Assignment 2 — Heap Data Structures

Author: Yerkebulan Sovet

Partner: Ernar Sadenov

Date: 05.10.25

1. Introduction

This report analyzes the implementation of the MaxHeap algorithm developed by Ernar Sadenov. MaxHeap is a binary heap data structure that maintains the largest element at the root. It supports efficient insertion, extraction of the maximum element, and key updates.

2. Algorithm Description

The MaxHeap is implemented using an array-based representation. Key properties:

- Each node is greater than or equal to its children.
- The root contains the maximum element.
- Tree is complete (all levels filled, except possibly the last).

Operations:

- Insert: Add an element at the end, then "heapify up."
- Extract Max: Remove the root, replace with the last element, then "heapify down."
- Increase Key: Update a key and move upwards if necessary.
- Merge: Combine two heaps and rebuild in $\Theta(n + m)$.

3. Theoretical Analysis

- Insertion: $O(\log n)$

- Extract Max: O(log n)

- Increase Key: O(log n)

- Build Heap: Θ(n)

- Merge: O(n + m)

Space complexity: O(n).

Theoretical time complexity matches classical heap performance.

4. Code Review

Strengths:

- Clear structure with helper methods.

- Checks for errors (e.g., empty extract).

- Merge supported via array concatenation + buildHeap.

Improvements:

- Use generics for flexibility.

- Add more boundary-case tests.

- Include more detailed comments.

5. Empirical Results

Benchmarks were conducted with random inputs of sizes 100, 1,000, 10,000, and 100,000. Results show MaxHeap scales as expected with O(log n). Performance is almost identical to MinHeap, confirming theoretical predictions.

6. Conclusion

The MaxHeap implementation is correct and efficient. It behaves symmetrically to MinHeap, with differences only in root element selection. Future improvements may include generics and extended test coverage.