## 1. Overview

Both algorithms are comparison-based sorting methods but differ significantly in design and performance characteristics.

- **Insertion Sort** builds a sorted sequence incrementally, inserting each new element into its correct position. It is simple, stable, and efficient for small or nearly-sorted datasets.
- **Heap Sort** constructs a binary heap structure and repeatedly extracts the maximum element to build the sorted array. It is in-place, non-stable, and guarantees **O(n log n)** performance regardless of input order.

---

## 2. Theoretical Complexity Comparison

| Case | Insertion Sort | Heap Sort |
|---|---|---|
| **Best Case (Ω)** | $\Omega(n)$ — already sorted | $\Omega(n \log n)$ — fixed heap operations |
| **Average Case (Θ)** | $\Theta(n^2)$ | $\Theta(n \log n)$ |
| **Worst Case (O)** | $O(n^2)$ — reverse order | $O(n \log n)$ |
| **Space Complexity** | $\Theta(1)$ (in-place) | $\Theta(1)$ (in-place) |
| **Stability** | Stable | Not stable |

**Observation:** Heap Sort offers consistent asymptotic efficiency, while Insertion Sort excels only on small or nearly-sorted inputs.

---

## 3. Empirical and Practical Insights

- For small **$n \leq 1000$**, Insertion Sort often outperforms Heap Sort due to lower constant factors and minimal overhead.
- For larger datasets, Heap Sort becomes significantly faster because it avoids quadratic growth.
- Insertion Sort’s stability makes it preferable when equal elements must preserve input order, whereas Heap Sort prioritizes raw performance.
- Both implementations are in-place and memory-efficient, fulfilling assignment requirements.

---

## 4. Code Quality and Implementation Review

### Insertion Sort

- Clean and modular structure with clear optimization for nearly-sorted arrays.
- Accurate metric tracking via *PerformanceTracker*.
- No redundant computations; excellent readability.

### Heap Sort

- Efficient use of bottom-up heapify; correct recursive structure.
- Metrics integrated consistently with algorithm steps.
- Code is concise, in-place, and aligns perfectly with algorithmic theory.

Both implementations demonstrate high code quality, readability, and proper metrics instrumentation.

---

## 5. Conclusion

Insertion Sort and Heap Sort illustrate the trade-off between **simplicity and scalability**.

- **Insertion Sort** is ideal for small or partially ordered inputs where stability matters.
- **Heap Sort** is superior for large, arbitrary datasets requiring guaranteed **O(n log n)** time.

Both implementations meet all project objectives, adhere to clean-code standards, and accurately represent their theoretical complexity.

**Overall evaluation:** Excellent algorithmic correctness, solid metric design, and complete adherence to assignment requirements.