Homework 11 – Sample Solution

1. (10 points)
   Write Java-like code for the method rightRotate that performs a right rotation about the specified node. You do not need to include precondition checks.

```
/**
 * BinaryTree
 *
 * @author CS3151
 * @param <T> type of the node values
 */
public class BinaryTree<T> {
    private BinaryNode root;

    /**
     * Instantiates a new empty binary tree
     */
    public BinaryTree() {
        this.root = null;
    }
    . . .

    /**
     * Performs a right-rotation about the specified node.
     *
     * @precondition node is a node in this binary tree && node has a left child
     * @param node the node that is right rotated
     */
    private void rightRotate(BinaryNode node) {
        BinaryNode leftChild = node.left;
        if (node.parent != null) {
            if (node == node.parent.left) {
                node.parent.left = leftChild;
            } else {
                node.parent.right = leftChild;
            }
        } else {
            this.root = leftChild;
        }
        leftChild.parent = node.parent;

        BinaryNode rightOfLeftChild = leftChild.right;
        node.left = rightOfLeftChild;
        if (rightOfLeftChild != null) {
            rightOfLeftChild.parent = node;
        }
        node.parent = leftChild;
        leftChild.right = node;
    }
```

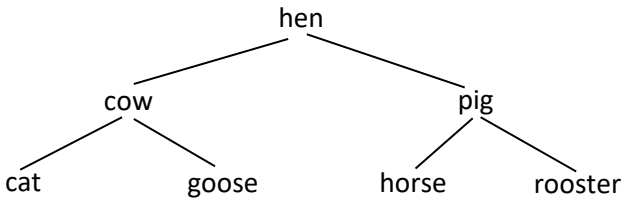
```
    /**
     * Class BinaryNode
     *
     * @author CS3151
     */
    protected final class BinaryNode {
        private T value;
        private BinaryNode parent;
        private BinaryNode left;
        private BinaryNode right;

        private BinaryNode(T value) {
            this.value = value;
            this.parent = null;
            this.left = null;
            this.right = null;
        }
    }
}
```

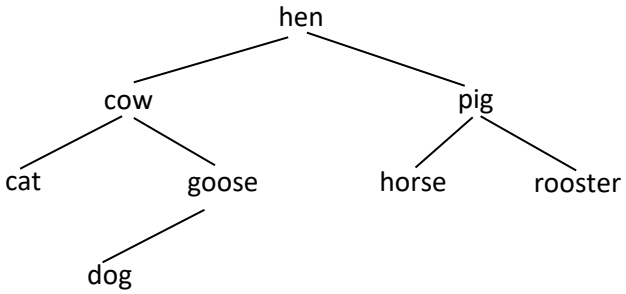
2. (12 points)

Draw the resulting AVL tree after each of the specified insert operations. You need to use the insert operation as covered in the class. The nodes should be sorted by the lexicographical order.

a) Insert “dog” into the following AVL tree:

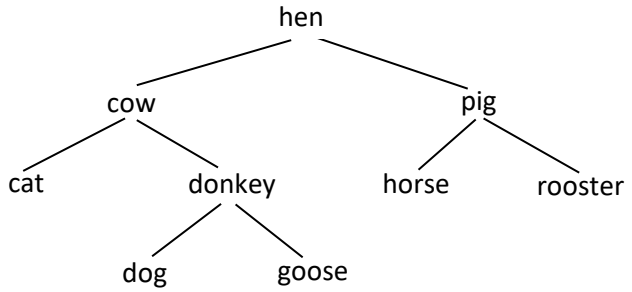


In the first step of the insert, dog is added as a leave node using the regular binary search tree insert operation:

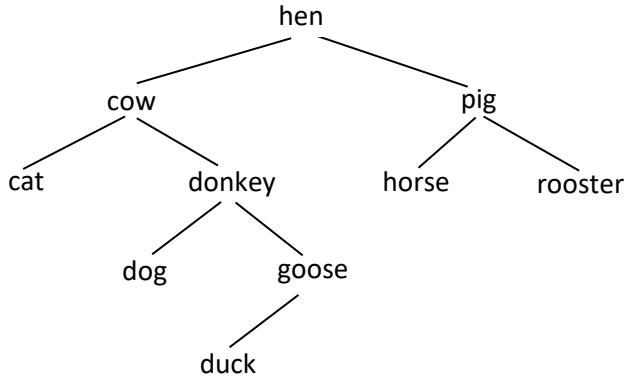


An update of the balance factors shows that the tree is still balanced, and therefore no rotations need to be performed.

