CSM 61B

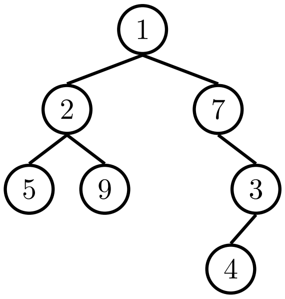
Fall 2017

Binary Trees

Mentoring 8: October 16, 2017

1 Binary Trees

```
public class BinaryTree<T> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
}
```



1.1 Define a procedure, `height`, which takes in a `Node` and outputs the height of the tree. Recall that the height of a leaf node is 0.

```
private int height(Node node) {









}
```

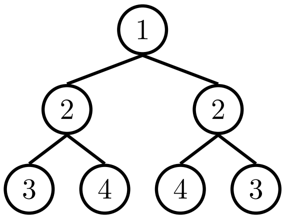
What is the runtime of `height`?

1.2 Define a procedure, `isBalanced`, which takes a `Node` and outputs whether or not the tree is balanced. A tree is **balanced** if the left and right branches differ in height by at most one and are themselves balanced.

```
private boolean isBalanced(Node node) {









}
```

What is the runtime of `isBalanced`?

1.3 Define `isSymmetric` which checks whether the binary tree is a mirror of itself.



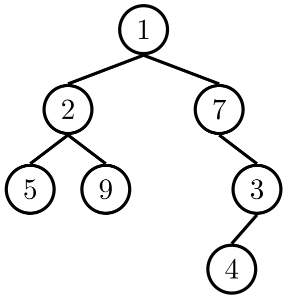
```
public boolean isSymmetric() {
    if (root == null) {
        return true;
    }
    return isSymmetric(root.left, root.right); // use helper method
}

private boolean isSymmetric(Node left, Node right) {









}
```

2 Binary Search Trees

2.1 Implement `fromSortedArray` for binary search trees. Given a sorted `int[] array`, efficiently construct a balanced binary search tree containing every element of the array.

```
public class BinarySearchTree<T extends Comparable<T>> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
    public static BinarySearchTree<Integer> fromSortedArray(int[] values) {
        BinarySearchTree<Integer> bst = new BinarySearchTree<>();
        bst.root = bst.fromSortedArray(values, 0, values.length - 1);
        return bst;
    }
    private Node fromSortedArray(int[] values, int lower, int upper) {
```

3 Successor *Extra Practice*



Level-Order Traversals Nodes are visited top-to-bottom, left-to-right.

Depth-First Traversals Visit deep nodes before shallow ones.

3.1 Give the ordering for each depth-first traversal of the tree.

(a) Pre-order

(b) In-order

(c) Post-order

3.2 Give the level-order traversal of the tree.

3.3 Given a node in a binary search tree (with parent pointers), implement `successor` which returns the next node in the in-order traversal of the BST. If there is no successor, return **`null`**.

```
public class BinarySearchTree<T extends Comparable<T>> {
    protected Node root;
    protected class Node {
        public T value;
        public Node parent, left, right;
    }
    private Node successor(Node node) {
```