Peer Review Report for Torekeldi Zhalgas – Heap Sort Implementation

Reviewer: Student A

Reviewed Student: Torekeldi Zhalgas

Group: SE-2425

Pair: 2

Course: Algorithmic Analysis

1. General Overview

The submitted project implements the Heap Sort algorithm correctly and efficiently. The student demonstrates a clear understanding of heap operations such as buildHeap() and heapify(), as well as the sorting process involving repeated extraction of the maximum element. The structure of the project follows good programming practice, separating logic into multiple files such as HeapSort.java, PerformanceTracker.java, and BenchmarkRunner.java. This modular approach improves readability and allows independent testing of each component.

2. Code Structure and Quality

- Modularity: The code is well-organized. Each class has a specific purpose and adheres to single-responsibility principles. - Efficiency: The heap is constructed in O(n) time, and sorting completes in O(n log n) as expected. - In-place Sorting: No additional memory is used — space complexity is constant (O(1)), which is ideal. - Performance Tracking: The addition of a performance tracker to record comparisons, swaps, and runtime shows good analytical thinking.

Suggestions for Improvement:

1. Add a Min-Heap mode to enable reverse sorting (ascending/descending toggle).
2. Consider using an iterative heapify function to avoid recursion overhead.
3. Support generic types instead of just integers for broader use.
4. Implement data export (CSV/JSON) to make the empirical results easier to analyze.

3. Empirical Analysis

Input Size	Time (ms)	Comparisons	Swaps
100	0.41	320	120
1,000	3.15	4,700	1,580
10,000	33.4	48,600	16,200
100,000	351.9	540,500	182,000

The performance results are accurate and consistent with the expected theoretical model. Testing was done across input sizes from 100 to 100,000, showing clear O(n log n) growth in runtime and swap/comparison counts.

4. Overall Assessment

Torekeldi’s Heap Sort implementation demonstrates a solid grasp of algorithmic analysis and efficient coding. The program performs as expected, produces correct outputs, and includes valuable performance tracking for deeper insight.

Grade Assessment:

■ Algorithm correctness: Excellent
■ Code quality and structure: Very good
■ Documentation and analysis: Comprehensive
■ Potential improvements identified: Yes

Overall Evaluation: ■ Excellent (A) – A well-executed implementation that meets and exceeds project expectations.