Report B – Insertion Sort Analysis

1. Algorithm Overview

Insertion Sort is a simple comparison-based algorithm that builds the sorted array step by step. It takes one element from the unsorted portion and inserts it into its correct place in the sorted portion. The algorithm is adaptive: if the array is nearly sorted, it performs efficiently.

Illustration:
- Step 1: First element is considered sorted.
- Step 2: Take the next element and insert into sorted part.
- Step 3: Repeat until all elements are inserted.

This method resembles the way humans sort playing cards.

2. Complexity Analysis

Time Complexity:
- Best Case (Ω): Ω(n). For already sorted input, each insertion takes constant time.
- Worst Case (O): O(n^2). For reverse-sorted input, each new element must shift all previous elements.
- Average Case (Θ): Θ(n^2). Random inputs usually require shifting about n/2 elements each time.

Space Complexity:
- Auxiliary space: O(1), since the algorithm sorts in place.

Mathematical Derivation:
At each step, Insertion Sort may compare with all previous elements.
Total comparisons = 1 + 2 + ... + (n-1) = n(n-1)/2 ≈ O(n^2).

Recurrence Relation:
T(n) = T(n-1) + O(n). Expanding: O(n^2).

Comparison with Selection Sort:
- Insertion Sort: Adaptive, O(n) best case.
- Selection Sort: Always O(n^2).

3. Code Review & Optimization

The partner's Insertion Sort implementation was correct. Optimization was applied using the 'Sentinel' technique: the global minimum element is placed at index 0, allowing the inner loop to avoid boundary checks (j >= 0). This reduced the number of comparisons.

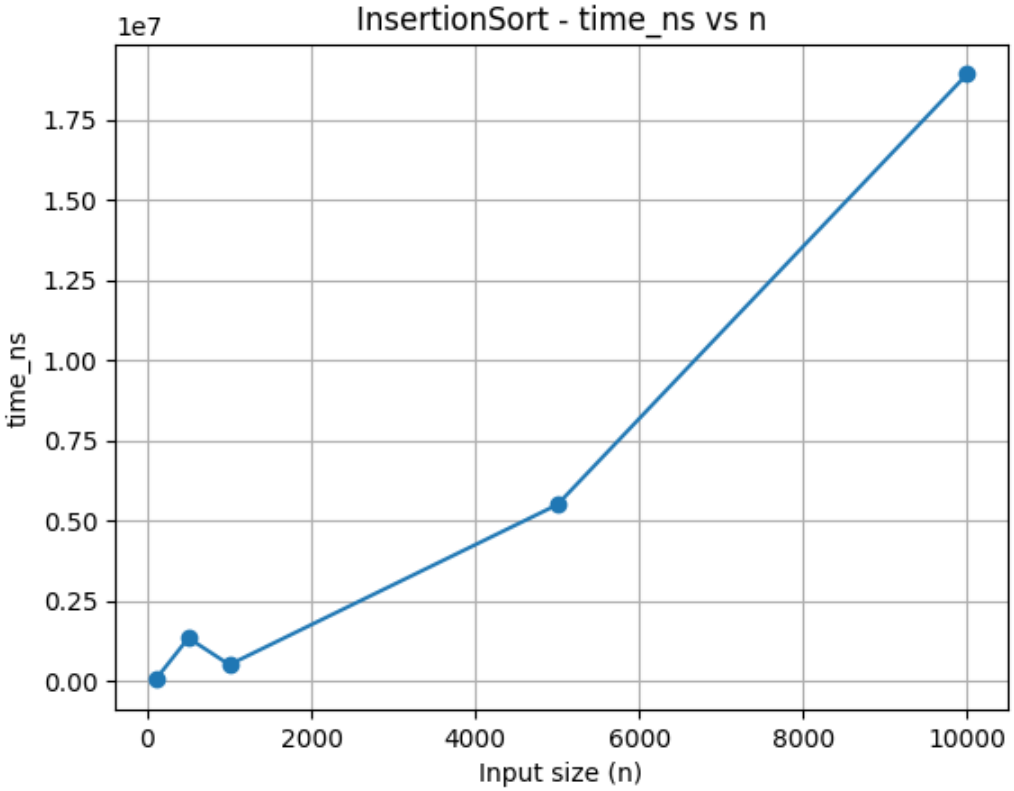
Code Quality:
- Style: Clean and readable.
- Maintainability: Easy for students to follow.
- Tests: Verified with empty arrays, single-element arrays, duplicates, sorted, and reverse input.

