CSE 326 DATA STRUCTURES
HOMEWORK 2

Due: **Friday, January 22, 2010** at the beginning of class. Be sure to write your name AND YOUR QUIZ SECTION on your homework.

Problem 1. Binary Min Heaps

This problem will give you some practice with the basic operations on binary min heaps. Make sure to check your work!

(a) Starting with an empty binary min heap, show the result of inserting, in the following order, 10, 12, 14, 1, 6, 15, 8, 5, 3, 9, 7, 11, 4, 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." Check your work!

(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. Merging "Full" Complete (aka Perfect) Binary Heaps

Floyd's method provides an $O(n)$ algorithm for merging binary heaps. However, there are special values of n for which we can do better. Here, we consider the case of merging two binary heaps that are full, i.e., every level i contains 2^i nodes. (We also described this as a "perfect" tree in lecture.)

(a) Weiss problem 6.17(a). A concise answer in words is sufficient. In addition, explain why your algorithm has the stated worst case complexity.

(b) Weiss problem 6.17(b). A concise answer in words is sufficient. In addition, explain why your algorithm has the stated worst case complexity.

Problem 3. Alternate `remove()` Algorithm for Heaps

One way to remove an object from a binary (min) heap is to decrease its priority value by ∞ and then call `deleteMin()`. An alternative is to simply remove it from the heap, thus creating a hole, and then repair the heap.

(a) Write pseudocode for an algorithm that will perform the `remove` operation according to the alternative approach described above. Your pseudocode should implement the method call `remove(int index)`, where `index` is the index into the heap array for the object to be removed. Your pseudocode can make calls to the following methods described in lecture: `insert()`, `deleteMin()`, `percolateUp()`, and `percolateDown()`. In the lecture pseudocode, the objects are the values themselves (rather than being objects that have a value field and a pointer to some other object of a different type); you may follow the lecture's example.

(b) What is the worst case complexity of this algorithm?

Problem 4. d-Heap Arithmetic

Binary heaps implemented using an array have the nice property of finding children and parents of a node using only multiplication and division by 2 and incrementing by 1. This arithmetic is often very fast on most computers, especially the multiplication and division by 2, since this corresponds to simple bitshift operations. In *d*-heaps, the arithmetic is also fairly straightforward, but is no longer necessarily as fast. In this problem you'll figure out exactly what this math is.

(a) If a d-heap is stored as an array, for an entry located in position i, where are the parents and children relative to i? You may find it convenient to place the root at position 0 instead of 1 to simplify calculations (be sure to specify if you make this change). Hint: the solution should be very concise—if it's becoming complicated, you might want to rethink your approach.

(b) (There is only a part a to this question. :-)