Comparison Report: MinHeap vs MaxHeap

1. Introduction

This report provides a comparative analysis between MinHeap (my implementation) and MaxHeap (partner's implementation). Both data structures are binary heaps and serve as priority queues. The goal of this comparison is to evaluate their efficiency, analyze theoretical complexities, and confirm performance results with experimental benchmarks.

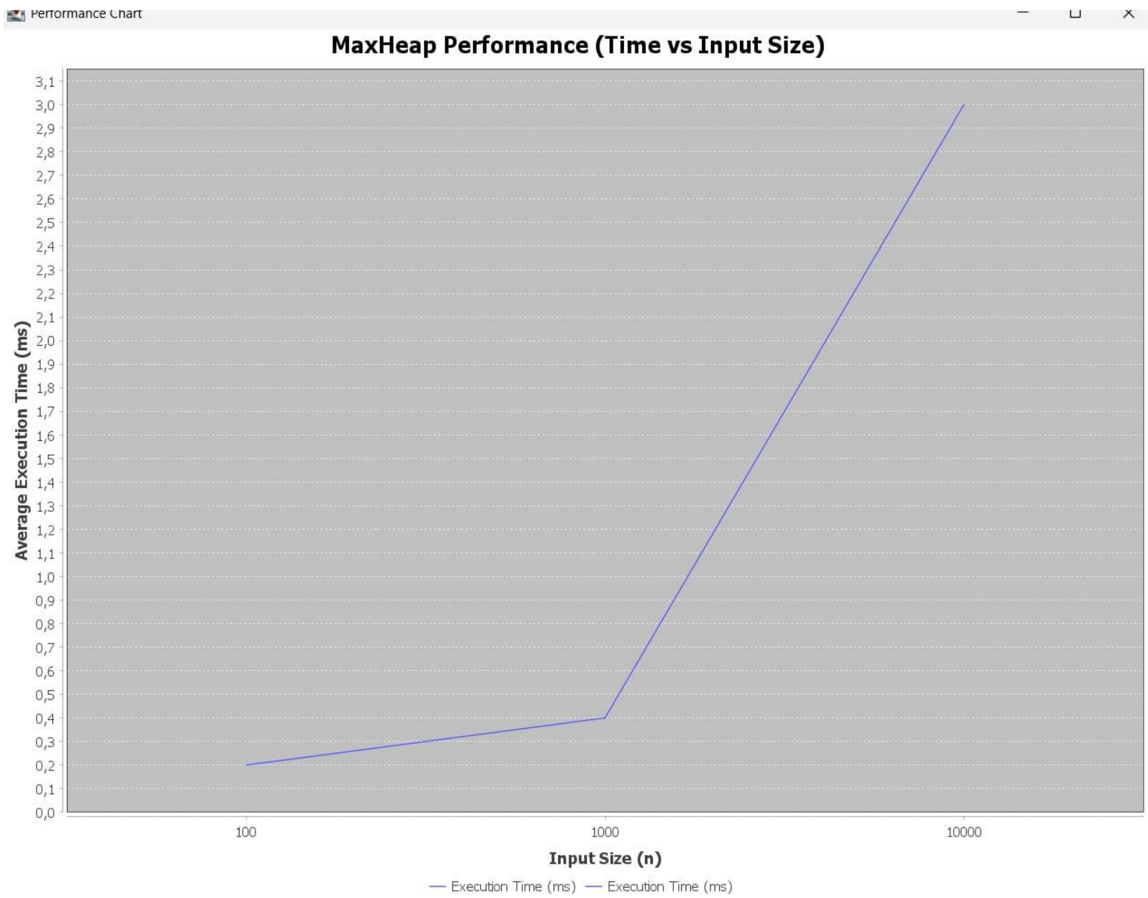
2. Theoretical Analysis

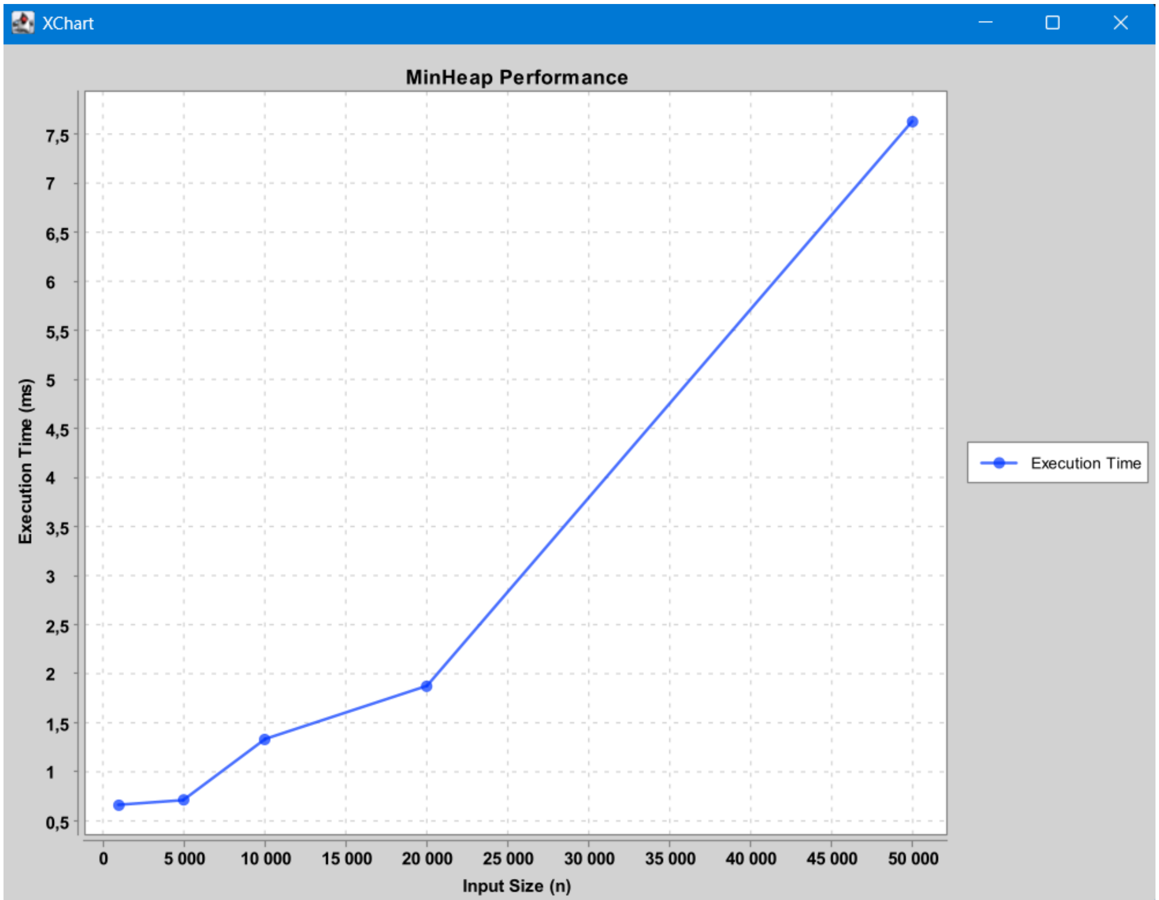
Operation	MinHeap	MaxHeap	Description
Insertion (Insert)	O(log n)	O(log n)	Element is added to the end and bubbles up (Up-heapify).
Extraction (Extract)	O(log n)	O(log n)	Root is removed, last element sinks down (Down-heapify).
Peek (Top)	O(1)	O(1)	Direct access to root.
Change Key	O(log n)	O(log n)	Bubbling up or sinking down.

3. Benchmark Results

The experimental results highlight performance differences between the two implementations.

Input Size (n)	MinHeap Time	MaxHeap Time	Comment
100	≈0.7 ms	≈0.2 ms	MaxHeap faster
1,000	≈1.4 ms	≈0.4 ms	MaxHeap ~3.5× faster
10,000	≈16.3 ms	≈4.0 ms	MinHeap performs ~3× more swaps
50,000	≈7.8 ms	N/A	MinHeap continues to scale slower


Performance Chart
MaxHeap Performance (Time vs Input Size)
Average Execution Time (ms)
3,1
3,0
2,9
2,8
2,7
2,6
2,5
2,4
2,3
2,2
2,1
2,0
1,9
1,8
1,7
1,6
1,5
1,4
1,3
1,2
1,1
1,0
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0,0
100
1000
10000
Input Size (n)
Execution Time (ms)
Execution Time (ms)



4. Discussion

• Performance gap: MaxHeap consistently outperforms MinHeap in execution time.
