Student A – Review of Max-Heap Implementation (Student B: Aslan Muratov)

Reviewer (Student A): Ayadil Kozhabek
Reviewed Student (Student B): Aslan Muratov
Group: SE-2425
University: Astana IT University

1. Algorithm Overview

The implementation provided by Student B covers the **Max-Heap data structure** with the following key operations:

- **insert(key):** inserts a new element while maintaining heap property.
- **getMax():** retrieves the maximum element without removing it.
- **extractMax():** removes and returns the maximum element, then restores the heap property.
- **increaseKey(i, newKey):** increases the value at index i and shifts the element upward if necessary.

The heap is backed by an array, with indices starting from 0. The implementation additionally tracks **performance metrics** (comparisons, swaps, array accesses), which is a very good extension for algorithm analysis.

---

2. Complexity Analysis

- **Insert:**
  - Worst case: O(log n) (when the new element bubbles up to the root).
  - Best case: Ω(1) (when the new element is placed at the bottom without movement).
  - Average case: Θ(log n).
- **Extract Max:**
  - Always requires restoring heap property: O(log n).
- **Increase Key:**
  - Similar to insert: O(log n) in the worst case.
- **Space Complexity:** O(n).

The theoretical analysis is consistent with expectations.

---

3. Code Review

Strengths:

- Clear structure and separation into packages (algorithms, metrics, cli).
- Usage of PerformanceTracker allows empirical analysis.
- Unit tests cover typical and edge cases (insert, extract, increaseKey, empty heap).
- Benchmark runner exports CSV for further visualization.

Areas for Improvement:

- **Dynamic resizing**: currently the heap is initialized with a fixed capacity; adding auto-expansion (like ArrayList) would make it more flexible.
- **Validation**: increaseKey should explicitly handle the case when newKey < oldKey (currently assumes only increase).
- **CSV output location**: results are saved in project root; could be placed into /docs/performance-plots/ automatically.
- **Code comments**: some methods (like heapify) could use more inline documentation for clarity.

Overall, the implementation is strong and well-structured, with only minor improvements suggested.

4. Empirical Results

Size (n)	Comparisons	Swaps	Array Accesses
100	714	542	100
1000	11,954	8,573	1000
5000	77,385	54,829	5000
10000	170,024	119,639	10000

- Growth is approximately **O(n log n)**, which matches theoretical expectations.
- The ratio between comparisons and swaps remains stable as input grows, which confirms the correctness of the heap operations.
- Performance scaling is efficient and no unexpected overheads are observed.

Recommendation: visualize these results in plots (Comparisons vs n, Swaps vs n) to better illustrate complexity trends.

5. Conclusion

The Max-Heap implementation by **Aslan Muratov** is correct, efficient, and meets the assignment requirements.

- **Strengths:** solid implementation, testing, and benchmarking integration.
- **Weaknesses:** minor code maintainability issues (fixed capacity, limited validation).
- **Final Assessment:** Excellent work; only small improvements are recommended.