Iskander's Algorithm (Insertion Sort)

1. Algorithm Overview

Insertion Sort is another simple quadratic sorting algorithm. On each step it takes the current element and places it in the correct position among the already sorted part of the array by shifting larger elements to the right. After each iteration, the prefix of the array becomes sorted.

In Iskander's code this algorithm is implemented together with a PerformanceTracker, which counts comparisons, swaps, array accesses and execution time. The implementation also includes a small optimization: if the element is already in the correct position (greater than or equal to the previous one), the inner loop is skipped. This makes the algorithm faster on nearly sorted arrays.

2. Complexity Analysis

- **Best Case:** $\Omega(n)$. If the array is already sorted, only one comparison per element is needed, no shifting.
- **Average Case:** $\Theta(n^2)$. On random data each element is compared and shifted on average n/2 times.
- **Worst Case:** $O(n^2)$. On reverse-sorted arrays every new element has to be moved to the beginning.
- **Space Complexity:** O(1), because sorting happens in place.

Comparison with Selection Sort (Alikhan's algorithm):

- Insertion Sort can be much faster on nearly sorted inputs ($\Omega(n)$), while Selection Sort is always quadratic.
- Insertion Sort may do more swaps in the worst case, but it adapts to input order.
- Selection Sort does fewer swaps (at most n-1), but never improves on sorted input.

3. Code Review

Good points:

- Implementation is clear and easy to follow.
- Optimization for nearly-sorted data is included.
- PerformanceTracker is integrated well.
- Unit tests cover different cases: empty, single element, duplicates, sorted and reversed arrays.

Problems:

- No explicit handling of null arrays.
- CLI prints large arrays directly, which is not practical for big inputs.
- BenchmarkRunner does not average results across multiple runs.

Suggestions:

- Add input validation (if arr == null return;).
- Limit output for big arrays (e.g. print only first 50 elements).
- Run benchmarks multiple times and average results for more stable timing.
- Save benchmark results automatically in CSV (already partly done).

4. Empirical Results

The benchmark runner tests sizes: 100, 1000, 5000, 10000, 20000. Example results:

n=100 comparisons=2734 swaps=2635 time≈0.178 ms
n=1000 comparisons=256059 swaps=255060 time≈4.388 ms
n=5000 comparisons=6218843 swaps=6213844 time≈11.570 ms

Findings:

- Comparisons and swaps grow as n^2, matching theoretical analysis.
- Execution time grows quickly with input size, but smaller than Selection Sort on nearly sorted inputs.
- For reverse-sorted arrays, Insertion Sort is slower because of many shifts.
- Overall, the algorithm adapts better to partially ordered data.

5. Conclusion

Insertion Sort is simple but shows better behavior on nearly-sorted data compared to Selection Sort. Iskander's implementation is correct, includes an optimization, and provides good test coverage. The benchmark confirms the expected $O(n^2)$ growth but also shows that in the best case the algorithm can approach linear time.

Compared with Alikhan's algorithm (Selection Sort), Insertion Sort is usually more practical, even though it can perform more swaps. Both implementations are good for study, but Insertion Sort is closer to how real sorting adapts to data.

Iskander did well.