CSE 373, Spring 2011
Homework #5: A Heap o’ Fun (60 points)
Due Wednesday, May 11, 2011, 10:00 PM

This assignment focuses on implementing a Priority Queue ADT using different variations of heaps. Turn in all parts of this assignment electronically by Wednesday, May 11, 2011 at 10:00PM. Your turnin should consist of four files: MinMaxBinaryHeapPQ.java, FourHeapPQ.java, HW5PriorityQueueTest.java (you will modify the provided file to add some testing code), and README.txt. You will need the additional supporting files found of the course's homework webpage: GenericPriorityQueue.java, BinaryHeapPQ.java, and HW5PriorityQueueTest.java.

Step 0 is a paper warm-up exercise where you will practice inserting into and removing from heaps. In Step 1, you will write a generic, binary heap that is able to toggle between a min-heap and a max-heap depending on how it is constructed. In Steps 2 and 3, you will create a generic, four min-heap implementation in which each node in the heap can have up to four children.

**Step 0: Warm-up**

For each of the problems below, draw the heap that results after the requested operations are performed. You will not turn in your drawings. Instead, use the final heap drawing to list the values found in the heap in order from top to bottom, left to right. For example, if your answer produced a final binary heap with a root of 3, 11 as the root's left child, 7 as the root's right child, and 15 as 11's left child, your answer would be "3, 11, 7, 15". In other words, the order you list the values should be the same as the order they would be kept in an array implementation of a heap as described in lecture.

Save the six lists of values (one for each problem) representing your final heap drawings in your README.txt.

1. Draw the heap that results when inserting the following values, one at a time, into an initially empty minimum binary heap: 10, 12, 1, 14, 6, 5, 8, 15, 3, and 9.

2. Draw the heap that results when inserting the following values, one at a time, into an initially empty maximum binary heap: 10, 12, 1, 14, 6, 5, 8, 15, 3, and 9.

3. Draw the heap that results when inserting the following values, one at a time, into an initially empty minimum four heap: 10, 12, 1, 14, 6, 5, 8, 15, 3, and 9. Draw your result as a four tree (a tree where each node can have up to four children) that maintains both the heap structure and min-heap ordering properties.

4. Draw the heap that results after performing three remove operations (i.e. three deleteMin operations) on the heap from #1.

5. Draw the heap that results after performing three remove operations (i.e. three deleteMax operations) on the heap from #2.

6. Draw the heap that results after performing three remove operations (i.e. three deleteMin operations) on the heap from #3.
Step 1: MinMaxBinaryHeapPQ

The goal of this part of the assignment is to make a new class MinMaxBinaryHeapPQ that extends the generic BinaryHeapPQ to allow the client to decide if they would like to have a min-heap or a max-heap. Recall that a min-heap is one where every node in the heap is less than or equal to all of its children and a max-heap is a heap where every node is greater or equal to all of its children.

The BinaryHeapPQ class already has the following constructors and methods (along with a few others):

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public BinaryHeapPQ()</td>
<td>constructs an empty min-heap</td>
</tr>
<tr>
<td>public void add(T value)</td>
<td>adds a value to the min-heap</td>
</tr>
<tr>
<td>public boolean isEmpty()</td>
<td>returns true if the heap has no elements</td>
</tr>
<tr>
<td>public T peek()</td>
<td>returns (but does not remove) the minimum element in the heap</td>
</tr>
<tr>
<td>public T remove()</td>
<td>removes and returns the minimum element in the heap</td>
</tr>
<tr>
<td>public int size()</td>
<td>returns the number of elements in the heap</td>
</tr>
<tr>
<td>public String toString()</td>
<td>returns a String representation of BinaryHeapPQ with values stored with heap structure and order properties</td>
</tr>
<tr>
<td>protected void bubbleDown()</td>
<td>performs the &quot;bubble down&quot; operation to place the element that is at the root of the heap in its correct place so that the heap maintains the min-heap order property</td>
</tr>
<tr>
<td>protected void bubbleUp()</td>
<td>performs the &quot;bubble up&quot; operation to place a newly inserted element (i.e. the element that is at the size index) in its correct place so that the heap maintains the max-heap order property</td>
</tr>
</tbody>
</table>

Your MinMaxBinaryHeapPQ should extend the BinaryHeapPQ class. Therefore, your MinMaxBinaryHeapPQ should have the following class header:

```java
public class MinMaxBinaryHeapPQ<T extends Comparable<T>> extends BinaryHeapPQ<T>
```

At minimum, your MinMaxBinaryHeapPQ should have the following constructors and methods:

- public MinMaxBinaryHeapPQ()
  This is your default constructor for MinMaxBinaryHeapPQ. It should be a min-heap by default.