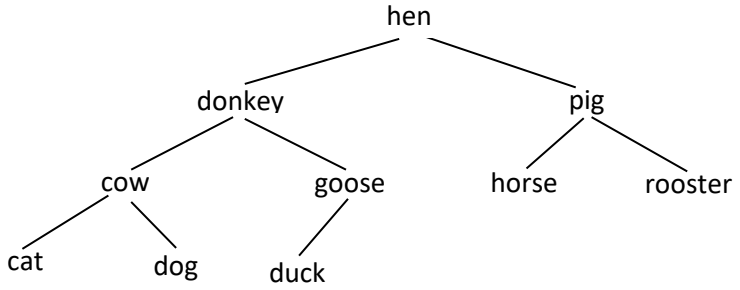
b) Insert “duck” into the following AVL tree:



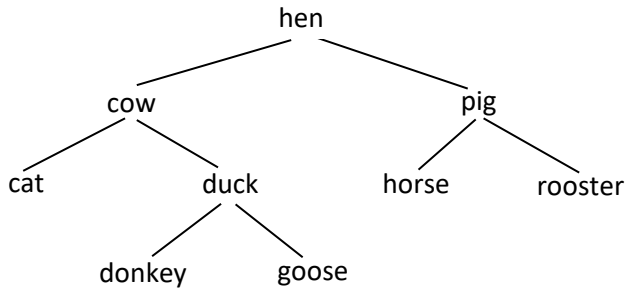
In the first step of the insert, duck is added as a leave node using the regular binary search tree insert operation:



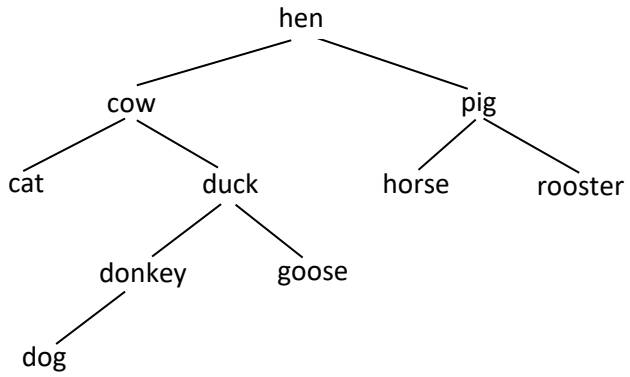
Starting at duck and traversing along the parents, the balance-factors are updated as follows: duck – 0, goose – -1, donkey – 1, cow – 2. Since the balance-factor of cow is 2, a left-rotation about cow will need to be performed. Possibly, a right-rotation about the right child of cow needs to be performed first. However, in this case the balance-factor of donkey is not -1, and therefore no right-rotation is necessary. The final tree after the left-rotation about cow is:



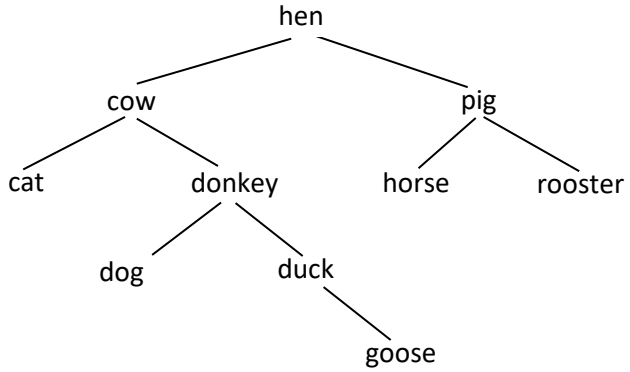
c) Insert “dog” into the following AVL tree:



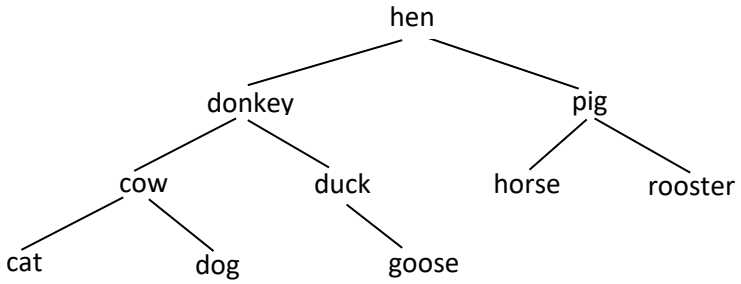
In the first step of the insert, dog is added as a leave node using the regular binary search tree insert operation:



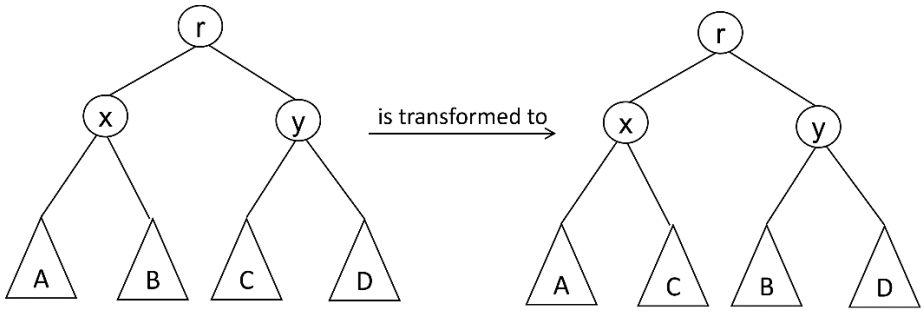
Starting at dog and traversing along the parents, the balance-factors are updated as follows: dog – 0, donkey – -1, duck – -1, cow – 2. Since the balance-factor of cow is 2, a left-rotation about cow will need to be performed. Possibly, a right-rotation about the right child of cow needs to be performed first. Indeed, the balance-factor of duck is -1, and therefore a right-rotation about duck is necessary resulting in the following tree:



