Algorithm Analysis Report: Min-Heap Implementation

1. Algorithm Overview

1.1 Theoretical Background
Min-Heap is a complete binary tree data structure where each parent node is less than or equal to its children. This property ensures the minimum element is always at the root.

1.2 Key Operations
- Insert: Add element and maintain heap property - O(log n)
- ExtractMin: Remove minimum element and restore heap - O(log n)
- DecreaseKey: Decrease value and bubble up - O(log n)
- Merge: Combine two heaps - O(n log n)

2. Complexity Analysis

2.1 Time Complexity

Operation	Best Case	Average Case	Worst Case
Insert	O(1)	O(log n)	O(log n)
ExtractMin	O(log n)	O(log n)	O(log n)
DecreaseKey	O(1)	O(log n)	O(log n)
Merge	O(n)	O(n log n)	O(n log n)

2.2 Space Complexity
- Auxiliary Space: O(1) for in-place operations
- Total Space: O(n) for storing n elements

3. Empirical Results

3.1 Performance Measurements

Testing with size: 100
Insert 100 elements: 0 ms, comparisons: 197, swaps: 102
Extract all: 0 ms, comparisons: 1028, swaps: 415

Testing with size: 1000
Insert 1000 elements: 0 ms, comparisons: 2205, swaps: 1212
Extract all: 1 ms, comparisons: 16682, swaps: 7342

Testing with size: 10000
Insert 10000 elements: 1 ms, comparisons: 22655, swaps: 12662
Extract all: 6 ms, comparisons: 233526, swaps: 106764

Testing with size: 100000
Insert 100000 elements: 13 ms, comparisons: 227243, swaps: 127250
Extract all: 50 ms, comparisons: 2999412, swaps: 1399707
=== Performance Metrics ===
extract_100000: avg time = 50.000 ms (samples: 1)
insert_100: avg time = 0.000 ms (samples: 1)
extract_10000: avg time = 6.000 ms (samples: 1)
extract_1000: avg time = 1.000 ms (samples: 1)
insert_10000: avg time = 1.000 ms (samples: 1)
insert_100000: avg time = 13.000 ms (samples: 1)
extract_100: avg time = 0.000 ms (samples: 1)
insert_1000: avg time = 0.000 ms (samples: 1)
extract_100_swaps: avg = 415.0 operations
extract_100000_swaps: avg = 1399707.0 operations
insert_100_comparisons: avg = 197.0 operations
extract_1000_swaps: avg = 7342.0 operations
extract_100_comparisons: avg = 1028.0 operations
insert_1000_comparisons: avg = 2205.0 operations
extract_10000_comparisons: avg = 233526.0 operations
insert_10000_swaps: avg = 12662.0 operations
extract_1000_comparisons: avg = 16682.0 operations
insert_100000_comparisons: avg = 227243.0 operations
insert_100000_swaps: avg = 127250.0 operations
extract_100000_comparisons: avg = 2999412.0 operations
extract_10000_swaps: avg = 106764.0 operations
insert_1000_swaps: avg = 1212.0 operations
insert_10000_comparisons: avg = 22655.0 operations
insert_100_swaps: avg = 102.0 operations

3.2 Complexity Verification
- Theoretical: O(log n) per operation
- Empirical: Time growth follows logarithmic pattern
- Validation: Plot of time vs n confirms O(log n) behavior

3.3 Results Analysis
The empirical results confirm theoretical expectations - time complexity grows logarithmically with input size. For 100,000 elements, insert operations take 13ms while extract operations take 50ms, demonstrating efficient O(log n) performance.

4. Comparison with Partner's Algorithm

4.1 Max-Heap vs Min-Heap
- Similarities: Same time/space complexity, same core operations
- Differences: Ordering property (max vs min), application use cases

4.2 Performance Comparison
- Both show identical asymptotic behavior
- Minor differences in constant factors

5. Conclusion

5.1 Summary of Findings
- Implementation correctly maintains O(log n) time complexity
- Space usage efficient with O(n) total memory
- Performance scales well with input size
- Code quality high with good optimization potential

Student: Pair 4, Student A
Date: 05.10.25
Course: Design and Analysis of Algorithms