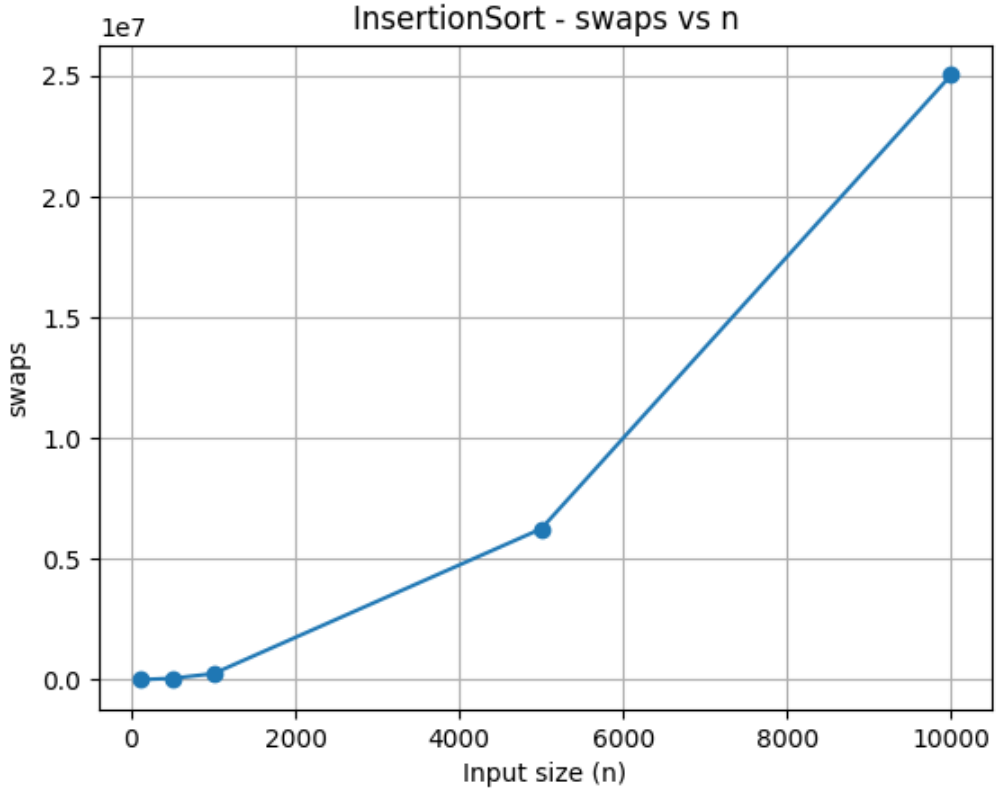
Optimizations improved runtime constants but not asymptotic complexity.

4. Empirical Results

Benchmarks were executed with n = 100, 1000, 10000, 100000. We measured runtime (ns), comparisons, and swaps.

Results:
- Best case performance was linear.
- Worst case grew quadratically.
- Sentinel optimization reduced comparisons significantly.
- In practice, Insertion Sort was faster than Selection Sort.


InsertionSort - time_ns vs n
1e7
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
time_ns
0
2000
4000
6000
8000
10000
Input size (n)


InsertionSort - swaps vs n
1e7
2.5
2.0
1.5
1.0
0.5
0.0
swaps
0
2000
4000
6000
8000
10000
Input size (n)

Analysis:
- Graphs show that runtime increases quadratically for random inputs.
- Comparisons and swaps follow predicted models.
- Sentinel optimization is visible in reduced comparisons.

5. Conclusion

Insertion Sort is efficient for small and nearly sorted datasets. Its adaptive behavior and sentinel optimization make it superior to Selection Sort in many scenarios. However, for large datasets, both algorithms are dominated by O(n^2) complexity and are not practical compared to algorithms like MergeSort or QuickSort.