CSE 326 DATA STRUCTURES
HOMEWORK 2

Due: **Wednesday, July 4, 2007** at the beginning of class.

Problem 1. Binary Min Heaps — problem 6.2

This problem will give you some practice with the basic operations on binary min heaps.

(a) Starting with an empty binary min heap, show the result of inserting, in the following order, 10, 12, 1, 14, 6, 5, 8, 3, 9, 13, and 2, one at a time (using percolate up each time), into the heap. By *show* here we mean "draw the resulting binary tree with the values at each node."

(b) Now perform two `deleteMin` operations on the binary min heap you constructed in part (a). Show the binary min heaps that result from these successive deletions ("draw the resulting binary tree with values at each node").

(c) Instead of inserting the elements in part (a) into the heap one at a time, suppose that you use the linear time worst case algorithm described on page 211 of Weiss (Floyd's algorithm). Show the resulting binary min heap tree. (It would help if you showed the intermediate trees so if there are any bugs in your solution we will be better able to assign partial credit, but this is not required.)

Problem 2. findMax for Min Heaps — problem 6.8

We showed in class that binary min heaps are capable of finding the minimum element of the heap in $O(1)$ time. This problem asks you to prove that finding the *maximum* element takes $O(N)$ time. To show this, prove carefully that:

(a) The maximum element of a min heap must be one of the leaves.

(b) There are exactly $\lceil N/2 \rceil$ leaves.

(c) Every leaf must be examined to find the maximum.

(d) Therefore findMax must take $O(N)$ time.

Problem 3. Min-Max Heaps — problem 6.18

We just saw that binary min heaps can't find the maximum element in an efficient way. One solution to this is to use what is called a Min-Max heap. In this problem you'll explore the min-max heap and give algorithms for insertion and deletion of the min and max into the min-max heap.

A min-max heap is a data structure that supports both deleteMin and deleteMax (and therefore findMin and findMax) in $O(\log N)$ time. The structure is the same as a binary heap, but the heap-order property is that for any node X at *even* depth, the element stored at X is smaller than the parent but larger than the grandparent (where this makes sense), and for any node X at *odd* depth, the element stored at X is larger than the parent but smaller than the grandparent.

(a) How do we find the minimum and maximum elements? How long does this take?

(b) Give an algorithm to insert a new node into a min-max heap.

(c) Give algorithms to perform deleteMin and deleteMax.

(d) Can you build a min-max heap in linear time? If yes, give an algorithm; if no, give a proof.

For parts (b) and (c), your answers should be given using a sufficiently precise pseudo-code to make your algorithms clear, but they need not be a syntactically correct and legal program in Java or other programming language.

Problem 4. An $m \times n$ Young tableau is an m by n matrix such that the entries of each row are in sorted (increasing) order from left to right, and the entries of each column are sorted from top to bottom. Some of the entries of a Young tableau may be ∞, which we treat as nonexistent. So a given Young tableau can hold $r \leq mn$ elements.

(a) Draw a 4×4 Young tableau containing the elements $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

(b) Argue that an $m \times n$ Young tableau Y is empty if $Y[1, 1] = \infty$. Argue that Y is full (i.e. contains mn elements) if $Y[m, n] < \infty$.

(c) Give an algorithm to implement deleteMin on a nonempty $m \times n$ Young tableau that runs in $O(m + n)$ time. Your algorithm should use a recursive subroutine that solves an $m \times n$ problem by recursively solving either an $(m - 1) \times n$ subproblem or an $m \times (n - 1)$ subproblem. Hint: Think about a recursive version of deleteMin. Prove your time bound.

(d) Show how to insert a new element into a non-full $m \times n$ tableau in $O(m + n)$ time.

(e) Give an $O(m + n)$-time algorithm to determine whether a given number is present within the tableau.

(f) Which ADT is more efficient for implementing priority queues: binary heaps, or Young tableaus? (Both support findMin, insert and deleteMin, so both can be used as a priority queue.) Which is faster, and by how much?