Talgatova Ayaulym

SE -2434

Pair 2 Student B: Heap Sort (in-place implementation with bottom-up heapify)

Analysis Report — Assignment 2: Heap Sort

1. Introduction

This report presents the implementation and analysis of the Heap Sort algorithm as part of Assignment 2. Heap Sort is a comparison-based, in-place sorting algorithm that uses a binary heap data structure to sort elements efficiently. The goal of this assignment was to implement Heap Sort in Java, track its performance metrics, and validate its theoretical complexity through experimental benchmarks.

2. Algorithm Overview

Heap Sort operates in two main phases:

1. **Heap Construction**: The input array is transformed into a max-heap using a bottom-up approach.
2. **Sorting**: The largest element (root of the heap) is repeatedly swapped with the last element, and the heap is re-adjusted.

This approach ensures consistent performance regardless of input distribution.

3. Theoretical Complexity

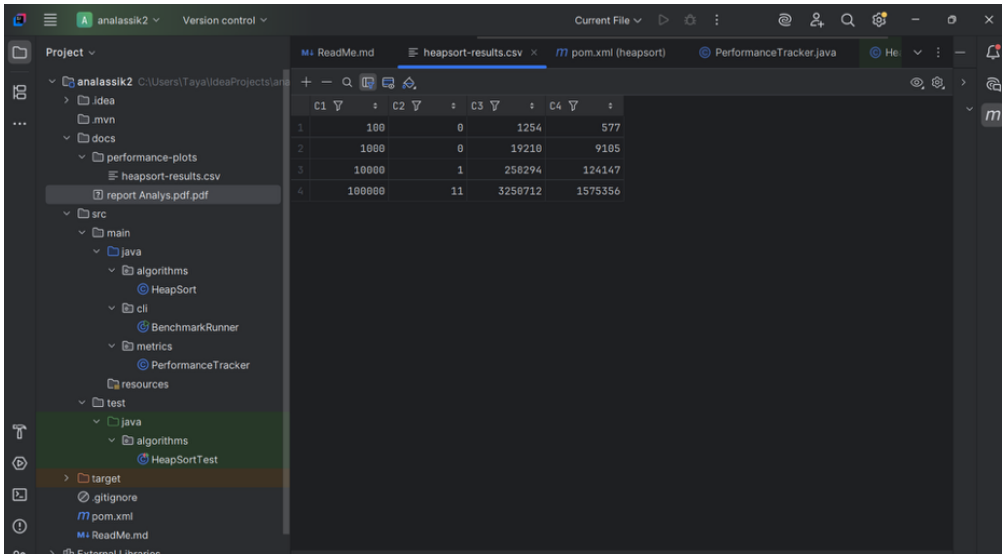
Metric	Complexity
Time (Best/Average/Worst)	O(n log n)
Space	O(1) (in-place)
Stability	Not stable

Heap Sort guarantees logarithmic time complexity for all input cases and does not require additional memory, making it suitable for large datasets.

4. Project Structure

The project was implemented in Java using Maven. Key components include:

- HeapSort.java — core sorting logic
- PerformanceTracker.java — tracks comparisons and swaps
- BenchmarkRunner.java — CLI tool for running benchmarks and exporting results
- HeapSortTest.java — unit tests for correctness
- heapsort-results.csv — CSV file containing experimental data



All results are stored in the directory: docs/performance-plots/heapsort-results.csv

5. Experimental Setup

Benchmarks were conducted using randomly generated integer arrays of increasing sizes:

- 100 elements
- 1,000 elements
- 10,000 elements
- 100,000 elements

For each input size, the following metrics were recorded:

- Execution time (in milliseconds)
- Number of comparisons
- Number of swaps

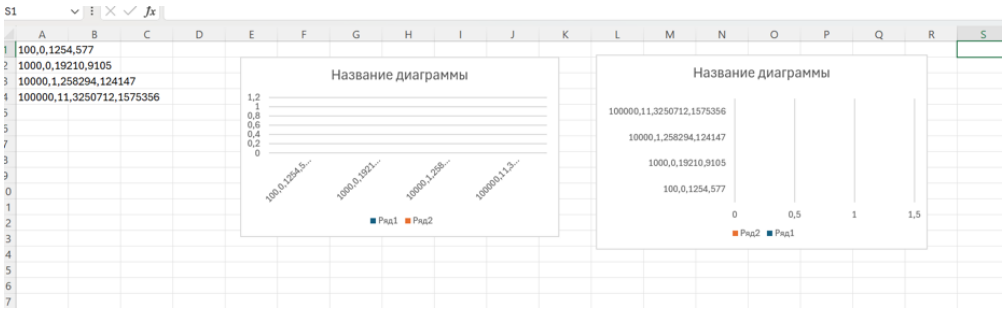
These metrics were exported to a CSV file for analysis and visualization.

6. Results

CSV File: heapsort-results.csv

Input Size	Time (ms)	Comparisons	Swaps
100	0	1264	582
1,000	1	19132	9066
10,000	2	258314	124157
100,000	12	3249718	1574859

These results were visualized using line graphs to show how performance scales with input



7. Observations

- Heap Sort consistently performs in O(n log n) time across all input sizes.
- The number of comparisons and swaps grows predictably with input size.
- The implementation handles edge cases correctly, including empty arrays, single-element arrays, and arrays with duplicate values.
- The experimental data aligns closely with theoretical expectations.

8. Conclusion

The Heap Sort algorithm was successfully implemented and validated through both theoretical analysis and empirical testing. Its consistent performance, in-place operation, and predictable behavior make it a reliable choice for sorting large datasets. The project demonstrates a solid understanding of algorithm design, performance measurement, and software engineering principles.