Assignment 2 — Heap Sort Analysis Report

Student: Yernur Syrlibayev
SE - 2439
Date: October 2025

Algorithm Overview

In this project I implemented the **Heap Sort** algorithm using Java.
Heap Sort works by first turning the array into a max heap — a structure where every parent is bigger than its children.
After that, the biggest element is moved to the end, and the heap is rebuilt for the rest of the array.
This repeats until everything is sorted.

I used the **bottom-up heapify** method which builds the heap faster and avoids extra recursive calls.
The algorithm sorts everything **in place**, meaning it doesn't use any extra memory besides a few variables.

What I Did

I created a Maven project with three main parts:

1. **HeapSort.java** — the algorithm itself
2. **PerformanceTracker.java** — counts comparisons, swaps, and time
3. **BenchmarkRunner.java** — runs the algorithm on different array sizes and saves results into CSV files

I also added unit tests to make sure sorting works correctly on different cases like empty arrays, single elements, duplicates, and random numbers.

After that, I ran benchmarks with arrays of 100, 1,000, 10,000, and 100,000 elements.
Each test used four types of data: random, sorted, reversed, and nearly sorted arrays.
The program saved all results into the folder `docs/performance-plots/` as `.csv` files.

What I Saw

The algorithm worked correctly for all cases.
As the array got bigger, the running time increased steadily — roughly how we expect from an **O(n log n)** algorithm.
It didn't matter much if the data was random or reversed; Heap Sort handled all of them consistently.
Sorted and nearly sorted data were just a bit faster, but not by much.

Heap Sort doesn't use extra memory and it's very stable in terms of performance.
It's not the fastest for small arrays, but it's predictable and reliable even for big ones.

What I Understood

From this task I understood how Heap Sort actually works inside —
how the heap is built and how the maximum element is moved step by step.
I learned how to measure real performance, not just theory,
and how to save data into CSV and read it later for analysis.

I also understood the importance of writing clean, modular code:
one class for the algorithm, one for tracking performance, one for running tests.
That makes debugging and future improvements much easier.

Conclusion

Heap Sort turned out to be a good mix of speed and simplicity.
It always gives solid performance, doesn't need extra memory, and is easy to test.
This project helped me connect the theory of time complexity with actual measurements in code.
Now I can clearly see how algorithm design affects real-world performance.