Analysis Report — Heap Data Structures

Course: Design and Analysis of Algorithms (DAA)
Pair: 4 — Heap Data Structures
Student: Bakytzhan Kassymgali
Partner: Alikhan Serik
Analyzed Part: *Min-Heap Implementation (decrease-key, merge)*

1. Introduction

This report presents a detailed analysis of my partner's implementation of the **Min-Heap data structure**, developed as part of the "Heap Data Structures" assignment.
The purpose of the analysis is to evaluate the algorithmic design, efficiency, and correctness of the operations, and to compare its performance with the **Max-Heap** implementation.

A **Min-Heap** is a complete binary tree where each parent node is smaller than or equal to its children.
The smallest element is always located at the root.
This structure efficiently supports key operations such as **insert**, **extractMin**, **decreaseKey**, and **merge**.

2. Algorithm Design and Implementation

My partner's implementation of the Min-Heap follows the standard array-based heap model using a 0-indexed list.
Below is a summary of the major operations and their behavior:

Operation	Description	Complexity
insert(key)	Adds a new key at the end and performs *heapify-up* to maintain order.	O(log n)
extractMin()	Removes the root (minimum element), replaces it with the last node, then performs *heapify-down*.	O(log n)
decreaseKey(index, newKey)	Decreases the value of a key and restores heap order via *heapify-up*.	O(log n)
merge(otherHeap)	Combines two heaps into one by inserting all elements or merging arrays.	O(n)

The code demonstrates a clean and modular design:

- Private helper methods such as `heapifyUp()` and `heapifyDown()` improve readability.
- Proper encapsulation and input validation prevent runtime errors.
- Naming conventions follow **Java Clean Code principles**, making the logic easy to follow.

3. Experimental Results

Performance benchmarking was performed using the BenchmarkRunner and PerformanceTracker classes.
Below are the results from the partner's **minheap_results.csv** file:

n	Comparisons	Swaps	Array Accesses	Allocations	Time (ns)
100	940	412	4300	3	95,000
1,000	16,500	7,900	71,200	6	2,950,000
10,000	235,000	118,000	987,000	10	12,600,000

The results indicate that as input size **n** increases, execution time grows approximately as **O(n log n)**, which is consistent with the theoretical complexity of heap operations.
The time increase is smooth and predictable, suggesting no performance bottlenecks or redundant computations.

4. Comparison with Max-Heap

Operation	Max-Heap (my part)	Min-Heap (partner)	Comment
insert	O(log n)	O(log n)	Identical logic (heapify-up)
extract	extractMax	extractMin	Symmetrical structure
key modification	increaseKey	decreaseKey	Mirror implementations
merge	✔	✔	Both linear in O(n)

Observation:
The Max-Heap and Min-Heap exhibit nearly identical performance trends.
Any minor timing differences are due to variation in comparison direction (> vs <) and system-level memory handling.
Overall, both implementations are efficient and complement each other as a pair.

5. Code Quality and Design Evaluation

Strengths:

- Clear separation of concerns and reusable helper functions.
- Correct handling of edge cases (empty heap, invalid indices).
- Good adherence to naming and formatting conventions.

Suggestions for Improvement:

- Add more unit tests for merge() and decreaseKey() to cover corner cases.
- Include inline comments explaining the recursive or iterative heapify process.

Overall, the code is well-structured, logically sound, and demonstrates a strong understanding of heap properties and algorithmic design.

6. Conclusion

The analysis confirms that the **Min-Heap implementation** by *Alikhan Serik* is correct, efficient, and cleanly written.
The experimental results align with theoretical expectations for logarithmic performance.
The project successfully demonstrates the relationship between Max-Heap and Min-Heap structures and validates their efficiency through empirical measurement.

Both implementations reflect the core principles of algorithm design and performance analysis covered in the DAA course.

Prepared by:
Bakytzhan Kassymgali
Design and Analysis of Algorithms — Assignment 2