Peer Review Report – Insertion Sort Analysis

Assignment 2: Algorithmic Analysis and Peer Code Review

Pair 1: Basic Quadratic Sorts

Reviewer: Agabekuly Asylbek (SE-2438)

Reviewed Student: Almukhamedov Temirlan – Insertion Sort

1. Algorithm Overview

The reviewed code implements the classical Insertion Sort algorithm, which builds the sorted part of the array one element at a time by inserting each new element into its correct position among previously sorted elements. The algorithm tracks performance metrics using a PerformanceTracker instance that measures comparisons, swaps, and execution time.

2. Code Review

The implementation is clean, readable, and follows good programming practices. The code properly checks for null and single-element arrays at the beginning, preventing unnecessary computation. It uses an early-exit condition: if the current element is already greater than or equal to the previous one, the algorithm skips unnecessary inner-loop iterations. This optimization significantly improves performance for nearly sorted data.

However, the code could be slightly improved by modularizing the comparison and shifting logic into helper methods for better readability and testing. Adding comments explaining the role of each block would also help for maintainability.

3. Complexity Analysis

Case	Time Complexity	Space Complexity	Explanation
Best	$\Theta(n)$	$O(1)$	Nearly sorted array – early exit prevents many comparisons
Average	$\Theta(n^2)$	$O(1)$	Random data – typical insertion behavior
Worst	$\Theta(n^2)$	$O(1)$	Reverse-sorted data – maximum number of shifts

4. Empirical Observation

Based on theoretical expectations and typical measurements, Insertion Sort performs linearly for small or sorted arrays and quadratically for large unsorted ones. For example, on arrays up to 100,000 elements, its time growth pattern closely follows $O(n^2)$, while for small arrays ($n < 1000$), it outperforms Selection Sort because it performs fewer swaps.

5. Optimization Suggestions

- Implement binary search to find the correct insertion position to reduce comparisons.
- Move the PerformanceTracker instance to a static or external utility to measure multiple runs.
- Add additional metrics for memory access count for deeper performance insight.

6. Conclusion

The reviewed Insertion Sort implementation by Almukhamedov Temirlan is correct, efficient for small and nearly sorted datasets, and well-structured. The early termination optimization improves performance compared to a naive version. Overall, the implementation meets assignment requirements and demonstrates clear algorithmic understanding.