CSE 332 Data Abstractions, Winter 2011

Homework 2

Due Friday, Jan 21, 2011 at the **beginning** of lecture. Please be sure your work is readable (either written clearly or typed). This homework has four problems. You should refer to the written homework guidelines for a reminder about what is acceptable *pseudocode*.

Problem 1: Binary Heaps

This problem gives 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, 12, 9, 3, 8, 5, 6, 14, 1, 7, 11 and 2, one at a time (using percolate up each time), into the heap. By show, we mean, "draw the tree resulting from each insert."

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, again by drawing the binary tree resulting from each delete.

c. Instead of inserting the elements in part (a) into the heap one at a time, suppose that you use the linear-time buildHeap operation known as Floyd's algorithm. Show the binary min heap tree that results from buildHeap. (It will help to show some intermediate trees so that 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 Perfect Trees

If we have two binary min heaps h1 and h2 of total size n, then Floyd's algorithm lets us combine them to build a new heap with all the elements from the two heaps in time O(n). However, there are special cases where we can do better. Here we consider the case where h1 and h2 are both perfect trees: a perfect tree is a binary tree where every level i contains 2^i nodes. That is, the rows of h1 and h2 are always full. We also allow the merge operation to "re-use/destroy" h1 and h2, i.e., only the merged heap is available after the merge operation. We assume the heaps are represented as pointer-based trees (not arrays) but we can find the "last" element of h1 and h2 in O(1) time.

a. Suppose h1 and h2 have the same height. Describe an O(log n) algorithm to merge the heaps. A concise English answer is sufficient.

b. Suppose the height of h1 is the height of h2 minus one. Describe an O(log n) algorithm to merge the heaps. A concise English answer is sufficient.

(See back of this page for remaining problems)

Problem 3: Alternative Remove

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

a. Write pseudocode for an algorithm that performs the remove operation using 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 – assume an array representation. Your pseudocode can call the following methods described in lecture: insert, deleteMin, percolateUp, and percolateDown. Like in lecture, you may assume that objects are just priority integers (no other data).

b. What is the worst case complexity of the algorithm you wrote in part (a)?

Problem 4: Binary Min Heap Algorithms

For both of these problems, for full credit your solution should be the most efficient possible – perhaps in the worst case they might need to examine every element in the heap, but in general this should not be the case. You may assume an array layout of the binary min heap as discussed in lecture and in the book. You also may assume that your algorithm has direct access to the heap array (it does not need to manipulate it just by using the standard heap operations – insert, deletemin, findmin, etc.). Your algorithm should not modify the heap (just like a findmin does not modify the heap) – or at the very least, if it does, it should put it back identical to how it was before you started.

a. Give pseudocode for an efficient algorithm to find the maximum value in a binary min heap. What is the worst case big-O running time or your algorithm?

b. Give pseudocode for an efficient algorithm to find all nodes less than a given value in a binary min heap. Your algorithm should just print out the values it finds. Note that the "given value" is not necessarily in the heap. I could ask you to find all values less than 34 in the heap, where the value 34 is not in the heap. What is the worst case big-O running time of your algorithm?