Assignment 9

R-3.11 Consider the following sequence of keys:

(5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 13, 33)

Consider the insertion of items with this set of keys, in the order given, into:

a. an initially empty (2,4) tree *T'*.
b. an initially empty red-black tree *T''.*

Draw *T'* and *T''* after each insertion.

At each step you can check your diagram using the following simulators:
http://cs.armstrong.edu/liang/animation/web/24Tree.html
https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

R-3.14 For each of the following statements about red-black trees, determine whether it is true or false. If you think it is true, provide a justification. If you think it is false, give a counterexample.

a. a subtree of a red-black tree is itself a red-black tree.
b. the sibling of an external node is either external or it is red.
c. given a red-black tree *T*, there is a unique (2,4) tree *T'* associated with *T*.
d. given a (2,4) tree *T*, there is a unique red-black tree *T'* associated with *T*.

Design a pseudo code algorithm **isValidAVL(T)** that decides whether or not a binary tree is a valid AVL tree. For this problem, we define valid to mean that the height of the left and right sub-trees of every node do not differ by more than one.

What is the time complexity of your algorithm?

Design an algorithm, **isPermutation(A,B)** that takes two sequences A and B and determines whether or not they are permutations of each other, i.e., they contain same elements but possibly occurring in a different order. Assume the elements in A and B cannot be sorted. **Hint**: A and B may contain duplicates. Same problem as in previous homework, but this time use a dictionary to solve the problem.

What is the worst case time complexity of your algorithm? Justify your answer.

C-3.10 Let D be an ordered dictionary with n items implemented by means of an AVL tree (or a Red-Black tree). Show how to implement the following operation on D in time O(log n + s), where s is the size of the iterator returned:

FindAllInRange(k1, k2):

Return an iterator of all the elements in D with key k such that k1 $\leq$ k $\leq$ k2.