Name: Assylay Shukirbay

Danel Li

Group: SE 2406

**Cross-review Summary & Optimization Results**

**Insertion Sort vs Selection Sort**

### **1. Introduction**

This document provides a comparative analysis of two sorting algorithms: **Selection Sort** (implemented by Assylay Shukirbay ) and **Insertion Sort** (implemented by Danel Li). Both algorithms are evaluated in terms of time complexity, space complexity, code quality, and practical performance. Additionally, optimizations are suggested and their effects measured.

**2. Algorithm Comparison**

| **Feature/Metric** | **Selection Sort** | **Insertion Sort** | **Notes** |
| --- | --- | --- | --- |
| Best-case Time Complexity | Θ(n²) | Θ(n) | Insertion Sort is adaptive, Selection Sort always scans array |
| Worst-case Time Complexity | Θ(n²) | Θ(n²) | Both quadratic |
| Average-case Time Complexity | Θ(n²) | Θ(n²) | Comparable |
| Space Complexity | O(1) | O(1) | Both in-place |
| Stability | Not stable | Stable | Insertion Sort preserves order |
| Adaptive | No | Yes | Insertion Sort faster for nearly sorted arrays |
| Metrics Tracked | Time, Comparisons, Swaps, Accesses | Time, Comparisons, Swaps, ArrayAccesses | Insertion Sort tracks array accesses explicitly |
| Code Quality & Readability | Clear, modular, includes optimized variant | Clear, uses binary search for insertion | Both maintainable; Insertion Sort slightly more complex due to binary search |

### **3. Strengths and Weaknesses**

**Selection Sort:**

* Simple logic and easy to understand.
* Optimized version detects already sorted arrays.
* Performance is predictable but inefficient on nearly sorted or large arrays.
* Not stable.

**Insertion Sort:**

* Adaptive and stable.
* Uses binary search to reduce comparisons, improving practical performance.
* Slightly more complex code.
* Better for small or nearly sorted arrays.

**4. Optimization Results**

**Benchmark Results - SelectionSortOptimized vs InsertionSortOptimized**

| **Array Size (n)** | **SelectionSortOptimized Time (ns)** | **Comparisons** | **Swaps** | **Array Accesses** | **InsertionSortOptimized Time (ns)** |
| --- | --- | --- | --- | --- | --- |
| 10 | 15,400 | 54 | 6 | 137 | 24,100 |
| 50 | 108,600 | 1,253 | 45 | 2,711 | 195,000 |
| 100 | 305,600 | 5,001 | 94 | 10,428 | 485,300 |
| 500 | 7,618,700 | 125,003 | 488 | 252,208 | 6,692,400 |
| 1,000 | 5,459,100 | 500,003 | 989 | 1,004,462 | 21,451,500 |
| 5,000 | 25,199,200 | 12,500,001 | 4,990 | 25,022,462 | – |
| 10,000 | 45,581,800 | 50,000,002 | 9,989 | 100,044,960 | 114,826,000 |

**Observations:**

* Execution time:  
  + SelectionSortOptimized is faster than InsertionSort for small arrays (n ≤ 100).
  + InsertionSortOptimized outperforms SelectionSort on medium arrays (n = 500, 1000) due to binary search reducing comparisons.
  + For very large arrays, SelectionSort scales worse (n ≥ 5000), but actual runtime depends on array distribution.
* Comparisons, swaps, array accesses:  
  + SelectionSortOptimized performs far more comparisons and array accesses as n grows.
  + InsertionSortOptimized reduces unnecessary swaps, especially for nearly sorted arrays.
* Efficiency notes:  
  + Binary search in InsertionSort significantly reduces comparisons.
  + SelectionSortOptimized benefits from min-max optimization and early exit on sorted arrays but remains less adaptive.

### **5. Conclusion**

* Insertion Sort is generally faster on small and nearly sorted arrays due to its adaptive and stable nature.
* Selection Sort Optimized shows some improvement over the basic Selection Sort but remains less efficient than Insertion Sort for most practical inputs.
* Both implementations are correct and maintainable.
* Recommended to use Insertion Sort for smaller or nearly sorted arrays, while Selection Sort is more straightforward for educational purposes.