Report on MinHeap Class

1. Purpose

The MinHeap class implements a min-heap data structure using an array, providing the following operations:

Insert an element (insert)

Extract the minimum element (extractMin)

Decrease the value of a key (decreaseKey)

Merge two heaps (merge)

Build a heap from an array (buildHeap)

Additionally, the class tracks performance metrics via a PerformanceTracker object (counting comparisons, accesses, and swaps).

2. Class Fields

Field	Type	Purpose
heap	int[]	Array storing the heap elements
size	int	Current number of elements in the heap
tracker	PerformanceTracker	Tracks operations for performance analysis

3. Constructors

MinHeap(int capacity)
Creates an empty heap with the given capacity. If capacity < 1, a size of 1 is used.

MinHeap(int[] data, boolean build)
Creates a heap from an existing array. If build = true, the array is converted into a valid min-heap using buildHeap().

4. Main Methods

4.1 Helper Methods

parent(i) — returns the index of the parent of element i

left(i) — returns the index of the left child

right(i) — returns the index of the right child

ensureCapacity(minCapacity) — dynamically increases the array size if needed

swap(i, j) — swaps two elements, incrementing the performance counters

4.2 Core Operations

4.2.1 Insert (insert(int key))

Algorithm:

Ensure sufficient array capacity.

Place the new element at the end of the array.

Restore the min-heap property using heapifyUp.

Time complexity:

Best case: O(1)

Worst case: O(log n)

4.2.2 Extract Minimum (extractMin())

Algorithm:

Save the root element (heap[0]) to return later.

Move the last element to the root.

Decrease the heap size.

Restore the min-heap property using heapifyDown.

Time complexity: O(log n)

4.2.3 Decrease Key (decreaseKey(int i, int newValue))

Algorithm:

Check index and new value validity.

Assign the new value to the element.

Restore the heap property upwards using heapifyUp.

Time complexity: O(log n)

4.2.4 Merge Heaps (merge(MinHeap other))

Algorithm:

Create a new array containing elements from both heaps.

Build a new heap from this combined array.

Time complexity: O(n + m), where n and m are the sizes of the heaps.

5. Heap Restoration Methods

heapifyUp(int i) — moves an element up until the min-heap property is satisfied.

heapifyDown(int i) — moves an element down, choosing the smaller child at each step.

Time complexity for both: O(log n)

6. Implementation Features

Array-based storage for heap elements.

Dynamic resizing of the array (ensureCapacity).

Performance tracking with PerformanceTracker.

Heap construction from an arbitrary array (buildHeap).

Heap merging through array combination and heap rebuilding.

7. Interaction Methods

size() — returns the number of elements

toArray() — returns a copy of the heap array

toString() — returns a string representation of the heap

8. Complexity Analysis

Operation	Best Case	Worst Case
insert	O(1)	O(log n)
extractMin	O(1)	O(log n)
decreaseKey	O(1)	O(log n)
merge	O(n + m)	O(n + m)
buildHeap	O(n)	O(n)

9. Potential Improvements

Use generics to support types other than int.

Implement deletion of arbitrary elements.

Optimize merge without full copying and rebuilding.

Add input validation in MinHeap(int[] data, boolean build) (e.g., null checks).

10. Conclusion

The MinHeap class provides a correct and efficient implementation of a min-heap using arrays. It supports all essential operations with good performance and includes operation tracking for analysis. The class is suitable for algorithmic tasks requiring a priority queue.