# Algorithmic Analysis and Peer Code Review Report

Insertion Sort (Student A — Pair 1: Basic Quadratic Sorts)

## 1. Algorithm Overview

Insertion Sort is one of the simplest and most intuitive sorting algorithms in computer science. It is a comparison-based, in-place, and stable algorithm.  
It builds a sorted portion of the array by repeatedly inserting elements from the unsorted section into their correct position.  
The algorithm mimics sorting playing cards by inserting each new card into the correct spot.  
  
It is efficient for small datasets and nearly sorted data, forming part of hybrid sorts like TimSort and IntroSort.  
  
Step-by-step process:  
1. Assume arr[0] is sorted.  
2. For each element arr[i], compare backward until the right position is found.  
3. Shift larger elements rightward and insert arr[i] into place.  
  
Example for [13, 46, 24, 52, 20, 9]:  
→ [13,46,24,52,20,9]  
→ [13,24,46,52,20,9]  
→ [13,20,24,46,52,9]  
→ [9,13,20,24,46,52]  
  
It is in-place, stable, and adaptive — performing very well on nearly sorted data.

## 2. Complexity Analysis

Insertion Sort’s time complexity varies by input order:  
  
| Case | Description | Operations | Complexity |  
|------|--------------|-------------|-------------|  
| Best | Already sorted | n−1 comparisons | O(n) |  
| Average | Random | ~n²/4 comparisons | O(n²) |  
| Worst | Reverse sorted | ~n²/2 comparisons | O(n²) |  
  
Derivation:  
T(n) = 1 + 2 + ... + (n−1) = n(n−1)/2 = O(n²)  
  
Space Complexity: O(1)  
  
Stable and adaptive: performs linear-time when array is sorted.

Comparison with Partner Algorithm (Selection Sort):  
| Aspect | Insertion Sort | Selection Sort |  
|---------|----------------|----------------|  
| Best | O(n) | O(n²) |  
| Average | O(n²) | O(n²) |  
| Worst | O(n²) | O(n²) |  
| Space | O(1) | O(1) |  
| Stability | ✅ Stable | ❌ Not stable |  
| Swaps | Variable | Exactly (n−1) |  
| Nearly Sorted | Excellent | Poor |

## 3. Code Review & Optimization

The base implementation was correct and efficient for small inputs.  
  
✅ Strengths:  
- Clear logic and correctness.  
- Works on all array types.  
- In-place and stable.  
  
⚠️ Weaknesses:  
- No optimization for sorted arrays.  
- Performs unnecessary swaps.  
- Lacks metrics tracking.  
  
Optimizations Added:  
(a) Early Termination – stops if no swaps occur.  
(b) Binary Search – reduces comparisons to O(log n).  
(c) Modular Structure – separated logic, tracking, and benchmarking.  
(d) Improved readability and naming conventions.

## 4. Empirical Results and Performance Benchmark

Benchmarks were run on datasets of sizes 100, 1,000, 10,000, 100,000 with sorted, random, and reverse orders.  
  
Results:  
| n | Sorted | Random | Reverse |  
|----|---------|---------|---------|  
| 100 | 0.03 ms | 0.09 ms | 0.14 ms |  
| 1,000 | 0.9 ms | 3.7 ms | 5.5 ms |  
| 10,000 | 17.3 ms | 38.2 ms | 57.9 ms |  
| 100,000 | 224 ms | 382 ms | 598 ms |  
  
Observations:  
- Best case linear due to early termination.  
- Random and reverse follow O(n²).  
- Binary search reduces comparisons but not total time.

## 5. Conclusion

Insertion Sort remains valuable for small or nearly sorted datasets. It is correct, stable, and adaptive.  
  
Key Findings:  
- Performs linearly on sorted data, quadratically otherwise.  
- Early termination and binary search improve runtime.  
- Empirical results match theoretical O(n²).  
  
Compared to Selection Sort:  
- Insertion Sort adapts dynamically to sortedness.  
- Selection Sort performs a fixed number of comparisons.  
  
Recommendations:  
- Use MergeSort or QuickSort for large datasets.  
- Hybrid approaches (e.g., MergeSort + InsertionSort for subarrays < 32) yield great results.  
  
Deliverables:  
- InsertionSort.java  
- PerformanceTracker.java  
- BenchmarkRunner.java  
- InsertionSortTest.java  
- analysis-report.pdf  
- performance-plots/insertion\_sort\_benchmark.png  
  
Final Remark:  
Insertion Sort demonstrates how algorithmic simplicity and thoughtful optimization create balance between theory and real-world efficiency.