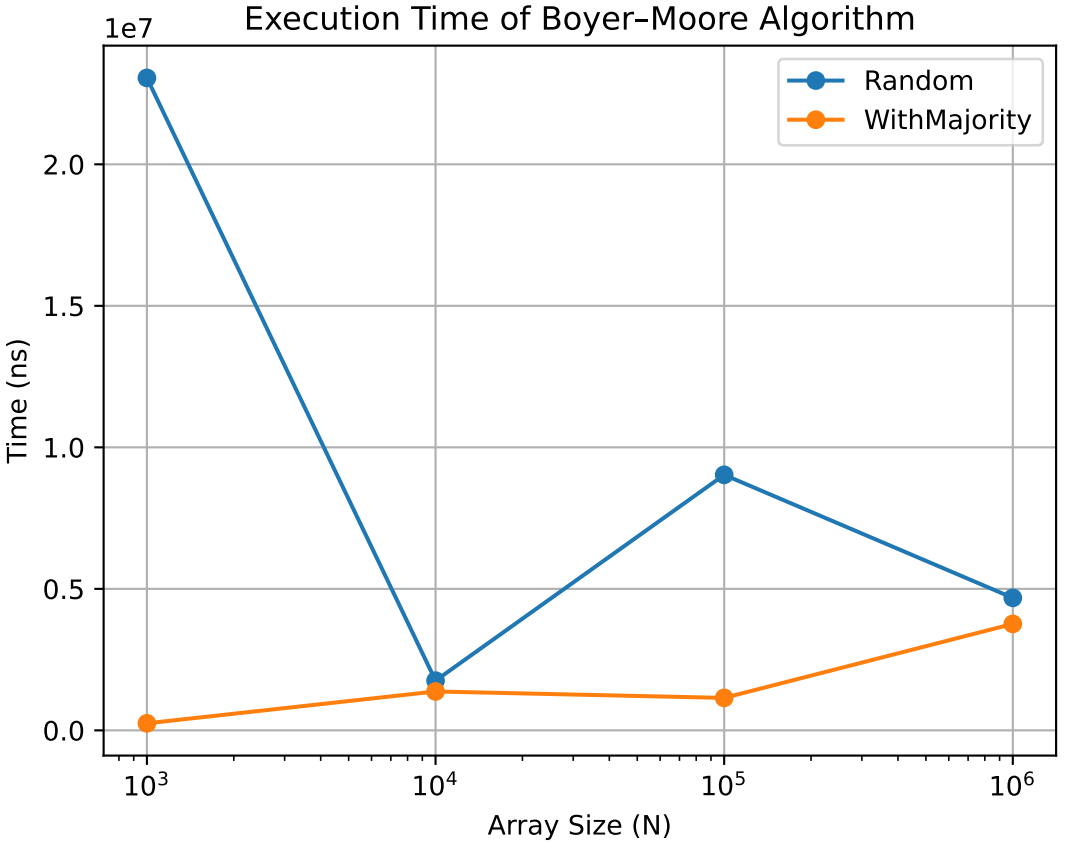

Comparisons vs Array Size
Comparisons
200000
175000
150000
125000
100000
75000
50000
25000
0
0
20000
40000
60000
80000
100000
Array Size (n)

