Assignment 2: Algorithmic Analysis and Peer Code Review

Student: Kuanysh Asaubaev
Group: SE-2438
Pair 4 — Student B (Heap Sort Implementation)

1. Algorithm Overview

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure.
It works in two main phases:
1. buildMaxHeap: constructs a max-heap from an unsorted array.
2. heapSort: repeatedly extracts the largest element (root), places it at the end, and restores the heap property.

Advantages:
- In-place algorithm (requires no extra memory).
- Guaranteed time complexity of O(n log n).
- Predictable performance for large datasets.

Applications:
- Task scheduling, priority queues, and large-scale data sorting.

2. Complexity Analysis

Operation	Best (Ω)	Average (Θ)	Worst (O)	Space
buildHeap	Ω(n)	Θ(n)	O(n)	O(1)
heapify	Ω(1)	Θ(log n)	O(log n)	O(1)
heapSort	Ω(n log n)	Θ(n log n)	O(n log n)	O(1)

Heap Sort achieves O(n log n) performance due to repeated heapify operations.
Building the heap takes O(n), and each extraction requires O(log n), resulting in overall O(n log n) time.

3. Code Review and Optimization

Reviewed peer implementation: Shell Sort (Student A).

Strengths:
- Modular structure and clear method design.
- Efficient implementation of different gap sequences.

Areas for Improvement:

- Missing performance metrics for comparisons and swaps.
- Could improve memory efficiency by reusing buffers.
- Benchmark runner can be extended to handle multiple datasets.

Suggested Optimizations:
- Use PerformanceTracker for empirical measurements.
- Add CLI argument parsing for easier benchmarking.
- Reduce redundant comparisons inside nested loops.

4. Empirical Results

Performance tests were conducted for input sizes n = 1,000, 5,000, 10,000, 50,000, and 100,000.
Each test was repeated three times, measuring average runtime in milliseconds.

n	time_ms
1,000	0.717
5,000	0.554
10,000	1.209
50,000	5.179
100,000	8.834

The results confirm near O(n log n) growth. Heap Sort shows stable performance and scalability
with increasing input size.

5. Conclusion

Heap Sort was successfully implemented and analyzed both theoretically and empirically.
The results matched the expected time complexity of O(n log n).

The algorithm proved to be efficient, reliable, and suitable for large-scale applications.
It meets all assignment requirements in terms of performance, correctness, and documentation.