CS 430 – FALL 2017
INTRODUCTION TO ALGORITHMS
HOMEWORK #4
DUE Fri Oct 13, 11:25am

1. (3 points) Consider the following algorithm for finding the "ith smallest element out of n elements":

    (a) Place the first i elements into a data structure S.

    (b) For each of the remaining elements, first insert the element into S, then remove the largest element from S. Thus, S always contains the "i smallest elements examined so far".

    (c) Return the largest element from S.

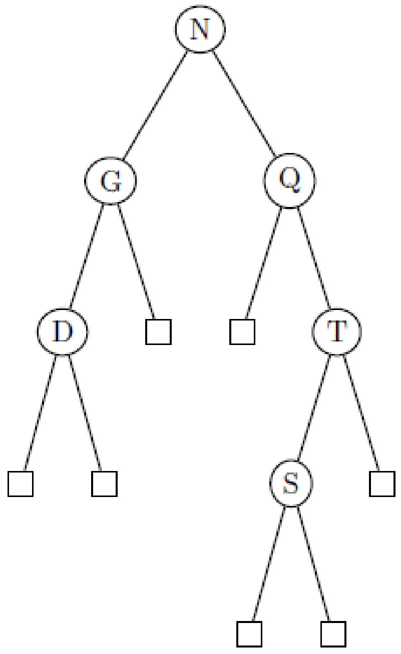
Discuss the pros and cons, including runtime analysis, of each of the following choices for the data structure S:

a red-black tree a min-heap a max-heap

Which data structure would be best for S? Justify your answer.

2. (3 points) The external path length of a full binary tree is the sum, taken over all nil leaves of the tree, of the depth of each leaf.

Given the following binary search tree, in which internal nodes are shown as circles and external nodes (leaves; nil pointers) are shown as small boxes:



2a) Calculate the external path length.

2b) Is there a binary search tree on the same set of letters, {D, G, N, Q, S, T}, that has lower external path length? Either give such a tree and how it is formed from the original BST or prove it does not exist.

2c) Can the nodes in the original BST be colored red and black so as to form a proper red-black tree? If so, show how to color them; if not, prove it.

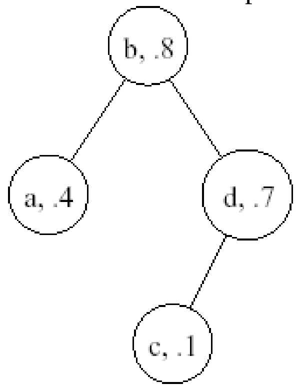
3. (4 points) Exercise 13.1-5

4. (3 points) How many rotations (in the worst case) are required for an insertion of a new key into a red-black tree already containing "n" nodes? Explain.

5. (4 points) Exercise 13.3-2

6. (4 points) Exercise 13.4-3

7. (4 points) Suppose that we would like to keep a set of keys in a binary search tree. New keys will be added at various times, and searches may occur at any moment. Of course we'd like the tree to be balanced at all times, so that we can search and insert in O(lg n) time. One way to try to keep the tree balanced is the following randomized scheme: upon insertion, assign each key a random real-valued score from [0 … 1]. Keep the keys in a structure that is a binary search tree on the keys and a heap on the scores. (I.e., if you only look at the keys of the resulting data structure, what you see is a binary search tree. If you only look at the scores, what you see is a heap.) This structure is called a treap, because it is both a tree and a heap. Here is an example with keys a, b, c, d and associated random scores .4, .8, .1, .7



a) Prove (by using strong induction) that when the keys are distinct and each key has a fixed score, there is a unique structure that is both a binary search tree on the keys and a heap on the scores. In other words, prove that there is only one possible treap for the given assignment of scores to keys. You may also assume that the scores are distinct, because with probability 1 they are.
b) Argue that regardless of the insertion order of the keys, a treap has expected height O(lg n) . Hint: See section 12.4.
c) Discuss how the TREAP-INSERT algorithm would work. Assume that a random score has already been attached to the element to be inserted.