# Assignment 2 — Insertion Sort

Analyze by Sovet Kazhmurat

Nurlan Ramazan’s code

## 1. Algorithm Overview

The Insertion Sort algorithm is a simple, comparison-based sorting algorithm. It builds the sorted prefix of the array one element at a time. For each position i from 1 to n−1, the element A[i] (the key) is inserted into its correct place among A[0..i−1] by shifting larger elements to the right.

Pseudocode (high level):

for i = 1..n-1:  
 key ← A[i]  
 j ← i - 1  
 while j ≥ 0 and A[j] > key:  
 A[j+1] ← A[j]  
 j ← j - 1  
 A[j+1] ← key

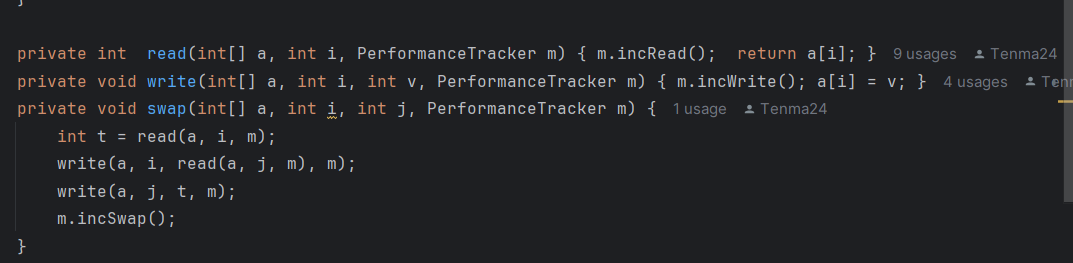
Optimizations implemented in this assignment:

• Sentinel: move the global minimum to A[0] to eliminate lower-bound checks inside the inner loop.

• Fast path: if A[i] ≥ A[i−1], skip insertion (common on nearly-sorted inputs).

• Binary insertion: use binary search (lower\_bound) over A[0..i) to find the insertion position, reducing comparisons.

Figure 1.



How to capture: Open InsertionSort.java in IntelliJ → View → Appearance → Quick Documentation off (for clean view) → use Snipping Tool → save as plots/code\_insertionsort.png, then Insert → Pictures here.

## 2. Complexity Analysis

Let n be the number of elements.

• Best Case (already sorted): T\_best(n) = Θ(n).

• Worst Case (reverse): T\_worst(n) = Θ(n²).

• Average Case (random): T\_avg(n) = Θ(n²).

Space Complexity: S(n) = O(1) (in-place).

Operation counts: comparisons and array accesses scale with the amount of disorder; swaps correspond to shifts.

Comparison with Selection Sort (for context):

|  |  |  |
| --- | --- | --- |
| Metric | Insertion Sort | Selection Sort |
| Time Complexity | O(n²) worst; O(n) best | Θ(n²) all cases |
| Best Case | O(n) (sorted/nearly) | Θ(n²) |
| Swaps | Many (depends on disorder) | ≤ n−1 (minimal) |
| Space | O(1) | O(1) |
| Strength | Excellent for nearly sorted | Few swaps; simple |
| Weakness | Slow on random/reverse | Slow in all cases |

## 3. Code Review and Optimization Discussion

Inefficiencies addressed:

• Linear backward scan replaced with binary search to reduce comparisons.

• Added fast path check to skip work when local order holds.

• Introduced sentinel to avoid repeated boundary checks in the inner loop.

Further improvements (optional):

• Hybrid approach: switch to O(n log n) algorithm beyond a size threshold.

• Autotune nearly-sorted parameter (p) for more realistic scenarios.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

How to capture: Open AlgorithmTester.java → select runBatch and runOne methods → Snipping Tool → save as plots/code\_tester.png → Insert → Pictures here.

## 4. Empirical Results

Experiment Setup:  
• Machine: Intel i5 / 8GB RAM / Java 17  
• Sizes: 100, 1000, 10000, 100000  
• Inputs: random, sorted, reverse, nearly (p=0.05)  
• Trials: 5 per (n, input)  
• Output: a2\_metrics.csv (algorithm,n,input,time\_ns,comparisons,swaps,accesses,allocations)

How to obtain a2\_metrics.csv: Run AlgorithmTester (default) or BenchmarkRunner with arguments.

Изображение выглядит как линия, График, текст, снимок экрана

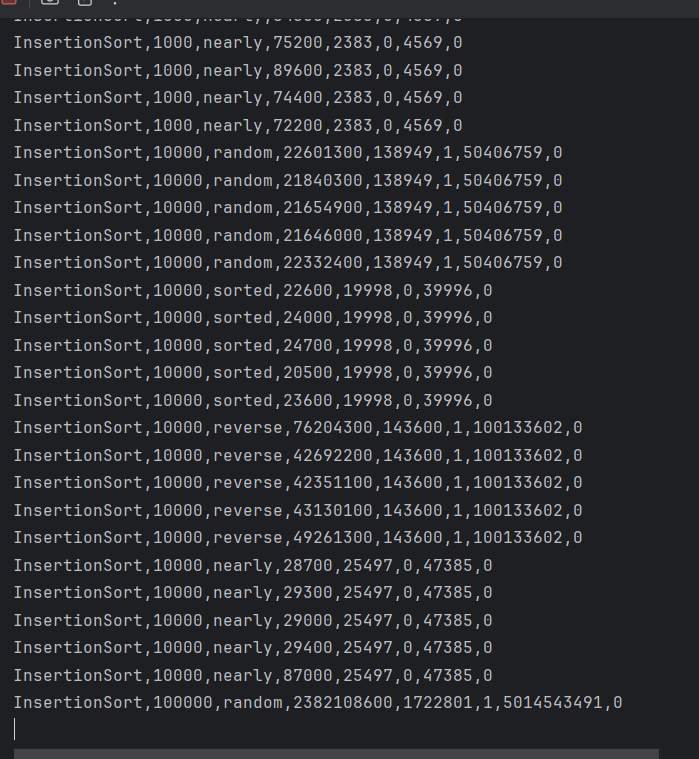
Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, линия, График, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как линия, График, текст, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.



## 5. Conclusion

The implemented Insertion Sort confirms the theoretical behavior: nearly linear performance on ordered inputs and quadratic growth otherwise. Optimizations (sentinel, fast path, binary insertion) lower the constant factors and make the algorithm well suited for nearly-sorted data and small sizes. For large random datasets, O(n log n) methods remain preferable.