Astana IT University
Course: Algorithms and Data Structures
Assignment 2 – Algorithm Analysis Report

Reviewer (Student A): Aslan Muratov
Reviewed Student (Student B): Ayadil Kozhabek
Group: SE-2425

1. Algorithm Overview

The implementation provided by Student B (Ayadil Kozhabek) presents a MaxHeap data structure. It includes the following main operations:

`insert(key)`: adds a new element while maintaining the heap property.

`extractMax()`: removes and returns the maximum element, then restores the heap property.

`getMax()`: returns the maximum element without removing it.

`increaseKey(i, newKey)`: increases the value at a given index and moves the node upward if needed.

The heap is implemented using an array starting from index 0.
The code also integrates a PerformanceTracker class that counts comparisons, swaps, and array accesses during execution, which is a valuable addition for performance measurement.

2. Complexity Analysis

Operation	Best Case	Average Case	Worst Case
Insert	Ω(1)	Θ(log n)	O(log n)
Extract Max	-	-	O(log n)
Increase Key	-	-	O(log n)
Space Complexity	-	-	O(n)

3. Code Review

Strengths:

The project structure is clean and modular (packages: algorithms, metrics, cli).

Code readability is high; variable and method names are descriptive.

Integration of performance measurement is a good practice for algorithmic comparison.

The benchmark runner allows easy testing with different input sizes.

Areas for Improvement:

The heap could include a check for underflow when performing `extractMax()` on an empty heap.

It would be helpful to implement automatic resizing when the heap becomes full.

Inline comments could be added in complex parts like `heapify()` and `increaseKey()` to improve clarity.

Benchmark results could automatically be saved into a `/docs/performance-plots/` folder for better organization.

Overall, the implementation is correct, efficient, and follows good software engineering practices.

4. Empirical Results

Size (n)	Comparisons	Swaps	Array Accesses
100	714	542	100
1,000	11,954	8,573	1,000
5,000	77,385	54,829	5,000
10,000	170,024	119,639	10,000

The results show that the algorithm scales approximately as **O(n log n)**.
The ratio between comparisons and swaps remains consistent, confirming the stability of the implementation.

5. Conclusion

The **MaxHeap** implementation by **Ayadil Kozhabek** is well-designed and performs efficiently.
All core heap operations work correctly, and performance analysis is properly integrated.

Strengths: clean structure, correct algorithms, and benchmark integration.
Weaknesses: minor issues with heap resizing and empty-heap validation.

Final Assessment: Excellent work with only small areas for potential improvement.