COLBY COLLEGE
Department of Mathematics
and Computer Science

CS231 **Allen B. Downey**
Fall 1998

Assignment 9

Due: Friday 4 December

Reading: Standish Chapter 8.6, 8.7, 8.11, and skim 8.8, and also review a copy of notes9.txt from the web page. Finally, look at the code in the Heap directory and the PriorityQueueGeneric directory of the coded solutions.

Big Picture: Using the heap implementation from last week, we're going to build a generic Heap implementation similar to the generic Priority Queue implementation from way back when, and test it on a simple application—reading and sorting a grade sheet.

Before you begin, you should have a working implementation of the Heap class, along with a `main` method that tests it by inserting 100 randomly-generated integers into the Heap and the removing them. They should be in order. I recommend that you print the Heap when it is full so you can confirm that it is a complete tree.

Fix the performance issue

In the solution we developed to last week's homework, we have a performance problem, which is that `insert` needs to know the rank of every node in the tree, but calculating those ranks takes time proportional to the number of nodes, O(n) time.

But a proper implementation of a Heap should be able to perform an insertion or deletion in time proportional to the height of the heap, which for a well-balanced tree is only O(log n) time.

1. Modify your solution so that both insertion and deletion take time proportional to the height of the Heap.

Make it a generic ADT

As an intermediate step toward the grade-sheet sorter, we're going to make a generic version of Heap and test it using a simple integer Sorter.

1. Copy the Comparable class definition from the Priority Queue assignment. We're going to adapt Heap to work with Comparable items and create a simple class that is a member of the Comparable abstract class.
2. Make a new class named HeapItem with a single integer as an instance variable. It should implement Comparable (and provide methods named `compareTo` and `toString`). See PQItem for an example of how to do this.
3. Compile and run the three class definitions so that they work again, generating 100 random numbers, inserting them into the Heap, then removing and printing them.

4. Move the `main` method from Heap.java into HeapItem.java. You will have to change the startup file, and you will have to qualify some of the method names that didn't use to need it.

Put it all together

1. Create a new class named Student, based on HeapItem. It should have instance variables to contain the name and test score for each student, and it should implement the Comparable abstract class.

2. Create a new class named Sorter that will contain `main`. Sorter will read a file, create Student objects, put the Student objects in a Heap, and then remove and print them (in sorted order). You might want to use Balance.java (in the Balance directory under coded solutions on the web page) as a template for Sorter, since it contains the basic loop for reading lines from a file.

3. Test your program with a range of inputs including some cases with errors, and any other things that are likely to cause problems (names with more than two words, or with punctuation, or students with very large, negative, non-existent, or identical test scores).