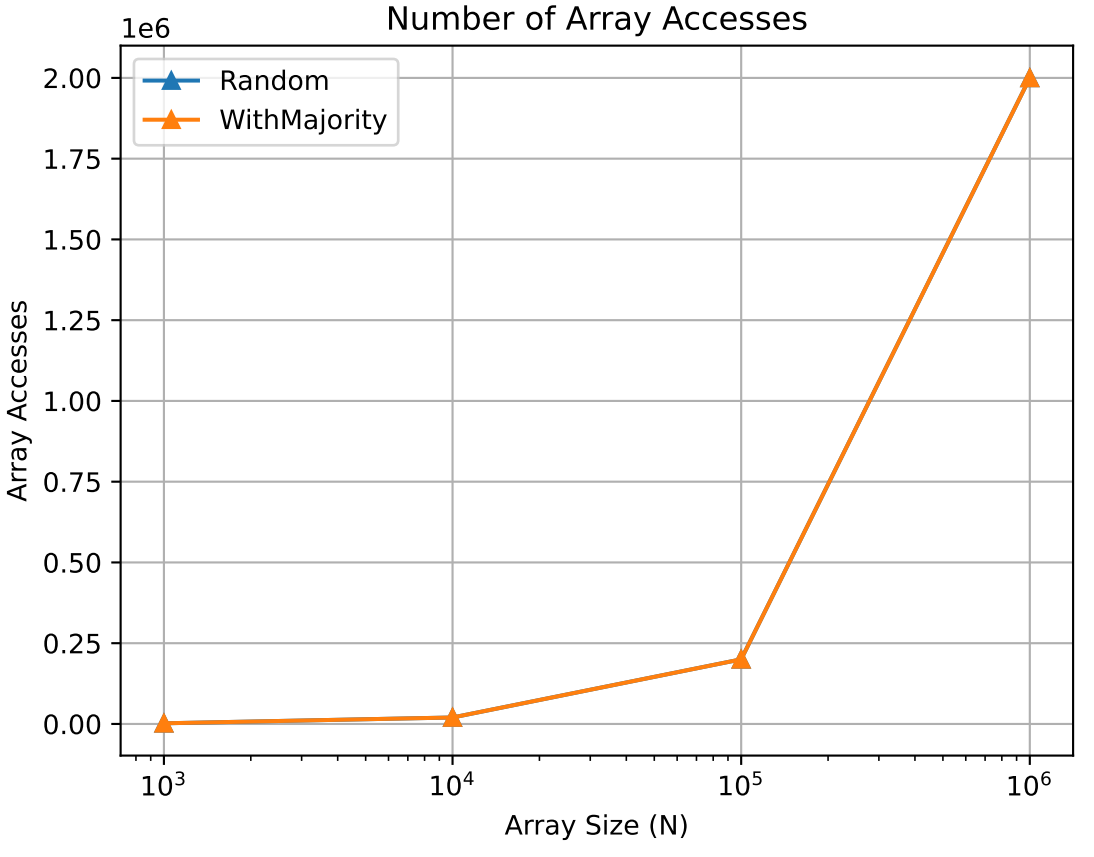

Array Accesses vs Array Size
Array Accesses
Array Size (n)
0
20000
40000
60000
80000
100000

