Peer Analysis Report — Insertion Sort

Course: Algorithmic Analysis and Peer Code Review

Prepared by: Armat Islambekov

Partner: Marsel

Group: SE-2403

Algorithm Under Review: Insertion Sort (Optimized for Nearly-Sorted Data)

Institution: Astana IT University (AITU)

1. Algorithm Overview

The reviewed algorithm is Insertion Sort, implemented by Marsel as part of Assignment 2 (Pair 1, Student A).   
Insertion Sort is a simple comparison-based sorting algorithm with quadratic time complexity in the general case.   
The implementation correctly follows the iterative insertion principle, where each new element is compared with previously sorted elements and inserted into its correct position.

For nearly-sorted data, Insertion Sort performs efficiently due to minimal element shifting. However, the project lacks explicit optimizations,   
such as binary insertion search or early termination checks that could further reduce the number of comparisons.

2. Theoretical Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Case | Time Complexity | Space Complexity |
| Best Case (nearly sorted) | O(n) | O(1) |
| Average Case | O(n²) | O(1) |
| Worst Case (reverse sorted) | O(n²) | O(1) |

Insertion Sort has linear behavior on nearly-sorted arrays, which makes it practical for datasets where minor displacements occur.   
The lack of auxiliary structures ensures O(1) space complexity, but the time complexity grows quadratically as disorder increases.

3. Code Review and Optimization

The code is syntactically correct, readable, and uses standard Java conventions. The repository follows a partial Maven structure,   
but lacks separation into packages (`algorithms/`, `metrics/`, `cli/`). There are no metrics collection or benchmarking modules implemented.

Recommendations for improvement:  
1. \*\*Binary Insertion Optimization\*\* — use binary search to locate insertion position, reducing comparisons from O(n²) to O(n log n) for nearly-sorted data.  
2. \*\*Metrics Collection\*\* — integrate a `PerformanceTracker` class to log comparisons, swaps, and memory allocations.  
3. \*\*Benchmark CLI\*\* — implement a `BenchmarkRunner` allowing testing on arrays of size n = 100 to 100,000.  
4. \*\*Edge Case Handling\*\* — add input validation for empty or single-element arrays.  
5. \*\*Code Comments\*\* — document algorithm steps for clarity.

4. Empirical Validation

The repository lacks CSV-based benchmark results, which are critical for validating theoretical complexity.   
For empirical testing, the project should measure runtime for datasets of different sizes and generate a CSV file   
to visualize time vs input size using performance plots.

Example expected results for Insertion Sort:  
- Nearly-sorted arrays: time growth ≈ linear  
- Random arrays: time growth ≈ quadratic  
- Reverse-sorted arrays: time growth ≈ quadratic

5. GitHub Workflow and Communication

The repository has limited commit activity (only 4 commits) and no clear branching strategy.   
Commits lack descriptive messages, and the `.idea/` directory is tracked in version control.   
No README file is provided, which makes the repository hard to evaluate without prior context.

Recommendations:  
- Create feature branches (`feature/algorithm`, `feature/metrics`, etc.).  
- Add `.gitignore` to exclude `.idea`, `target`, `.DS\_Store`.  
- Include `README.md` describing algorithm logic, build/run instructions, complexity summary, and benchmark guide.

6. Evaluation Summary

|  |  |  |
| --- | --- | --- |
| Criterion | Weight | Score |
| Implementation Quality | 40% | 28 / 40 |
| Analysis Depth | 35% | 22 / 35 |
| Empirical Validation | 15% | 8 / 15 |
| Communication & Workflow | 10% | 5 / 10 |
| Final Grade | 100% | 63 / 100 (Satisfactory) |

7. Conclusion

The Insertion Sort implementation is functional but minimal. While it correctly sorts arrays, it lacks optimization, benchmarking,   
and analytical documentation required by the assignment. The project can reach a higher grade by adding empirical validation,   
structured code organization, and a proper Git workflow.

Overall, the algorithm demonstrates correct logic but requires additional work in analysis depth and presentation   
to meet the standards of Assignment 2 at AITU.