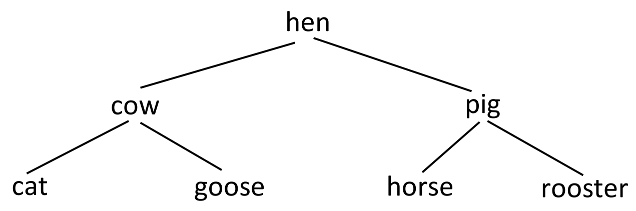
Homework 11

Due date: Apr 23, 2020, 9:30am

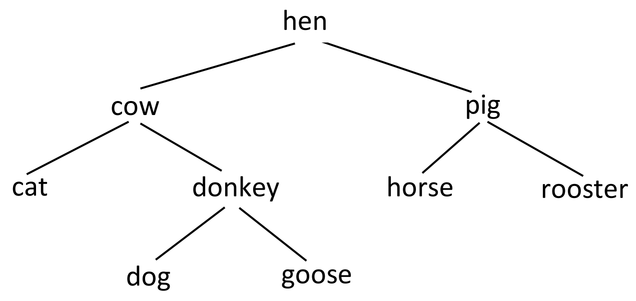
1. (10 points)
   Write Java-like code for the method rightRotate that performs a right rotation about the specified node. You do not need to include precondition checks.

```
/**
 * 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) {




    }
    /**
     * 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;
        }
    }
}
```

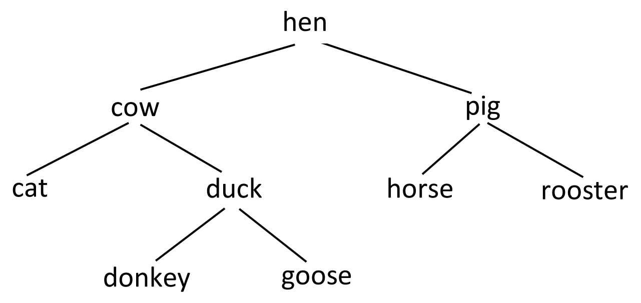
2. (12 points)
Draw the resulting AVL tree after each of the specified insert operations. You need to use the insert operation as covered in the class. The nodes should be sorted by the lexicographical order.
a) Insert "dog" into the following AVL tree:



b) Insert "duck" into the following AVL tree:

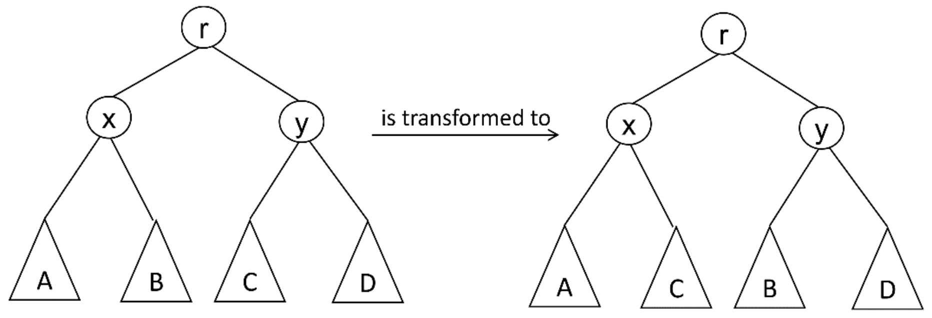


c) Insert "dog" into the following AVL tree:



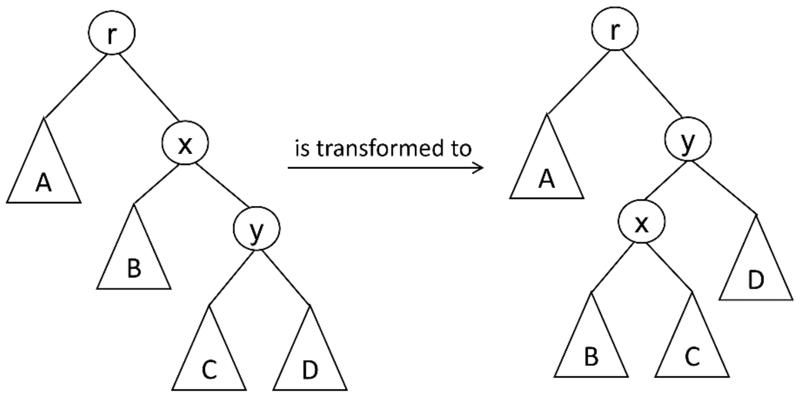
3. (20 points)
For each of the operation below, determine whether it maintains the binary search tree property and the height-balance. The binary search tree property is maintained, if the tree is still a binary search tree after the operation in case the tree is a binary search tree at the beginning of the operation. Analogously, the height-balance is maintained, if the tree is still the height-balanced after the operation in case the tree is height-balanced at the beginning of the operation.
Lowercase letters are node labels, capital letters represent subtrees. A subtree may be an empty tree. The labels are unrelated to the order on the node values.

a) Swap subtrees B and C:



Does the operation maintain the binary search tree property? Yes / No
Does the operation maintain the height-balance? Yes / No

b) Move x and y and subtree C:



Does the operation maintain the binary search tree property? Yes / No

Does the operation maintain the height-balance? Yes / No

4. (8 points)

Consider the following Java-like code below that performs a sequence of operations on the initially empty priority queue `events`. Each `Event` object maintains a timestamp and an event name that are passed in to the Event constructor. The smaller the timestamp, the higher the priority of an event. Determine the order in which the events are removed from the priority queue. (List the names of the removed events.)

```
PriorityQueue<Event> events = new PriorityQueue<Event>();
Event event = null;
events.add(new Event(28, “movie”));
events.add(new Event(18, “5k race”));
events.add(new Event(22, “art festival”));
event = events.remove();
event = events.remove();
events.add(new Event(14, “dinner party”));
events.add(new Event(15, “fireworks”));
events.add(new Event(17, “concert”));
event = events.remove();
event = events.remove();
events.add(new Event(24, “wedding”));
events.add(new Event(23, “conference”));
event = events.remove();
event = events.remove();
event = events.remove();
```

Event names in the order the corresponding events are removed:

Submission

Submit a single PDF file (preferred) or a single MS Word document with your solutions. No other file formats are accepted. If you prefer to write (or draw) your solution by hand and you do not have a scanner, take pictures of your hand-written solutions and imbed the pictures in a Word document.