Overview:

The MinHeap class implements a minimum heap with basic operations: insertionInHeap, deleteRoot, mergeTwoMinHeaps, decreaseKey.

The main idea:

To insert: add an element to the end of the array

To delete the root: swap the root element with the last element, reduce the size

mergeTwoMinHeaps: inserts all elements of the second heap using insertionInHeap.

decreaseKey: decreases the value of the element at position pos

The class also calculates metrics: the number of comparisons, permutations, start/end time using Metrics.startTimer() / stopTimer() and logs operations.

Complexity Analysis

- Insert: O(1)

Heapify: worst case, when a new element goes to the root of the array O(logn)

- Delete root:

on average O(1), worst case, when the root searches for the correct position, O(logn)

Average/worst case: O(logn)

Best case: if the root is already smaller than both children Ω(1)

- Decrease key:

Checking and putting O(1)

heapifyUp O(logn)

- Metrics: O(1)

Space complexity:

- A dynamic array is used, occupying O(n) memory
- Additional variables and counters O(1)

Inefficient code section

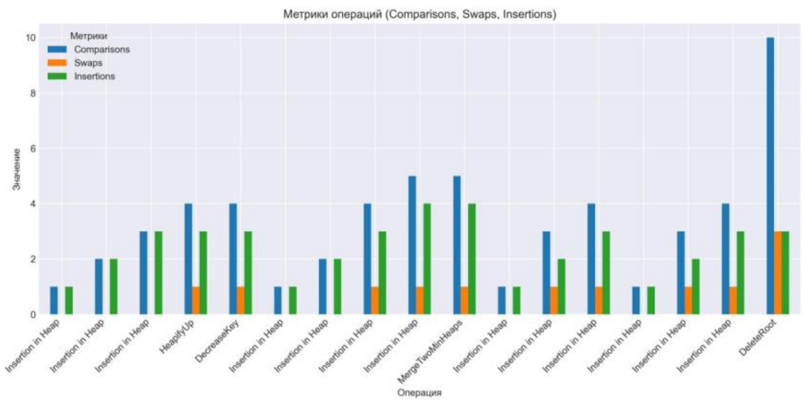
Instead of inserting elements one by one, merge both heaps into a single array and call heapify once.
This reduces time complexity from O(n log n) to O(m + n) and improves performance for large heaps.

Empirical results

To verify the theoretical assessment of the complexity of the main MinHeap operations, experiments were conducted with different input sizes.

Randomly generated data sets of various sizes

The execution time of the operations was measured using a built-in timer.



Operation	Comparisons	Swaps	Insertions	Time(ns)	Time(ms)
Insertion in Heap	1	0	1	6400	0
Insertion in Heap	2	0	2	1900	0
Insertion in Heap	3	0	3	1500	0
HeapifyUp	4	1	3	4600	0
DecreaseKey	4	1	3	1343500	1
Insertion in Heap	1	0	1	1300	0
Insertion in Heap	2	0	2	1700	0
Insertion in Heap	4	1	3	1700	0

Insertion in Heap	5	1	4	1600	0
MergeTwoMinHeaps	5	1	4	941700	0
Insertion in Heap	1	0	1	1100	0
Insertion in Heap	3	1	2	2400	0
Insertion in Heap	4	1	3	1500	0
Insertion in Heap	1	0	1	1200	0
Insertion in Heap	3	1	2	1600	0
Insertion in Heap	4	1	3	1200	0
DeleteRoot	10	3	3	7000	0

We observe that the data confirms the empirical analysis. The time grows within the expected limits of O(logn) and O(n) for merge two heaps

Conclusion

Analysis of MinHeap implementation shows that the algorithm correctly follows the theoretical behaviour of heap data structures. Empirical results match the expected time complexity:

insertion and deletion operations demonstrate logarithmic growth O(log n),

merging two heaps follows the expected trend O(n)

Overall, the implementation is functionally correct and consistent with theoretical predictions