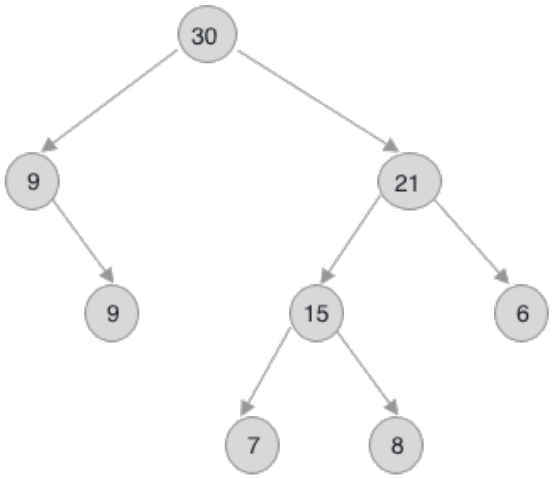
CS 61B Exam Prep 9: Trees Spring 2020

1 Binary Trees

For each of the following questions, use the following `Tree` class for reference -

```
public class Tree {
    public Tree(Tree left, int value, Tree right) {
        _left = left;
        _value = value;
        _right = right;
    }
    public Tree(int value) {
        this(null, value, null);
    }
    public int value() {
        return _value;
    }
    public Tree leftChild() {
        return _left;
    }
    public Tree rightChild() {
        return _right;
    }
    private int _value;
    private Tree _left, _right;
}
```

(a) Given a binary tree, check if it is a sum tree or not. In a sum tree, value at each non-leaf node is equal to the sum of all elements presents in its left and right subtree. For example, the following binary tree is a sum tree -



```
public boolean isSumTree(Tree t) {
        if (t == null || (t.leftChild()==null && t.rightChild()==null)) {
            return true;
        }

        int left = t.leftChild() == null ? 0 : t.leftChild().value();
        int right = t.rightChild() == null ? 0 : t.rightChild().value();

        return (t.value() == left + right) &&
                isSumTree(t.leftChild()) &&
                isSumTree(t.rightChild());
    }
```

(b) Given a binary tree with distinct items, an input list, and an empty output list, add all the elements in the input list to the output list that appear in the tree. The elements in the output list should be ordered in the same order that would be returned from an inorder traversal.

For example, for the tree in Q.1(a) assuming it has only distinct items, if the input list is [15, 9, 8, 30, 6], then after the operation the output list should be [9, 30, 15, 8, 6].

```
public static void sortRelative(Tree t,
                                List<Integer> inputList,
                                List<Integer> outputList) {
    if (t != null) {
        sortRelative(t.leftChild(), inputList, outputList);
        if (inputList.contains(t.value())) {
            outputList.add(t.value());
        }
        sortRelative(t.rightChild(), inputList, outputList);
    }
}
```

(c) Given a Tree t and two integers x and y, return the lowest common ancestor of x and y in t. Assume all labels in t are nonnegative, nonzero, and distinct, and that x and y are in t. The behavior of the function can be undefined if the above does not hold.

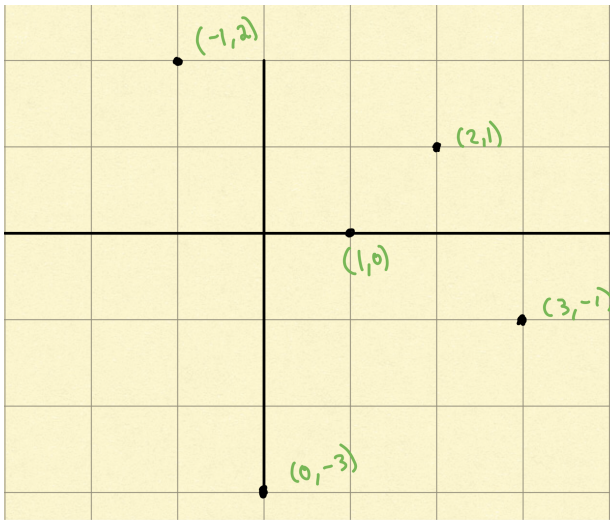
For example, for the tree in Q.1(a), the lowest common ancestor for node 6 and node 7 will be node 21.

```
public static int commonAncestor(Tree t, int x, int y) {
    assert x != y;
    if (t == null) {
        return 0;
    }
    boolean here = (t.value() == x) || (t.value() == y);
    int a = commonAncestor(t.leftChild(), x, y);
    int b = commonAncestor(t.rightChild(), x, y);
    if ((a == -1 && b == -1) || ((a == -1 || b == -1) && here)) {
        //The current node is the common ancestor if
        // 1. x and y are located on different branches
        // 2. one of x or y is the current label and the other is farther
           down the tree
        return t.value();
    } else if (here) {
```

```
            // We "flag" this node by sending -1 up the recursive calls
            return -1;
        } else {
            // If the node is not the common ancestor nor contains x/y as a
                label,
            // we can just pass up values from recursive calls as are
            return a + b;
        }
    }
```

2 Quad Trees

Assuming we are starting with an empty quad tree, using the points below, what order of insertion produces the quad tree with -



(a) Maximum Height

`(-1, 2) -> (0, -3) -> (3, -1) -> (1, 0) -> (2, 1)`

`OR`

`(-1, 2) -> (0, -3) -> (3, -1) -> (2, 1) -> (1, 0)`

`The ordering of the last two doesn't matter!`

(b) Minimum Height

`Insert (1, 0) first. The insertion order of the rest doesn't matter!`

3 Challenge

Given five points along the line y = x and an initially empty quad tree, how many different orders of insertion produce a quad tree of maximum height?

Now instead of five points, what if we were given n points?

16 different orderings, i.e. $2^{(n-1)}$.

Observe that given n points on the line y = x, we can first insert the one with the smallest OR largest x/y value to create the max-tree of maximum height. This is because after inserting the smallest value, the other points are all above and to the right (same idea **for** largest value). Then, after inserting the first/last values, we just need to insert the rest of the n - 1 values, and we develop the recursive relationship: $f(n) = 2 * f(n-1)$, where $f(n)$ is the number of ways of inserting n values, and the 2 comes from the 2 ways of choosing the first point to insert. Adding in the base **case** $f(1) = 1$, we see that - $f(n) = 2^{(n-1)}$ and $f(5) = 16$.