Cross-Review Summary: Insertion Sort vs Selection Sort

This joint report provides a comparative analysis of two fundamental sorting algorithms: Insertion Sort and Selection Sort. Both algorithms operate in-place and are widely used in educational and small-scale computational contexts due to their simplicity and clarity. The comparison considers time complexity, space efficiency, implementation details, and practical behavior.

Algorithm Overview

• Insertion Sort iteratively builds a sorted portion of the array by inserting each element into its correct position relative to previously sorted elements.
• Selection Sort repeatedly selects the minimum element from the unsorted portion and swaps it with the first unsorted element.

Time Complexity

• Best Case:
  - Insertion Sort: O(n) when the array is already sorted (only one pass, no swaps).
  - Selection Sort: $O(n^2)$ even for a sorted array since it must always scan remaining elements.

• Average Case:
  - Insertion Sort: $O(n^2)$ due to nested comparisons and shifts.
  - Selection Sort: $O(n^2)$ because it always performs $(n-1) + (n-2) + ... + 1$ comparisons.

• Worst Case:
  - Insertion Sort: $O(n^2)$, when the array is sorted in reverse order.
  - Selection Sort: $O(n^2)$, independent of initial order.

Memory Usage and Performance Characteristics

• Both algorithms are in-place sorts with O(1) additional space complexity.
• Insertion Sort typically performs fewer swaps than Selection Sort, making it more efficient on nearly-sorted or small datasets.
• Selection Sort minimizes the number of swaps (at most $n-1$) but performs a fixed number of comparisons.
• In terms of cache behavior, Insertion Sort has better locality since it works with contiguous array segments, while Selection Sort scans the entire unsorted portion on each iteration.

Implementation and Metric Tracking

• Insertion Sort uses a PerformanceTracker object to count array accesses, comparisons, and movements, providing precise runtime and operation data.
• Selection Sort employs a Metrics class that additionally tracks memory allocations, swaps, and total execution time.
• The Insertion Sort implementation emphasizes simplicity and stability, while the Selection Sort implementation includes an early termination check to detect already-sorted arrays.

Conclusion

Both algorithms are straightforward, educational examples of quadratic-time sorting. Insertion Sort outperforms Selection Sort on nearly sorted inputs due to its adaptive nature, while Selection Sort provides more predictable performance across all cases. Neither is suitable for large datasets, but both offer valuable insights into basic sorting mechanics and algorithmic efficiency measurement.