MA/CSSE 473 HW 10 textbook problems and hints

Problem 1 (20) not from textbook

(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 $\sum_{k=0}^{H} k2^{H-k}$.

(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?

Problem 2 6.4.2 (6)

2. Outline an algorithm for checking whether an array $H[1..n]$ is a heap and determine its time efficiency.

Assume a max heap.

Author's hint:

2. For a heap represented by an array, only the parental dominance requirement needs to be checked.

Problem 3 6.4.6 (15) PQ implementations

present your answer as a table whose columns are the five implementations (in the order given) and whose rows are findmax, deletemax, insert, (in that order).

Indicate the time efficiency classes of the three main operations of the priority queue implemented as

(a) an unsorted array.
(b) a sorted array.
(c) a binary search tree.
(d) an AVL tree.
(e) a heap.

Author's hint:

Is totally useless. It just re3states the problem.

Problem 4 6.4.12 [6.4.11] (10)

11. *Spaghetti sort* Imagine a handful of uncooked spaghetti, individual rods whose lengths represent numbers that need to be sorted.

    a. Outline a "spaghetti sort"—a sorting algorithm that takes advantage of this unorthodox representation.

    b. What does this example of computer science folklore (see [Dew93]) have to do with the topic of this chapter in general and heapsort in particular?

Author's hint:

11. Pick the spaghetti rods up in a bundle and place them end-down (i.e., vertically) onto a tabletop.

Problem 5 6.5.10 [6.5.9] (4)

9. Is it a good idea to use a general-purpose polynomial evaluation algorithm such as Horner's rule to evaluate the polynomial $p(x) = x^n + x^{n-1} + \dots + x + 1$?

Author's hint:

9. Use a formula for the sum of the terms of this special kind of a polynomial.

Problem 6 7.1.6 (10)

6. ▷ The ***ancestry problem*** asks to determine whether a vertex u is an ancestor of vertex v in a given binary (or, more generally, rooted ordered) tree of n vertices. Design a $O(n)$ input enhancement algorithm that provides sufficient information to solve this problem for any pair of the tree's vertices in constant time.

Author's hint:

Take advantage of the standard traversals of such trees.

Instructor notes

7. (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 is simply a connected directed graph with no cycles and a single source node (the root)

Problem 7 Not in textbook (15)

(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.