MA/CSSE 473 – Design and Analysis of Algorithms

Homework 10 (80 points total) Updated for Winter, 2017

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

6.1.1 [6.1.2] (closest numbers in an array with pre-sorting)
6.1.2 [6.1.3] (intersection with pre-sorting)
6.1.8 [6.1.10] (open intervals common point)
6.1.11 (anagram detection)
6.2.8ab (Gauss-Jordan elimination)
6.3.9 (Range of numbers in a 2-3 tree)
6.5.3 (efficiency of Horner's rule)
6.5.4 (example of Horner's rule and synthetic division)
7.1.7 (virtual initialization)

Problems to write up and turn in:

1. (20) Not in book (sum of heights of nodes in a full tree) In this problem, we consider completely full binary trees with N nodes and height H (so that $N = 2^{H+1} - 1$)

   (a) (5 points) Show that the sum of the heights of all of the nodes of such a tree can be expressed as $\sum_{k=0}^{H} k\, 2^{H-k}$.

   (b) (10 points) Prove by induction on H that the above sum of the heights of the nodes is N - H - 1. You may base your proof on the summation from part (a) (so you don't need to refer to trees at all), **or** you may do a "standard" binary tree induction based on the heights of the trees, using the definition that a non-empty binary tree has a root plus left and right subtrees. I find the tree approach more straightforward, but you may use the summation if you prefer.

   (c) (3 points) What is the big Θ estimate for the sum of the *depths* of all of the nodes in such a tree?

   (d) (2 points) How does the result of parts (b) and (c) apply to Heapsort analysis?

   Example of height and depth sums: Consider a full tree with height 2 (7 nodes).
   Heights: root:2, leaves: 0. Sum of all heights: 1*2 + 2*1 + 4*0 = 4.
   Depths: root: 0, leaves: 2. Sum of all depths: 1*0 + 2*1 + 4*2 = 10.
   [**Response to a 201640 student question on Piazza:** You should compare the naive approach to building the heap in preparation for heapsort (inserting the elements one at a time, Levitin calls it *heaptopdown*) vs. the more efficient approach (Levitin calls it *heapbottomup*) approach. Weiss has more details in Chapter 21. Next, what is the impact of the heap-building algorithm in the running time of the entire heapsort algorithm?

2. (6) 6.4.2 Heap Checking
   [{"uniqueId":"B018ANVSFE","libraryId":"","type":"track","asin":"B018ANVSFE"}]

3. (15) 6.4.6 PQ implementations.
   Present your answer as a table whose columns are the 5 implementations (in the order given) and whose rows are findmax, deletemax, insert (in that order).

4. (10) 6.4.12 [6.4.11] (spaghetti sort)

5. (4) 6.5.10 [6.5.9] (Use Horner's rule for this particular case?)

6. (10) 7.1.6 (ancestry problem)

You may **NOT** assume any of the following:

- The tree is binary
- The tree is a search tree (i.e. that the elements are in some particular order)
- The tree is balanced in any way.

The tree for this problem is simply a connected directed graph with no cycles and a single source node (the root).

7. (15) Not in textbook. (tile grid with pluses and minuses)

For what values of n can we fill an n-by-n grid with + and – signs, such that each square has exactly one neighbor of the opposite sign?
A *neighbor* is an adjacent square that is in the same row or column. Hint: Try to solve the puzzle for n=2, n=3, n=4.
For all "valid" n, show (or describe) all the ways of tiling the grid. For "invalid" n, show that it cannot be done.