Algorithm Analysis Report – Insertion Sort

Assignment 2 – Algorithmic Analysis and Peer Code Review
Student Reviewed: Gaziza Bakir
Algorithm Reviewed: Insertion Sort
Reviewer: Sultan Agibaev
Date: 5 October 2025

1. Introduction

This report presents a detailed analysis of the *Insertion Sort* algorithm implemented by Gaziza Bakir. The purpose is to evaluate the correctness, algorithmic efficiency, and empirical performance, as well as provide recommendations for potential improvements.

2. Algorithm Overview

Insertion Sort is a comparison-based sorting algorithm that builds a sorted array incrementally. Each element is inserted into its correct position within the already sorted portion of the array.

Pseudocode:

```
for i ← 1 to n − 1 do
    key ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > key do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← key
```

Characteristics:

- In-place sorting (O(1) additional space)
- Stable: preserves relative order of equal elements
- Efficient for small or partially sorted datasets
- Quadratic time complexity in the worst case

3. Complexity Analysis

Case	Time Complexity	Explanation
Best Case	$\Omega(n)$	Array already sorted; only one comparison per element
Average Case	$\Theta(n^2)$	Each element compared with roughly half of the sorted portion
Worst Case	$O(n^2)$	Reverse-sorted array; maximum shifts required
Space Complexity	$O(1)$	In-place sorting, no extra memory needed
Stability	Yes	Equal elements retain their order

4. Empirical Performance

The algorithm was benchmarked using `BenchmarkRunner` with multiple input sizes.

Benchmark Table:

	A	B	C	D
1	Input Size (	Time (ms)	Operations	
2	100	1	9,536	
3	1000	10	1,026,268	
4	10000	61	100,158,750	
5	50000	1440	2,507,376,862	
6				

Observation:
Execution time and operations grow quadratically with input size, confirming the theoretical $O(n^2)$ behavior.

5. Code Review and Optimization Suggestions

Strengths:

- Correct and clean Java implementation
- Proper integration of `PerformanceTracker`
- Unit tests cover edge cases
- Algorithm is readable and maintainable

Potential Improvements:

1. Adaptive insertion: Use binary search to reduce comparisons.
2. Hybrid algorithm: Combine Insertion Sort with Merge Sort for larger datasets.
3. Input randomization for unbiased performance testing.
4. Minimize metric tracking overhead for large arrays.

6. Conclusion

The Insertion Sort implementation is correct, stable, and well-documented.
Experimental results confirm theoretical $O(n^2)$ complexity. While efficient for small datasets, performance is limited for large arrays. Suggested optimizations can improve performance for specific cases.

Overall Evaluation:

- Implementation Quality: Excellent
- Code Readability: Excellent
- Efficiency for Large Inputs: Limited
- Documentation and Reporting: Excellent