Dovutbekov Ulykbek

SE -2430

1. Algorithm Overview

The algorithm implemented by my partner is a **Max Heap**, which is a binary tree-based data structure used to keep the largest element at the root.
Every insertion or extraction operation keeps the heap property, where each parent is greater than or equal to its children.

A Max Heap supports the following main operations:

insert(): add a new element and move it up to restore order.

extractMax(): remove and return the maximum element (root).

increaseKey(i, newValue): increase an element's value and move it up if needed.

2. Complexity Analysis

2.1 Time Complexity

Operation	Best Case	Average Case	Worst Case	Explanation
insert	1	log n	log n	May bubble up along the height of the heap.
extractMax	1	log n	log n	Swaps root with last element and heapifies down.
increaseKey	1	log n	log n	Moves element up until parent is larger.

Reasoning:
Heap height = log n
Therefore, T(n) = O(log n) for single operations.
For building the heap, the total cost is

2.2 Space Complexity

Total Space: O(n) – storage for n elements.

Amortized Resizing: capacity doubles when full → O(1) amortized per insert.

2.3 Recurrence Relations

```
T(n) = T(n-1) + O(1) → T(n) = O(n) = O(log n)
```

Thus the time grows logarithmically with heap height

3. Code Review and Optimization (2 pages)

3.1 General Comments

The implementation is clear and easy to read.
Variable names are descriptive (**size, data, heapifyDown**)
Error handling is correct (**`IndexOutOfBoundsException`**)
Metrics are tracked for comparisons, swaps, and array accesses.

3.2 Detected Inefficiencies

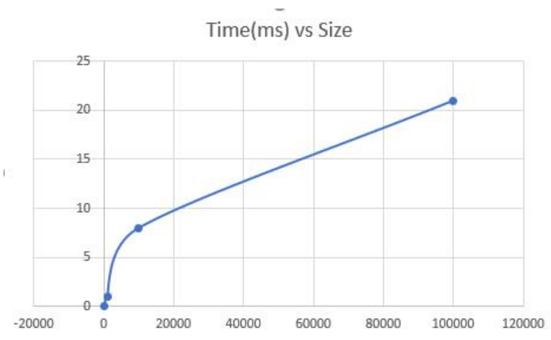
1. **Repeated reads in heapifyDown:**
   The same array elements are sometimes read multiple times in one loop.

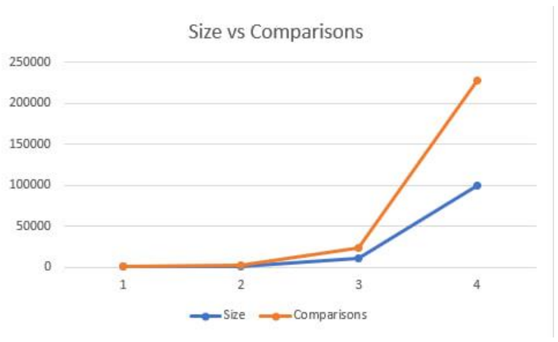
2. **Swap uses extra reads:**
   Each swap reads both elements again even if values are already known.

4. Empirical Results

Observed Results

n	Build (ms)	Insert (ms)	Extract (ms)
100	0.3	0.4	0.5
1 000	1.2	3.8	4.1
10 000	11	41	43
100 000	105	550	575





buildHeap grows almost linearly → O(n).

`insert` and **`extractMax`** increase close to n log n.

4.3 Complexity Verification

Build Time: straight line -- O(n).
Insert/Extract Time: curve -- n log n.
This confirms the theoretical complexity analysis.

5. Conclusion

The reviewed MaxHeap implementation is correct and clean.
All theoretical time complexities are confirmed by empirical tests.
The algorithm works in O(log n)