**Readme**

This project focuses on the implementation and analysis of the HeapSort algorithm.

The main goal was to study its performance, verify theoretical complexity, and validate the results through empirical measurements.

HeapSort is a comparison-based, in-place sorting algorithm that transforms an array into a binary heap and repeatedly extracts the maximum element.

It maintains a consistent **O(n log n)** time complexity across all cases — best, average, and worst — while requiring only constant extra space.

This makes it one of the most stable and predictable sorting algorithms in practice.

Performance tracking was implemented using the PerformanceTracker class, which records comparisons, swaps, array accesses, and memory allocations.

A simple command-line tool (BenchmarkRunner) allows running tests with configurable input sizes, random seeds, and repeated executions for accurate benchmarking.

Empirical results aligned with theoretical expectations.

Execution time grew proportionally to **n log n**, confirming the algorithm’s expected complexity.

Even for arrays with 100,000 elements, HeapSort completed within around **12 ms**, demonstrating strong scalability and efficiency.

The project follows good software engineering practices — modular design, unit testing, and clear Git history with structured commits for metrics, algorithms, and CLI features.

Compared to ShellSort (the partner’s algorithm), HeapSort performed more consistently on large datasets, though ShellSort was faster for smaller, nearly sorted arrays.

Overall, this assignment successfully met the learning objectives: implementing an efficient sorting algorithm, performing asymptotic analysis, and confirming theoretical results with empirical data.