Code Review Report: Insertion Sort (Student A) Implementation

Reviewer: Abdunur Amangeldiev

Author: Sergey Balakarev

Overview

The reviewed code provides a basic in-place Insertion Sort implementation for integer arrays, with optional integration of a PerformanceTracker to measure algorithmic metrics such as comparisons and array accesses. The algorithm builds a sorted prefix incrementally by inserting each element into its correct position. Two variants are implemented: one simple sort and one with metric tracking.

The implementation supports two variants:

- sort(int[] arr) — plain version without tracking.
- sort(int[] arr, PerformanceTracker tracker) — enhanced version that records detailed metrics.

```
public static void sort(int[] arr) {
    sort(arr, null);
}

/**
```

```
public static void sort(int[] arr, PerformanceTracker tracker) {
    if (arr == null || arr.length == 0) {
        return;
    }
```

Algorithm Summary

Insertion Sort builds the sorted portion of the array by inserting each new element into its proper position. Time Complexity: Best O(n), Average O(n^2), Worst O(n^2). Space Complexity: O(1), since it sorts in place.

Time Complexity:

- **Best case (already sorted):** O(n)
- **Average case:** O(n^2)
- **Worst case (reverse order):** O(n^2)

Space Complexity:

- O(1) — sorting is performed in-place.

Code Structure & Readability

The code is clear, well-documented, and uses descriptive variable names. The Javadoc comments effectively explain parameters and complexities. The metric tracking adds educational value.

Performance Considerations

The algorithm uses in-place operations and includes a best-case optimization check. The 'anyShift' variable adds minimal overhead. Tracker updates cause extra operations, but these are intentional for analysis purposes.

Strengths:

- **Clear Javadoc comments:** The class and methods are thoroughly documented with descriptions, parameter explanations, and complexity details.
- **Good naming conventions:** Variables such as unsortedIndex, sortedIndex, and valueToInsert clearly describe their purpose.
- **Structured method overloads:** The use of two sort methods (one with a tracker, one without) is elegant and maintains code reuse.

Minor Suggestions:

- The anyShift variable is used to detect if the array was already sorted. However, its benefit is limited — insertion sort naturally stops early when no shifts occur. Removing it or using a simpler “early-exit check” per iteration could simplify the logic.
- The tracker calls slightly clutter the core algorithm; wrapping metric updates in small helper methods could improve readability.

Code Quality & Maintainability

The formatting, documentation, and code structure are consistent and professional. Further clarity could be achieved by moving inner-loop logic to a helper method and adding example usage in the Javadoc.

Testing Recommendations

The algorithm should be tested on sorted, reverse-sorted, duplicate, single-element, and random arrays, as well as arrays containing negative integers. The tracker should be validated for accurate counts of array accesses and comparisons.

Overall Evaluation

The code demonstrates strong correctness, readability, and efficiency. It's well-suited for educational or benchmarking purposes.