The final tree after the left-rotation about cow is:

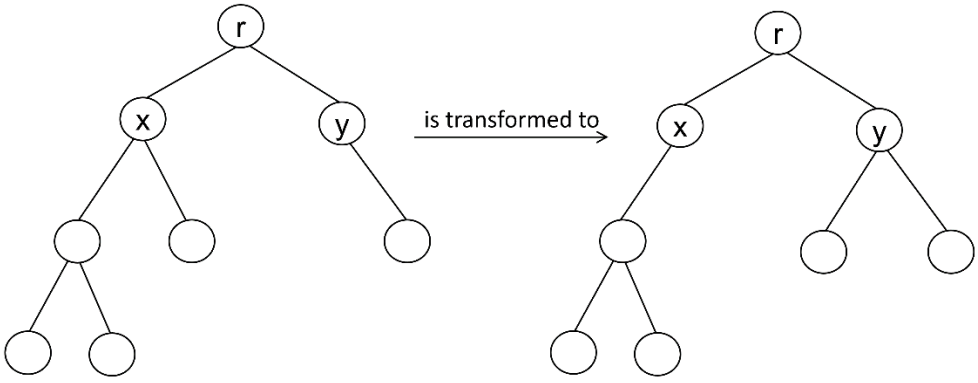


3. (20 points)
   a) Swap subtrees B and C:



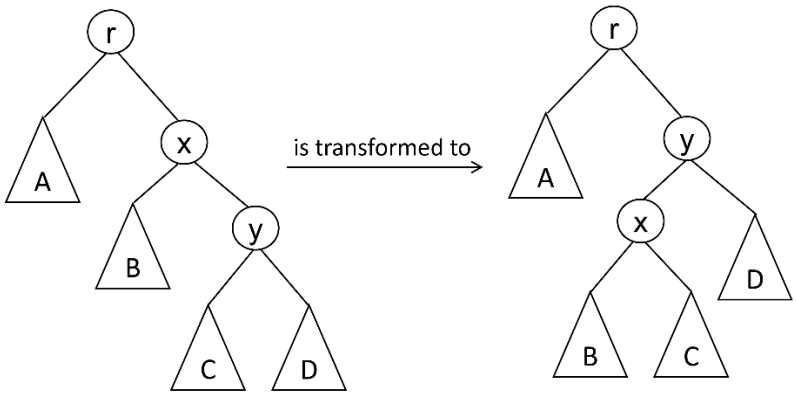
Does the operation maintain the binary search tree property? **No, the nodes in C are greater than node r and therefore cannot be located in the left subtree of r.**

Does the operation maintain the height-balance? **No, here is an example where the height-balanced is not maintained:**



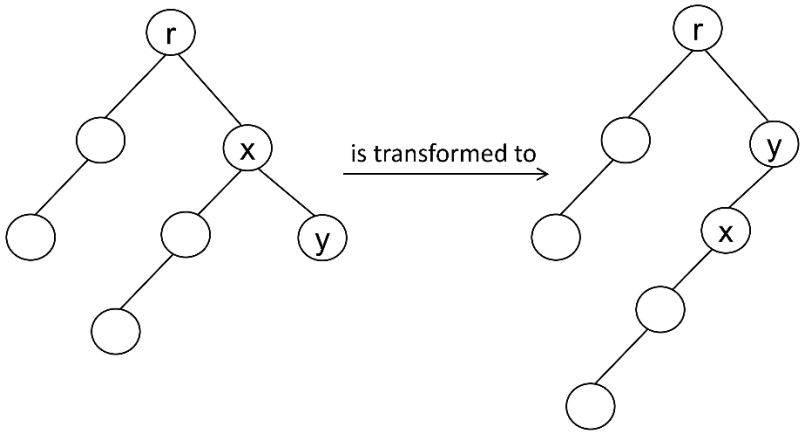
The original tree is height-balanced. After the transformation, the heights of the subtrees of x differ by two in the resulting tree. Thus, the resulting tree is not height-balanced.

b) Move x and y and subtree C:



Does the operation maintain the binary search tree property? **Yes, the transformation is a left-rotation about x. We know that the rotation operations maintain the binary search tree property.**

Does the operation maintain the height-balance? **No, here is an example where the height-balanced is not maintained:**



4. (8 points)

Event names in the order the corresponding events are removed:

5k race, art festival, dinner party, fireworks, concert, conference, wedding