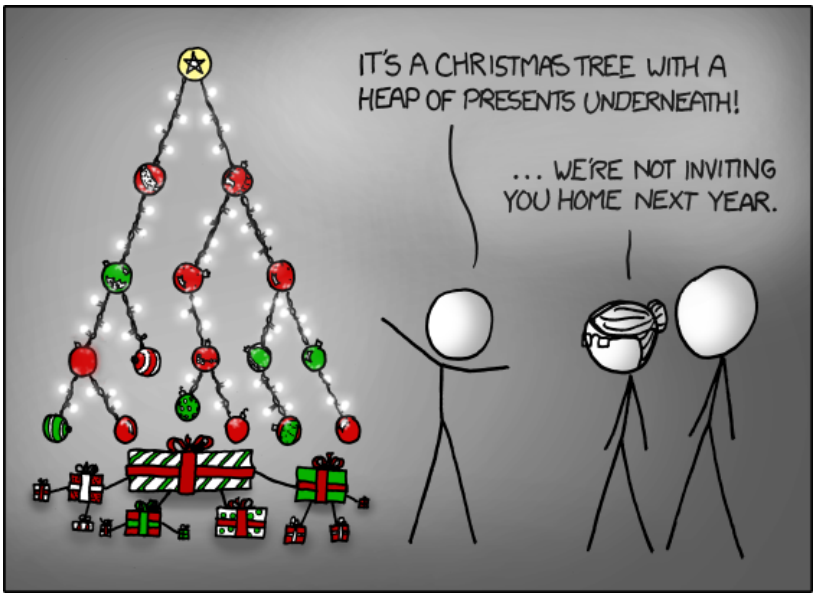
CS201 - Assignment 9
Due: Wednesday April 30, at the beginning of class

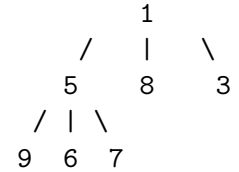


http://xkcd.com/835/

In this assignment we will be playing around with heaps other than binary heaps and the effect that this has on run-time. This is a solo assignment.

n-ary heaps

So far, all of the heaps we have looked at have been binary heaps where each node has two children. As mentioned in class, we can also think about n-ary trees where rather than having two children, each node has n children. An n-ary heap, is a heap that is based on an n-ary tree. For example:



is a 3-ary heap. The definition of the heap remains the same where a parent's value must be smaller than or equal to that of its children and the tree must be complete. The only difference is that rather than having two children, each node has n children.

As with a binary heap, you can implement an n-ary heap using an `ArrayList` to store the data. The heap above would be stored as:

```
[1, 5, 8, 3, 9, 6, 7]
```

You are to implement a new class that represents a heap as an `ArrayList` but has an addtional parameter to the constructors n that allows you to build an n-ary heap, rather than just a binary heap. The class should be defined as:

```
ArrayListNPriorityQueue<E extends Comparable<E>> implements PriorityQueue<E>
```

You need to implement all of the required methods for the `PriorityQueue` interface. In addition, you should provide two constructors, one that builds an empty n-ary heap:

```
public ArrayListNPriorityQueue(int numChildren)
```

and one constructor that builds an n-ary heap from data:

```
public ArrayListNPriorityQueue(ArrayList<E> data, int numChildren)
```

This constructor *must* use the `heapify` approach for building the heap rather than simply adding the data items.

Time

Once you have this working, I'd like you to investigate how the size of n in your n-ary heap affects the performance of the heap. To do this, I'm going to have you experiment with the data and then give a short write-up explaining your results.

First, experiment how the size of n changes the run-time of the heap. In particular, vary n from 2 to 20. For each n, see how how long it takes to remove all of the elements from the heap using calls to `extractMin`. For each n, average the results over 10-20 runs to get better data. A heap of size 5,000-10,000 elements is big enough to get good data, but not so large that it is too computationally expensive.

Work towards generating a table that looks something like:

```
degree |          time |
     2 | <some_number> |
     3 | <some_number> |
     4 | <some_number> |
       ...
```

Once you have this data, think about what is happening and why. In addition to submitting the code above for the `ArrayListNPriorityQueue` class, you also need to submit a short write-up. Your write-up should include:

- Your name at the top :)

- Your table showing your data.
- One paragraph explaining what you see in your data and why you see that behavior. You should talk about how your results match with what we would theoretically expect to see.

Make sure to look at the submission instructions for making sure that your writeup gets included in your submission.

Getting started

I've included copies of the the `StopWatch` class and the `PriorityQueue` interface at:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign9/part2/`

When You're Done

Make sure you have comments at the top of any file/class that you wrote with with your name and the assignment number.

JavaDoc

All of your methods MUST have JavaDoc style headers. I've included it for many of the methods, but make sure you include it for any additional ones you write.

To include the writeup in your .jar file:

- Add your writeup to your Eclipse project. To do this you have a couple of options:
  - Drag the file and drop it in your project in Eclipse.
  - Right-click on the project and select "New->File". For the location, select the project (not the src directory) and then give it a name. This will create a blank file in your project. Copy and paste your experiment writeup into the file and then save it.
- When you generate your .jar file (see below if you're rusty) make sure that your .txt file is selected when you're exporting.

Generating the jar file:

1. Right-click on the project (on Mac, ctrl+click) and select `Export`.
2. You'll see a number of options. Open up the `Java` folder and select `JAR file` and click `Next`.
3. You should just see your project selected. Below, make sure **only** the following two options are checked:

- "Export Java source files and resources"
- "Compress the contents of the JAR File"

4. Click on the `Browse...` button and pick a location to save the output file. Give the file a name like "kauchak1.jar", where "kauchak" is your last name and "1" is the assignment number.
5. Click `Finish`.

If all went well this will generate a single `.jar` file wherever you picked to save it.

Submit this JAR file as assignment number "9.2" via the online submission mechanism on the course web page.