MA/CSSE 473 – Design and Analysis of Algorithms

Homework 10 (70 points total) Updated for Summer, 2016

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. (10) 6.1.5 [6.1.7] (to sort or not to sort)
2. (10) 6.2.8c (compare Gaussian Elimination to Gauss-Jordan) **You should compute and compare actual number of multiplications, not just say that both are Θ(n^3). Use division when you compare.**
3. (6) 6.3.7 (2-3 tree construction and efficiency) Show the steps in the construction and show your calculation of the average key comparisons.
4. (3) 6.3.8 (2-3 tree vs. binary tree). Include a proof if it is true, or a counterexample if it is false.
5. (3) 6.3.9 (range of a 2-3 tree)
6. (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 = 3.
   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?
7. (10) 6.4.12 [6.4.11] (spaghetti sort)
8. (4) 6.5.10 [6.5.9] (Use Horner's rule for this particular case?)
9. (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).