- public MinMaxBinaryHeapPQ(boolean isMinHeap)
  For this additional constructor, you should create MinMaxBinaryHeapPQ as a min-heap if isMinHeap is true or as a max-heap if isMinHeap is false.

- protected void bubbleDown()
  This method overrides the BinaryHeapPQ's bubbleDown method. In this overridden version, if your MinMaxBinaryHeapPQ was constructed as a min-heap, your method should perform the same as BinaryHeapPQ's bubbleDown. However, if MinMaxBinaryHeapPQ was constructed as a max-heap, your method should perform the bubble down operation by bubbling the root element down to the correct place in the heap such that the heap maintains the max-heap order property after a remove has been performed.

- protected void bubbleUp()
  This method overrides the BinaryHeapPQ's bubbleUp method. If your MinMaxBinaryHeapPQ was constructed as a min-heap, your method should perform the same functionality as BinaryHeapPQ's bubbleUp. However, if MinMaxBinaryHeapPQ was constructed as a max-heap, your method should perform the bubble up operation by bubbling the last inserted value up to the correct place in the heap such that the heap maintains the max-heap order property.
Step 2: FourHeapPQ

The goal of this part of the assignment is to write a generic, four min-heap. A four min-heap is similar to a binary min-heap but each node can have up to four child nodes. Your FourHeapPQ should implement the generic `GenericPriorityQueue` interface and maintain both the heap structure and the minimum heap order property.

You should implement your FourHeapPQ using an array like we did for our BinaryHeapPQ. However, the computations to find the index of a node’s children and parent will be different. You will have to figure these out. Unlike with a binary heap, though, the math is a bit simpler if you put your first element at the 0th index instead of the 1st index as we did in the BinaryHeapPQ. If you decide to go this route, the way the size is maintained for the FourHeapPQ will be a different than the BinaryHeapPQ. If you decide to put the 1st element in the 0th index and you don't adjust the FourHeapPQ's size properly, you will likely get NullPointerExceptions.

The `toString` method of your FourHeapPQ should use `Arrays.toString` to print out the elements in your array (see BinaryHeapPQ’s `toString` method for reference).

We are providing you a few methods that you can use to help you test your FourHeapPQ. The testing code provided is found in HW5PriorityQueueTest.java. However, you should add your own test cases to this file and turn it in along with your FourHeapPQ.java. You will be graded on how well you have tested your code.

![Four min-heap diagram](image)

Figure 1 A four min-heap

Step 3: Write-Up

In addition to the code that you turn in, answer the following questions in a file called README.txt.

1. On inserts, do you expect `BinaryHeapPQ` or `FourHeapPQ` to perform better? Explain why.
2. On remove, do you expect `BinaryHeapPQ` or `FourHeapPQ` to perform better? Explain why.
3. Using the `buildBinaryHeap` and `buildFourHeapPQ` methods in the given `HW5PriorityQueueTest.java` file provided, perform inserts of different sizes and time them. Choose a number of different sizes and include them and the timing results in your README.txt. Empirically, which heap is performing better for inserts? Is it what you expected? If not, why do you think this may be the case?

4. Using the `emptyHeap` method in the given `HW5PriorityQueueTest.java` file provided, perform different number of removes on `BinaryHeapPQ` and `FourHeapPQ` and time them. Choose a number of different sizes and include them and the timing results in your README.txt. Empirically, which heap is performing better for removes? Is it what you expected? If not, why do you think this may be the case?
Style Guidelines and Grading

Part of your grade will come from appropriately utilizing an array data structure to implement your MinMaxBinaryHeapPQ.java and FourHeapPQ.java classes. There will be significant deductions if you use Java collections to implement these classes. Additionally, you should not use any other auxiliary data structures other than the single array in your implementations. Redundancy is another major grading focus; some methods are similar in behavior or based off of each other's behavior. You should avoid repeated logic as much as possible. Your class may have other methods besides those specified, but any other methods you add should be private.

You should follow good general style guidelines such as: making fields private and avoiding unnecessary fields; appropriately using control structures like loops and if/else; properly using indentation, good variable names and types; and not having any lines of code longer than 100 characters.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, and exceptions. The files provided to you use the "doc comments" format used by Javadoc, but you do not have to do this. For reference, our solution for MinMaxBinaryHeapPQ.java is 54 lines long (29 lines if you ignore blank and commented lines) and our solution for FourHeapPQ.java is approximately 181 lines long (79 lines if you ignore blank and commented lines).