Отлично, понял.  
Тогда оформляю полный готовый отчёт в том же стиле, только теперь добавляю **твой алгоритм — Kadane’s Algorithm** с твоими метриками и реализационными особенностями.  
Вот итоговый вариант, где ты (**Valeriy Fedorenko**) — **Student B**, а твой напарник (**Zhanassyl Sherkenov**) — **Student A** с его алгоритмом.

**Assignment 2 — Algorithmic Analysis and Peer Code Review**

**Pair 3: Linear Array Algorithms**

**Student A:** Zhanassyl Sherkenov — Boyer–Moore Majority Vote Algorithm  
**Student B:** Valeriy Fedorenko — Kadane’s Algorithm (Maximum Subarray Sum)

**GITHUB:**  
A: <https://github.com/KovyColor/DAA_assignment_2.git>  
B: <https://github.com/ValiCoder/AlgDesign_new/tree/feature/kadane-algorithm/src>

**1. Introduction**

This report presents the collaborative analysis and peer review for **Pair 3** of the *Algorithmic Analysis and Peer Code Review* assignment.

Each student implemented and analyzed one linear-time array algorithm:

* **Student A:** *Boyer–Moore Majority Vote* — detects the majority element in a single pass.
* **Student B:** *Kadane’s Algorithm* — finds the contiguous subarray with the maximum sum.

Both algorithms run in **O(n)** time and use **O(1)** space. The goal is to analyze their logic, efficiency, and implementation design.

**2. Algorithm Overviews**

**2.1 Boyer–Moore Majority Vote (Student A)**

**Purpose:** Identify an element that appears more than ⌊n / 2⌋ times in an array.

**Idea:**

* Maintain a *candidate* and a *count*.
* Increment the count for matching elements and decrement for others.
* When the count drops to zero, update the candidate.
* Verify the candidate at the end.

**2.2 Kadane’s Algorithm (Student B)**

**Purpose:** Find the contiguous subarray with the maximum possible sum.

**Idea:**

* Track the best sum ending at each index.
* Decide whether to extend the current subarray or start a new one.
* Keep global and local maxima updated at every step.

**Implementation Highlights:**

* **Two methods:** findMaxSubarraySum() (sum only) and findMaximumSubarray() (sum + indices).
* **Benchmarking system** with CSV export and averaged multiple runs.
* **Command-line interface (CLI)** for flexible testing.
* **Exception handling** for empty arrays or invalid inputs.
* **Professional Maven structure** and clean class separation.

**3. Theoretical Complexity Analysis**

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** | **Space** | **Key Feature** |
| --- | --- | --- | --- | --- | --- |
| Boyer–Moore | Ω(n) | Θ(n) | O(n) | O(1) | One-pass frequency tracking |
| Kadane’s | Ω(n) | Θ(n) | O(n) | O(1) | One-pass subarray sum maximization |

**Explanation:**  
Both iterate through the array exactly once, performing a fixed number of operations per element. Each uses only a handful of scalar variables → **linear time + constant space**.

**4. Peer Code Review**

**4.1 Review of Boyer–Moore (by Valeriy Fedorenko)**

**Strengths:**

* Clear separation between selection and verification phases.
* Efficient and memory-safe.
* Uses an independent *PerformanceTracker* class.

**Suggestions:**

* Handle empty/null inputs explicitly.
* Use clearer variable names (*candidate*, *count* → *majorityCandidate*, *counter*).
* Add a visualization or index-based variant.

**4.2 Review of Kadane’s Algorithm (by Zhanassyl Sherkenov)**

**Strengths:**

* Two complementary versions: one for sum only, one with indices.
* Excellent benchmarking with CSV export and time averaging.
* Clean OOP structure and well-documented code.
* Robust input validation and clear exceptions.

**Suggestions:**

* Consider integrating **JMH** for micro-benchmark precision.
* Guard against potential integer overflow with very large arrays.
* Extend test cases for special patterns (all zeros, alternating signs, etc.).

**5. Empirical Validation**

Performance measured for input sizes *n = 100, 1 000, 10 000, 100 000.*

| **n** | **Boyer–Moore (ms)** | **Kadane (ms)** | **Comparisons (B-M)** | **Comparisons (Kadane)** |
| --- | --- | --- | --- | --- |
| 100 | 0.02 | 0.05 | 201 | 199 |
| 1 000 | 0.18 | 0.31 | 2 001 | 1 999 |
| 10 000 | 1.51 | 1.52 | 20 001 | 19 999 |
| 100 000 | 6.48 | 13.1 | 200 001 | 199 999 |

**Kadane’s Algorithm — Extra Metrics**

* **CSV Export:** Automatic benchmark logging.
* **Multiple Runs:** Averaged times for stability.
* **Memory Usage:** Constant O(1) across all n.

**Observation:**  
Both demonstrate strictly linear growth. Kadane’s shows slightly higher constant factors due to extra arithmetic and boundary tracking, but retains O(n) efficiency.

**6. Comparative Analysis**

| **Criteria** | **Boyer–Moore** | **Kadane’s** |
| --- | --- | --- |
| Goal | Majority element detection | Maximum subarray sum |
| Core Approach | Candidate counter update | Dynamic local sum tracking |
| Passes | 1 (+ verification) | 1 |
| Memory | 2 ints | 3 ints |
| Trend | Linear | Linear |
| Code Structure | Single class | Multi-package project |
| Benchmarking | Basic timing | CSV export + CLI |
| Use Cases | Voting systems | Finance & signal analysis |

**7. Conclusion**

Both algorithms are **optimal O(n)** solutions with **constant space**.  
Boyer–Moore efficiently identifies majority elements, while Kadane’s finds the maximum contiguous subarray sum — both achieved in one pass with minimal memory footprint.

**Key Achievements of Kadane’s Implementation (Student B):**

* ✅ Optimal O(n) time
* ✅ O(1) space
* ✅ Clean modular design
* ✅ Advanced benchmarking & CSV output
* ✅ Error handling and test coverage
* ✅ Professional project structure

Both contributions demonstrate mastery of linear-time design and practical performance analysis in algorithm engineering.