CS 61B Summer 2020 | Small Group Tutoring Section 7: BSTs and Balanced Trees | Worksheet 7

1 Runtime Questions

Provide the best case and worst case runtimes in theta notation in terms of N, and a brief justification for the following operations on a binary search tree. Assume N to be the number of nodes in the tree. Additionally, each node correctly maintains the size of the subtree rooted at it. [Taken from Final Summer 2016]

```
boolean contains(T o); // Returns true if the object is in the tree
```

Best: Θ(   )   Justification:

Worst: Θ(   )   Justification:

```
void insert(T o); // Inserts the given object.
```

Best: Θ(   )   Justification:

Worst: Θ(   )   Justification:

```
T getElement(int i); // Returns the ith smallest object in the tree.
```

Best: Θ(   )   Justification:

Worst: Θ(   )   Justification:

2 Is This a BST?

(a) The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which `brokenIsBST` fails.

```
public static boolean brokenIsBST(TreeNode T) {
    if (T == null) {
        return true;
    } else if (T.left != null && T.left.val > T.val) {
        return false;
    } else if (T.right != null && T.right.val < T.val) {
        return false;
    } else {
        return brokenIsBST(T.left) && brokenIsBST(T.right);
    }
}
```

(b) Now, write `isBST` that fixes the error encountered in part (a).
Hint: You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

```
public static boolean isBST(TreeNode T) {
    return isBSTHelper(                                            );
}

public static boolean isBSTHelper(                                    ) {
















}
```

3 Pruning Trees

Assume we have some binary search tree, and we want to prune it so that all values in the tree are between *L* and *R*, inclusive. Fill out the method below that takes in a BST, as well as *L* and *R*, and returns the pruned tree. Note that the root of the original tree might not be between *L* and *R*, so make sure you return the root of the new pruned tree.

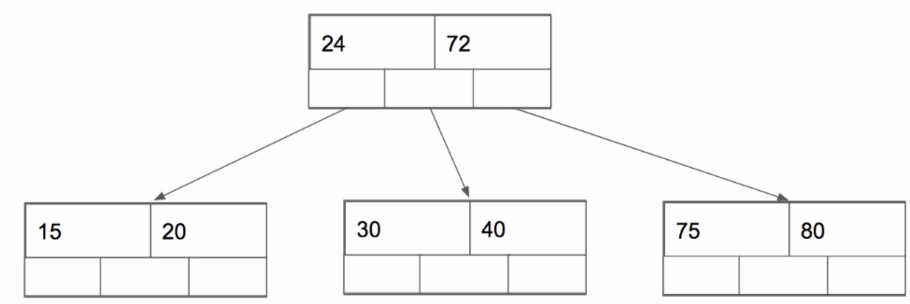
```
class BST {
    int label;
    BST left; // null if no left child
    BST right; // null if no right child
}

public BST pruneBST(BST root, int L, int R) {
    if (_______________) {
        return ________;
    } else if (____________________) {
        return pruneBST(____________, _____, _____);
    } else if (____________________) {
        return pruneBST(____________, _____, _____);
    }
    ____________ = pruneBST(____________, _____, _____);
    ____________ = pruneBST(____________, _____, _____);
    return _______;
}
```

4 All about Trees

1. Why does a binary search tree have a worst case runtime of $\theta(n)$ for *contains*?

2. Give a sequence of operations, such that if they were inserted in the order they appear, would result in a "poor" binary search tree.

3. Examine this B-tree with order 3. Mark the paths taken when the user calls $contains(40)$.



4. Now call $insert(35)$, and draw the resulting tree.

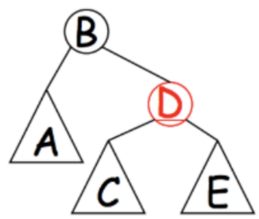
5. What property of a B-tree rectifies problems of binary search trees, such as the one in 1.1? Why would you not use a B-tree?

5 The Holy LLRB Invariant

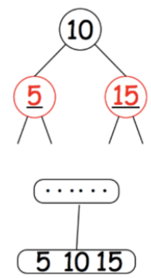
RB Tree Invariants: Node labels are in order from left to right. All paths through the tree contain the same number of black nodes. No red nodes have red parents. As a result, the height of a RB tree with n nodes is $O(logn)$.

LLRB trees must also maintain the following invariant (in addition to the regular red-black invariant):

No right-leaning trees (black parent with right red child):



No “4-nodes” (black parent with two red children):



1. What are the ”fixups” for the two cases above in order to preserve the LLRB invariant (i.e. what operations do we perform on each tree to ensure it is a proper LLRB)?

Consider the following RB tree:

2. Draw the tree after applying all necessary fixups to make it a proper LLRB tree.

3. Next, insert 10 into the tree, and apply all fixups to preserve the LLRB invariant.

4. Finally, draw the corresponding 2-3 tree.