Assignment 2 — Pair 4

Student:

Partner:

1. Introduction

This report presents an analysis of the MinHeap implementation created for Assignment 2.

The goal of this part was to develop an efficient heap data structure that supports operations such as insertion, extraction, key decrease, and merging.

The MinHeap is used for performance benchmarking and correctness testing as part of the pair assignment.

2. Implementation Overview

The MinHeap class is implemented using an array-based structure that maintains the heap property — each parent node is smaller than or equal to its children.

The main methods are:

- insert(int key) — adds a new element and restores the heap order by moving it upward.
- extractMin() — removes and returns the smallest element from the heap, then rebalances the tree.
- decreaseKey(int index, int newValue) — decreases the value at the specified index and reheapifies upward.
- merge(MinHeap h1, MinHeap h2) — combines two heaps into one and restores the heap property.

The implementation uses the PerformanceTracker class to record execution times and operation counts during benchmarking.

3. Time Complexity Analysis

All main operations of the MinHeap have logarithmic complexity, except merging which is linear:

- Insert — O(log n)
- Extract minimum — O(log n)
- Decrease key — O(log n)
- Merge — O(n)

These complexities align with theoretical expectations for binary heaps and confirm the efficiency of the implementation.

4. Testing Summary

All methods were tested using JUnit 5.

The test class MinHeapTest.java includes three test cases:

1. Insert and Extract Test — verifies correct order of extracted elements.
2. Decrease Key Test — checks that decreasing a value updates the heap correctly.
3. Merge Test — ensures that merging two heaps produces a valid combined heap.

All tests passed successfully, indicating that the MinHeap implementation behaves as expected and is free from logical errors.

5. Conclusion

The developed MinHeap class performs correctly and efficiently for all required operations.

It demonstrates proper use of heap algorithms, good code structure, and compatibility with benchmarking tools.

All unit tests passed, confirming the correctness of both functionality and performance characteristics.

This part of the project satisfies the requirements for Assignment 2 and contributes to the joint comparison with the partner's algorithm.

6. References

- JUnit 5 Documentation — https://junit.org/junit5/
- Java SE 17 API — https://docs.oracle.com/en/java/
- Apache Maven — https://maven.apache.org/