CSE 373, Winter 2011
Programming Project #3: A Heap o' Fun (60 points)
Step 0: Due Monday, February 7, 2011, Beginning of Class
Step 1: Due Tuesday, February 8, 2011, 10:00 PM
Steps 2 and 3: Due Wednesday, February 16, 2011, 10:00 PM

This assignment focuses on implementing a Priority Queue ADT using different variations of heaps. This assignment has four steps.

Step 0 (due Monday, February 7 at the beginning of lecture) 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. On Tuesday, February 8, turn in Step 1 electronically by submitting a file named `MinMaxBinaryHeap.java`. You will also need the additional supporting files: `PriorityQueue.java` and `BinaryHeap.java` (similar to what we implemented in class but has slight differences so make sure to download the file off of the homework page).

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. On Wednesday, February 16, turn in Steps 2 and 3 electronically. For these steps, you will be given supporting files `PriorityQueue.java` and `P3PriorityQueueTest.java`. You will turn in `FourHeap.java`, `P3PriorityQueueTest.java` (you will modify the provided file to add some testing code), and `README.txt`.

Step 0: Warm-up

1. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty **minimum binary heap**. Draw your result as a binary tree that maintains both the heap structure and min-heap ordering properties.

2. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty **maximum binary heap**. Draw your result as a binary tree that maintains both the heap structure and max-heap ordering properties.

3. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty **minimum four heap**. 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. Show the result of performing three remove operations (i.e. three deleteMin operations) on the heap from #1. Draw your result as a binary tree that maintains both the heap structure and min-heap ordering properties.

5. Show the result of performing three remove operations (i.e. three deleteMax operations) on the heap from #2. Draw your result as a binary tree that maintains both the heap structure and max-heap ordering properties.

6. Show the result of performing three remove operations (i.e. three deleteMin operations) on the heap from #3. 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.

Step 1: `MinMaxBinaryHeap`

The goal of this part of the assignment is to make a new class `MinMaxBinaryHeap` that extends the generic `BinaryHeap` we wrote in lecture but has some additional functionality that allows 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 `BinaryHeap` class already has the following constructors and methods (along with a few others):

`public BinaryHeap()`	constructs an empty min-heap
`public void add(T value)`	adds a value to the min-heap
`public boolean isEmpty()`	returns `true` if the heap has no elements
`public T peek()`	returns (but does not remove) the minimum element in the heap
`public T remove()`	removes and returns the minimum element in the heap
`public String toString()`	returns a String representation of `BinaryHeap` with values stored with heap structure and order properties
`protected void bubbleDown()`	performs the "bubble down" 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
`protected void bubbleUp()`	performs the "bubble up" 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 min-heap order property

Your `MinMaxBinaryHeap` should extend the `BinaryHeap` class. Therefore, your `MinMaxBinaryHeap` should have the following class header:

```
public class MinMaxBinaryHeap<T extends Comparable<T>> extends BinaryHeap<T>
```

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

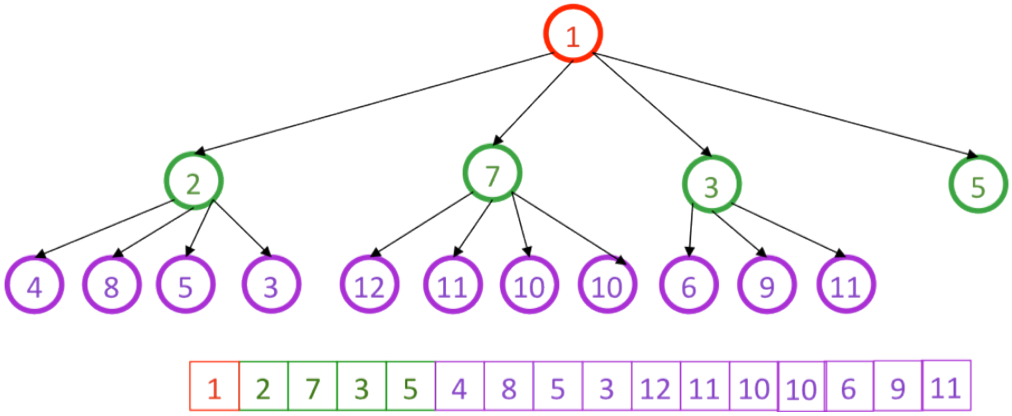
- `public MinMaxBinaryHeap()`
  This is your default constructor for `MinMaxBinaryHeap`. It should be a min-heap by default.
- `public MinMaxBinaryHeap(boolean isMinHeap)`
  For this additional constructor, you should create `MinMaxBinaryHeap` as a min-heap if `isMinHeap` is `true` or as a max-heap if `isMinHeap` is `false`.
- `protected void bubbleDown()`
  This method overrides the `BinaryHeap`'s `bubbleDown` method. In this overridden version, if your `MinMaxBinaryHeap` was constructed as a min-heap, your method should perform the same as `BinaryHeap`'s `bubbleDown`. However, if `MinMaxBinaryHeap` 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 `BinaryHeap`'s `bubbleUp` method. If your `MinMaxBinaryHeap` was constructed as a min-heap, your method should perform the same functionality as `BinaryHeap`'s `bubbleUp`. However, if `MinMaxBinaryHeap` 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: `FourHeap`

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 `FourHeap` should implement the generic `PriorityQueue` interface and maintain both the heap structure and the minimum heap order property.

You should implement your `FourHeap` using an array like we did for our `BinaryHeap`. 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 like we did in the `BinaryHeap`. If you decide to go this route, the way the size is maintained for the `FourHeap` will be a different than the `BinaryHeap`. If you decide to put the 1st element in the 0th index and you don't adjust the `FourHeap`'s size properly, you will likely get `NullPointerException`s.

We are providing you a few methods that you can use to help you test your `FourHeap`. The testing code provided is found in `P3PriorityQueueTest.java`. However, you should add your own test cases to this file and turn it in along with your `FourHeap.java`. You will be graded on how well you have tested your code.



1	2	7	3	5	4	8	5	3	12	11	10	10	6	9	11

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 `BinaryHeap` or `FourHeap` to perform better? Explain why.
2. On remove, do you expect `BinaryHeap` or `FourHeap` to perform better? Explain why.
3. Using the `buildBinaryHeap` and `buildFourHeap` methods in the given `PriorityQueueTest.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 `PriorityQueueTest.java` file provided, perform different number of removes on `BinaryHeap` and `FourHeap` 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 `MinMaxBinaryHeap.java` and `FourHeap.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 `MinMaxBinaryHeap.java` is 54 lines long (29 lines if you ignore blank and commented lines) and our solution for `FourHeap.java` is approximately 181 lines long (79 lines if you ignore blank and commented lines).