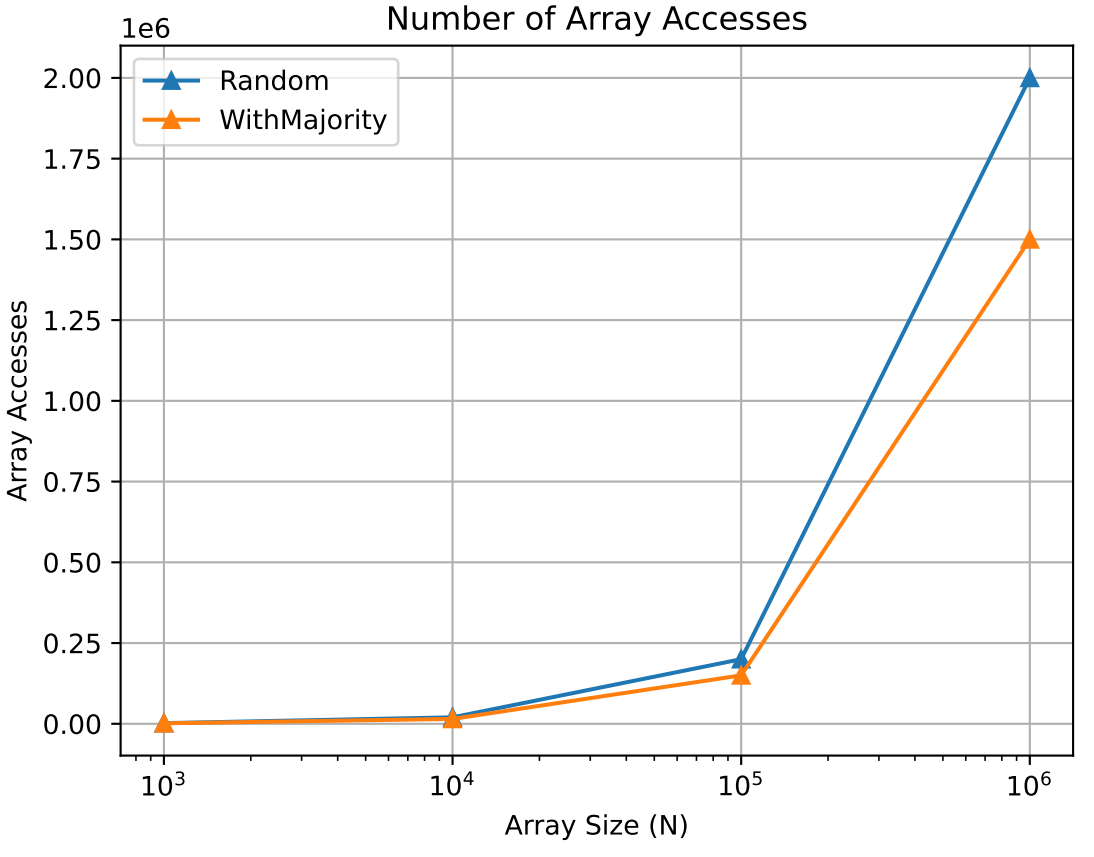


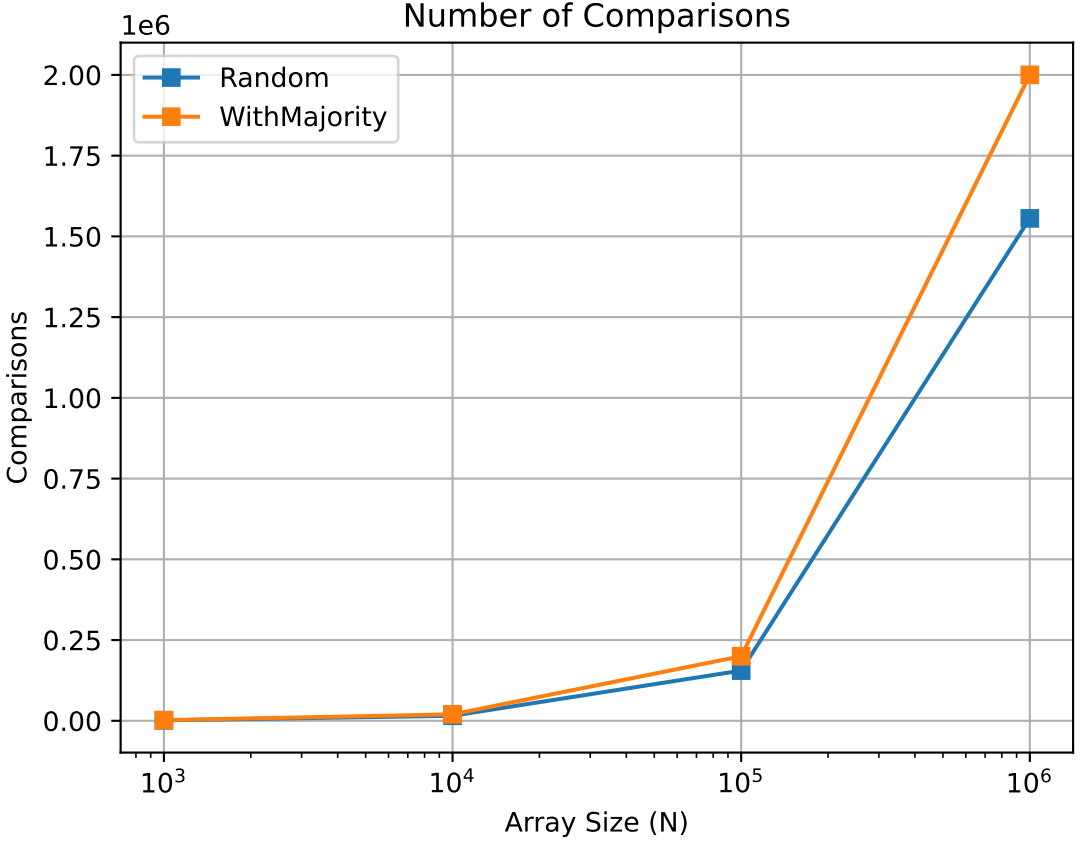
On graphs we can see small changes becocuse Kadane is optimized algotirthm and small code optmiztaions will more visible on weak CPUs

