CS 61B Spring 2026

BSTs, 2-3 Trees, LLRBs

Exam-Level Discussion 08: March 9, 2026

1 Is this a BST?

In this setup, assume a **BST** (Binary Search Tree) has a **key** (the value of the tree root represented as an **int**) and pointers to two other child BSTs, **left** and **right**. Additionally, assume that **key** is between **Integer.MIN_VALUE** and **Integer.MAX_VALUE** non-inclusive.

(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(BST tree) {
    if (tree == null) {
        return true;
    } else if (tree.left != null && tree.left.key >= tree.key) {
        return false;
    } else if (tree.right != null && tree.right.key <= tree.key) {
        return false;
    } else {
        return brokenIsBST(tree.left) && brokenIsBST(tree.right);
    }
}
```

(b) Now, write **isBST** that fixes the error encountered in part (a).

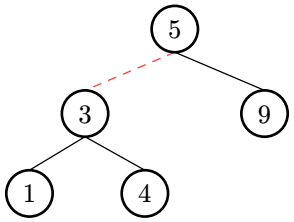
Hint: You will find **Integer.MIN_VALUE** and **Integer.MAX_VALUE** helpful.

Hint 2: You want to somehow store information about the keys from previous layers, not just the direct parent and children. How do you use the parameters given to do this?

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

public static boolean isBSTHelper(BST T, int min, int max) {

    if (_____) {

        _____;

    } else if (_____) {

        _____;

    } else {

        _____;

    }
}
```

2 LLRB Insertions

Given the LLRB below, perform the following insertions and draw the final state of the LLRB. In addition, for each insertion, write the balancing operations needed in the correct order (**`rotateRight`**, **`rotateLeft`**, or **`colorFlip`**). If no balancing operations are needed, write "Nothing". Assume that the link between 5 and 3 is red and all other links are black at the start.



(a) 1. Insert 7

2. Insert 6

3. Insert 2

4. Insert 8

5. Insert 8.5

6. Final state

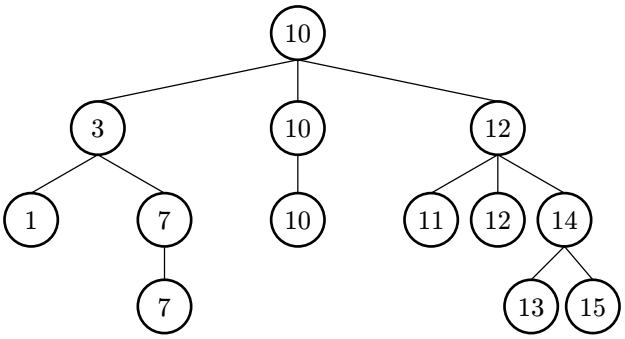
(b) Convert the final LLRB to its corresponding 2-3 Tree.

3 Tri-nary Search Tree

We'd like a data structure that acts like a BST (Binary Search Tree) in terms of operation runtimes but allows duplicate values. Therefore, we decide to create a new data structure called a TST (Trinary Search Tree), which can have up to three children, which we'll refer to as `left`, `middle`, and `right`. In this setup, we have the following invariants, which are very similar to the BST invariants:

1. Each node in a TST is a root of a smaller TST
2. Every node to the `left` of a root has a value "lesser than" that of the root
3. Every node to the `right` of a root has a value "greater than" that of the root
4. **Every node to the `middle` of a root has a value equal to that of the root**

Below is an example TST to help with visualization.



Describe an algorithm that will print the elements in a TST in **descending** order. *(Hint: recall that an in-order traversal for a BST gives elements in increasing order.)*