Cross-Review Summary

Assignment 2: Algorithmic Analysis and Peer Code Review

Group: SE-2416

Students:

- *Student A:* Altynay (Insertion Sort with optimizations for nearly-sorted data)
- *Student B:* Nuray (Selection Sort with early termination optimizations)

1. Introduction

The goal of this cross-review is to compare two fundamental quadratic sorting algorithms: **Insertion Sort (Student A)** and **Selection Sort (Student B)**. Each algorithm was implemented with optimizations, benchmarked on multiple input distributions, and analyzed for both theoretical and empirical performance.

2. Algorithm Overview

Insertion Sort (Student A – Altynay)

- Builds the sorted prefix step by step by inserting elements into the correct position.
- **Optimizations:**
  - Early termination for nearly-sorted arrays.
  - Reduced element shifting when no moves are needed.
- **Properties:**
  - Stable.
  - Best case: $\Omega(n)$ (already sorted).
  - Worst/average: $O(n^2)$.

Selection Sort (Student B – Nuray)

- Repeatedly selects the minimum (and in optimized version also the maximum) and swaps them into position.
- **Optimizations:**
  - Two-way selection (min and max in one pass).
  - Early termination if no swaps occur.
  - Sorted suffix detection.
- **Properties:**
  - Not stable.
  - Best case: $\Omega(n)$ with early exit.
  - Worst/average: $O(n^2)$.
  - Very few swaps compared to Insertion Sort.

3. Theoretical Comparison

Criterion	Insertion Sort (Altynay)	Selection Sort (Nuray)
Best Case	$\Omega(n)$ on sorted input	$\Omega(n)$ with early exit
Average Case	$\Theta(n^2)$	$\Theta(n^2)$
Worst Case	$O(n^2)$	$O(n^2)$
Stability	Stable	Not stable
Swaps	Up to $O(n^2)$ (many shifts)	$\leq n$ (very few swaps)
Use Case Strength	Nearly-sorted arrays	Expensive swap operations

4. Code Review Observations

- **Student A (Insertion Sort)**
  - Clean modular implementation.
  - Handles nearly-sorted arrays very efficiently.
  - Improvement opportunity: reduce redundant comparisons inside inner loop.
  - Suggestion: implement binary search insertion to reduce comparisons.
- **Student B (Selection Sort)**
  - Well-structured with a `twoWay` flag for classic vs optimized mode.
  - Integrated metrics collection (comparisons, swaps, accesses).
  - Improvement opportunity: handle overlapping min/max swap cases more carefully.
  - Suggestion: remove extra array scans in `isSortedSuffix()`.

5. Empirical Results (Summary)

(*Detailed graphs provided in individual reports.*)

- **Sorted Input:**
  - Insertion Sort runs nearly linear, extremely fast.
  - Selection Sort also improves via early termination, but still performs a full scan.
- **Random Input:**
  - Both show quadratic growth.
  - Selection Sort slightly reduces passes with two-way optimization.
- **Reverse Input:**
  - Both degrade to $O(n^2)$.
  - Selection Sort performs fewer swaps, while Insertion Sort performs many shifts.
- **Nearly-Sorted Input:**
  - Insertion Sort is significantly faster due to minimal shifts.
  - Selection Sort still quadratic, though better than naive version.

6. Conclusion

Both algorithms remain quadratic in time complexity, but their optimizations make them suitable for different scenarios:

- **Insertion Sort (Altynay)** is highly efficient for nearly-sorted datasets and maintains stability.
- **Selection Sort (Nuray)** minimizes the number of swaps, which is beneficial when swaps are expensive operations.

The theoretical analyses were validated by empirical benchmarks, confirming the expected trends. Each implementation can be further optimized: binary search insertion for Insertion Sort, and more efficient suffix detection for Selection Sort.

This cross-review highlights not only the individual strengths and weaknesses of each algorithm but also the importance of selecting the right tool based on input characteristics.