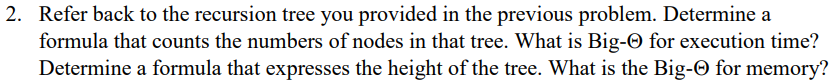
Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

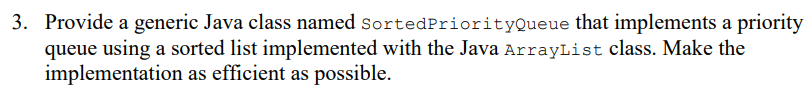
Description automatically generated with medium confidence

In the best case the array is already in sequential order, and the recursion tree would just have the nodes for the insert method, and only one shift node for each insert node. Bringing the total of nodes down to 10.



Text

Description automatically generated



First image below Is my implementation, second is the code to use the queue and show it working. I had this set as a max queue, but can easily be turned into a min queue by negating comparisons after compareTo on lines 16, 35, 38.

Code adapted from: [Binary Heap - GeeksforGeeks](https://www.geeksforgeeks.org/binary-heap/)

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text, letter

Description automatically generated

Execution times for sortedPriorityQueue:

* Worst O(n) for add operation:
  + **O(log n),** since we add new element to the end and just compare it to its parent, we only have to compare the new element to log n nodes already present.
* Worst O(n) for remove operation:
  + **O(log N),** Removing an element happens in O(1) time, because we just have to remove the root and replace it with the last element in the array. However, since this method also calls heapifyMin(0) that needs to sift the root node down, decreasing how many nodes to compare to by log n each level.
* Worst O(n) for sort operation:
  + **O(log n),** since the root may not be in the right place, we check its left and right children and sift it down to its correct position. If the root node needs to be moved all the way to the bottom of the tree, we decrease the number of nodes needed to compare it to by log n each level it goes down.

Execution times for Java Priority Queue:

* Worst O(n) for add operation:
  + O(log n)
* Worst O(n) for remove operation:
  + O(n)
* Worst O(n) for sort operation:
  + Java priority queue uses heap sort (binary heap) so execution time is O(n log n)

I believe my implementation is better in terms of time complexity when comparing the two.

References for #4:

[Priority queue - Wikipedia](https://en.wikipedia.org/wiki/Priority_queue)

[Time Complexity of Java PriorityQueue (heap) insertion of n elements? - Stack Overflow](https://stackoverflow.com/questions/47420638/time-complexity-of-java-priorityqueue-heap-insertion-of-n-elements)