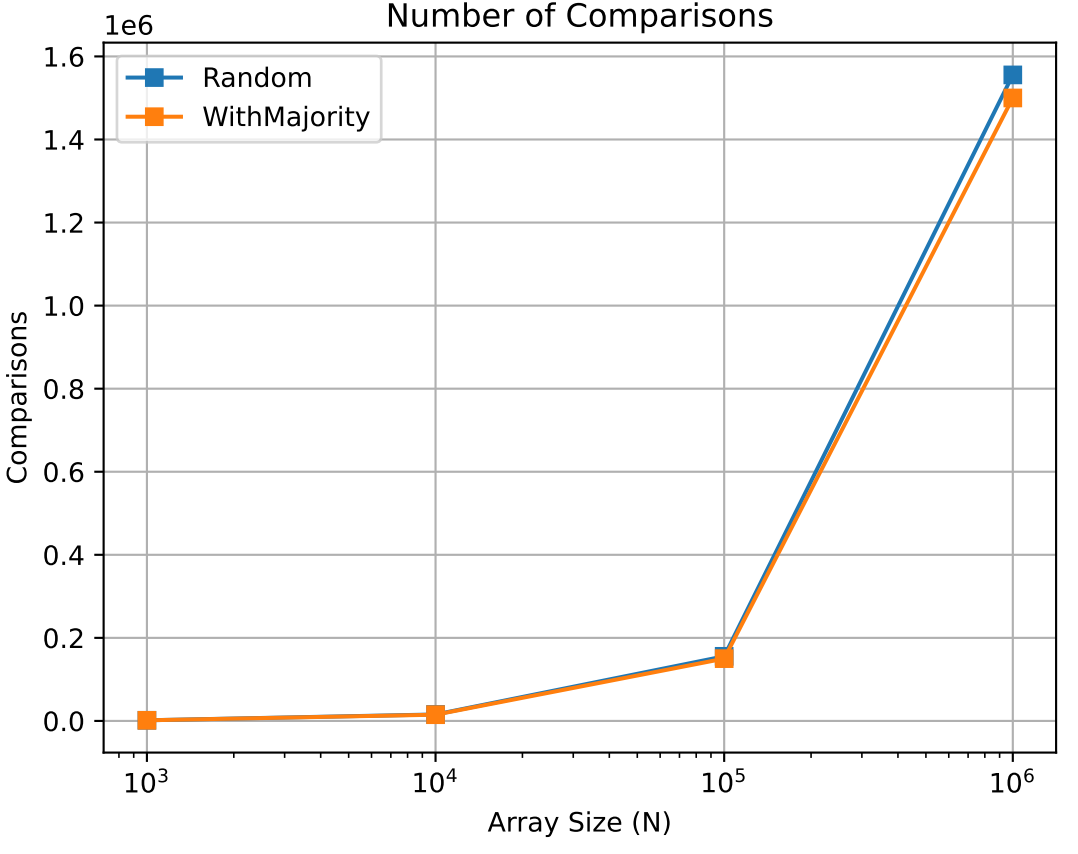

Execution Time of Boyer–Moore Algorithm
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Time (ns)
Random
WithMajority
10^3
10^4
10^5
10^6
Array Size (N)


Execution Time of Boyer–Moore Algorithm
1e7
Random
WithMajority
2.0
1.5
1.0
0.5
0.0
Time (ns)
10^3
10^4
10^5
10^6
Array Size (N)


Number of Array Accesses
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Array Accesses
Random
WithMajority
10^3
10^4
10^5
10^6
Array Size (N)


Number of Array Accesses
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Array Accesses
Random
WithMajority
10^3
10^4
10^5
10^6
Array Size (N)


Number of Comparisons
1e6
Random
WithMajority
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Comparisons
10^3
10^4
10^5
10^6
Array Size (N)


Number of Comparisons
1e6
Random
WithMajority
Comparisons
Array Size (N)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
10^3
10^4
10^5
10^6