TCSS 342 - Data Structures
Assignment 3 - Priority Queue

Time Estimate: 6-8 hours

Learning Outcomes

The purpose of this assignment is to:

- Build a priority queue data structure out of an array list.
- Implement the priority queue as a minimum heap.
- Gain experience testing and debugging data structures.
- Build data structures exactly to an API's specifications.

Overview

With a few years as Java programmers you may have had an opportunity to use a Java PriorityQueue. In this assignment we will build our own priority queue. There are multiple ways to build this data structure. We have already built one last assignment when we built an ordered list. A (minimum) binary heap is a specialized data structure that also implements a priority queue more efficiently than an ordered list. For this reason we will build our priority queue out of a (minimum) binary heap.

To complete this you will implement one public class:

- MyPriorityQueue<Type>

Formal Specifications

MyPriorityQueue<Type extends Comparable<Type>>
-heap : MyArrayList<Type>
+insert(item : Type) +removeMin() : Type +min() : Type +size() : int +isEmpty() : boolean +toString() : String -bubbleUp() -sinkDown() -parent(index : int) : int -right(index : int) : int -left(index : int) : int

Field summary

- `heap` - Stores the values in the heap in an underlying MyArrayList.

Method summary

- `insert` - Inserts the `item` into the heap and corrects the invariant. It does so by following these steps:
    a. Inserts the `item` at the end of the array list.
    b. Calls bubbleUp to move the `item` to the correct location.

  This method should run in O(lgn) time.
- `removeMin` - Removes the first element and corrects the invariant. It does so by following these steps:
    a. Swaps the first element with the last element.
    b. Removes the last element.
    c. Calls sinkDown to move the first element to the correct location.
    d. Returns the element removed.

  This method should run in O(lgn) time.
- `min` - Returns the first element. This method should run in O(1) time.
- `size` - Returns the number of elements in the heap. This method should run in O(1) time.
- `isEmpty` - Returns true if the heap is empty and false otherwise. This method should run in O(1) time.
- `toString` - Returns the contents of the heap in String format. This method should run in O(n) time.
- `bubbleUp` - Shifts the last element up to a position where it belongs to correct the heap invariant. It does so by swapping the element with its parent if they are out of order (using the compareTo method). Remember that a node should always be greater than its parent in a minimum heap. This method should run in O(lgn) time.
- `sinkDown` - Shifts the first element down to a position where it belongs to correct the heap invariant. It does so by swapping the element with its smallest child if they are out of order (using the compareTo method). Remember that a node should always be less than its children in a minimum heap. This method should run in O(lgn) time.
- `parent` - Returns the index of the parent node in the heap. This method should run in O(1) time.
- `left` - Returns the index of the left child node in the heap of the index passed in.This method should run in O(1) time.
- `right` - Returns the index of the right child node in the heap of the index passed in.This method should run in O(1) time.

Testing

It is important that you test your code to ensure it works on all potential inputs. Please do this in a separate Main class, and without using additional tools (like JUnit). You will not submit your test code. Instead, your data structures will be tested by code developed by the instructor and/or grader. This code will not be provided to you, as you should test your code before submitting it. If your code fails our tests we will let you know which tests it fails so you can find and correct your errors.

Here is the output from my solution:

```
min: null
Heap contents: []
heap.insert(4)
min: 4
Heap contents: [4]
heap.insert(7)
min: 4
Heap contents: [4, 7]
heap.insert(5)
min: 4
Heap contents: [4, 7, 5]
heap.insert(2)
min: 2
Heap contents: [2, 4, 5, 7]
heap.insert(3)
min: 2
Heap contents: [2, 3, 5, 7, 4]
heap.insert(6)
min: 2
Heap contents: [2, 3, 5, 7, 4, 6]
heap.insert(8)
min: 2
Heap contents: [2, 3, 5, 7, 4, 6, 8]
heap.insert(9)
min: 2
Heap contents: [2, 3, 5, 7, 4, 6, 8, 9]
heap.insert(1)
min: 1
Heap contents: [1, 2, 5, 3, 4, 6, 8, 9, 7]
heap.insert(0)
min: 0
Heap contents: [0, 1, 5, 3, 2, 6, 8, 9, 7, 4]
removeMin: 0
```

```
Heap contents: [1, 2, 5, 3, 4, 6, 8, 9, 7]
removeMin: 1
Heap contents: [2, 3, 5, 7, 4, 6, 8, 9]
removeMin: 2
Heap contents: [3, 4, 5, 7, 9, 6, 8]
removeMin: 3
Heap contents: [4, 7, 5, 8, 9, 6]
removeMin: 4
Heap contents: [5, 7, 6, 8, 9]
removeMin: 5
Heap contents: [6, 7, 9, 8]
removeMin: 6
Heap contents: [7, 8, 9]
removeMin: 7
Heap contents: [8, 9]
removeMin: 8
Heap contents: [9]
removeMin: 9
Heap contents: []
removeMin: null
Heap contents: []
```

Submission

You will submit a .zip file containing:

- MyArrayList.java
- MyPriorityQueue.java

Grading Rubric

In order to count as complete your submission must:

1. Implement the priority queue efficiently and without errors.
2. Follow the API exactly so that your classes will work with any testing code.
3. Pass all tests in the testing code. If you fail any tests, you will be told which test failed.

Reminder: Incomplete assignments can always be corrected and resubmitted. If they are completed within 7 days of the due date they will count as late and after that period they will count as missed. Please review the grading matrix for the number of permitted late and missed assignments.