**Assignment 14**

R-3.11 Consider the following sequence of keys: (5, 16, 22, 45, 2, 10, 18, 30, 50, 12, 1, 7, 55)

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

1. an initially empty (2,4) tree T’.

Diagram

Description automatically generated

1. an initially empty red-black tree T’’.

A picture containing indoor

Description automatically generated

Draw T’ and T’’ after each insertion.

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

1. a subtree of a red-black tree is itself a red-black tree. --- **False**

Diagram

Description automatically generated

***if the node 0009 is red, the subtree 0009 is not a red-black tree.***

1. the sibling of an external node is either external or it is red. ---**False.**

***As the example above, 0010 is sibling node of an external node of the node 0015, but the node 0010 is black.***

1. given a red-black tree T, there is an unique (2,4) tree T’ associated with T. ---**True**

***Because a red node always belongs to a black parent node***

A picture containing text, watch, clock

Description automatically generated

1. given a (2,4) tree T, there is an unique red-black tree T’ associated with T. -- **False.**

***For example: the node (3,5) as below can perform to 2 types of red-black tree***

A picture containing text, clock

Description automatically generated

1. Design a pseudo-code 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**. Hint: A and B may contain duplicates. Same problem as in previous homework, but this time use a dictionary to solve the problem.

Algorithm isPermutation(A, B)

D <- new Dictionary(HT)

for each **a** in A.elements() do

D.insertElement(a, a)

for each **b** in B.elements() do

p <- D.findElement(b)

if p = NO\_SUCH\_KEY then

return false

else

D.removeElement(b)

return true

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

O(n)

3. Design and solve this problem in four ways in JavaScript:

a. By sorting A and B

b. Using a Priority Queue

c. Using a Hash Table based Dictionary

d. Using a BST based Dictionary

4. Assume the elements in A and B cannot be sorted, i.e., there is no comparator. How would this restrict the way you would have to implement a solution to isPermutation(A,B), i.e., which of the above strategies could you use and which couldn’t you use?

5. Which of the above strategies leaves the inputs A and B unchanged?

6. Are any of the approaches considered in-place?

7. Calculate the height of a Binary Tree. Implement your solution in the JavaScript file RBTree-HW.js that is provided. You are to do this both as a recursive function that traverses the tree and secondly using the Euler Tour template class (i.e., implement two different functions in JavaScript).