CSE 332 Data Abstractions, Spring 2016
Homework 2

Due: **Wednesday, April 13, 2016** at the BEGINNING of lecture. Your work should be readable as well as correct. You should refer to the written homework guidelines on the course website for a reminder about what is acceptable pseudocode.

Problem 1. Binary Min-Heaps

This problem will give you some practice with the basic operations on binary min heaps. You are welcome to show more intermediate trees or arrays than the numbers listed below if you like.

(a) Starting with an empty binary min heap, show the result of inserting, in the following order, 12, 10, 4, 8, 6, 7, 15, 3, 14, 9, and 2, one at a time (using percolate up each time), into the heap. Be sure to draw the result after **every** insertion. By show here we mean draw the resulting binary tree with the values at "each node." In addition, give the array representation of your **final** answer. We expect 11 trees and 1 array as your answer.

(b) Instead of inserting the elements in part (a) into the heap one at a time, suppose that you use Floyd's buildheap 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). In addition, give the array representation of your final answer. We expect 1 tree and 1 array as your answer.

(c) Now perform TWO deleteMin operations on the binary min heap you constructed in part (b). Show the binary min heaps that result from these successive deletions ("draw the resulting binary tree with values at each node"). Be sure to draw the result after **every** deletion. In addition, give the array representation of your **final** answer. We expect 2 trees and 1 array as your answer.

Problem 2. Two 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. Be sure to answer all four parts (a-d) of the question!

(a) Write pseudocode for an efficient algorithm that will find the maximum value in a binary min heap.

(b) What is the worst case complexity of the algorithm you wrote in part (a)? Give your answer in big-O.

(c) Write pseudocode for an efficient algorithm that will find all values 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. We could ask you to find all values less than 42 in the heap, where the value 42 is not in the heap.

(d) What is the worst case complexity of the algorithm you wrote in part (c)? Give your answer in big-O. What is the runtime of your algorithm if it finds k values less than the input value?

Problem 3. 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 these correspond to simple bitshift operations. In d-heaps, the arithmetic is also fairly straightforward, but is no longer necessarily as fast. In this problem you will figure out how the arithmetic works in those heaps. In case the general idea is not clear, d-Heaps are discussed in section 6.5 of Weiss.

(a) We will begin with considering a 3-heap (a heap where each node has ≤ 3 children. If a 3-heap is stored as an array, for an entry located at index i, what are the indices of its parent and its children? You may find it convenient to place the root at index 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 is becoming complicated, you might want to rethink your approach.

(b) Generalize your solution from (a) to work for d-heaps in general. If a d-heap is stored as an array, for an entry located at index i, what are the indices of its parent and its children?

(c) For what values of d will these operations be implementable with bit shifts instead of divisions and multiplications?

(d) If a d-heap has height h, what is the maximum number of nodes that it can contain? What is the mininum? (again, give an exact expression, NOT something in big-O or theta etc.) SHOW how you came up with your answer.

(e) If a d-heap has n nodes, what will its height be? (give an exact expression, not something in big O or theta etc.) SHOW how you came up with your answer.