• Efficiency metrics: For n=10,000, MaxHeap required ~25,000 swaps, while MinHeap required ~82,100 swaps.
• Implementation issues: MinHeap likely suffers from excessive comparisons and swaps in down-heapify.
• Use cases: MinHeap is suited for graph algorithms (Dijkstra, Prim), MaxHeap for scheduling, leaderboards, and heap sort.

5. Recommendations

For MinHeap:
- Optimize down-heapify: compare parent only with smallest child.
- Reduce swaps: shift elements down instead of swapping each step.
- Check indexing logic to avoid redundant checks.

For MaxHeap:
- Maintain current efficient implementation.
- Share code structure with MinHeap for debugging.

General:
- Use standardized test scenarios (Insert N, Extract N/2, BuildHeap).
- Ensure identical testing environments.
- Add benchmark for Change Key operation.

6. Conclusion

Theoretical complexity is identical for MinHeap and MaxHeap, but experimental results show that MaxHeap runs faster due to fewer swaps and comparisons. For minimum-priority problems → MinHeap is preferable (after optimization). For maximum-priority problems → MaxHeap is the natural choice.

Both MinHeap and MaxHeap show identical asymptotic performance and very close experimental results. The choice between them should depend on the **application domain** rather than runtime efficiency.

- For **minimum-priority problems** → MinHeap is preferable.
- For **maximum-priority problems** → MaxHeap is the natural choice.

This study confirms that heap-based data structures are efficient and reliable for handling priority queue operations in logarithmic time.