# 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) {

if (node.left == null) {

throw new IllegalArgumentException("node must have a left");

}

if (!this.nodeInTree(node, this.root)) {

throw new IllegalArgumentException("node is not in tree");

}

BinaryNode parent = node.parent;

BinaryNode x = node.left;

BinaryNode a = node.left.left;

BinaryNode z = node;

BinaryNode b = node.left.right;

BinaryNode c = node.right.right;

x.parent = parent;

node = x;

z.parent = node;

z.left = b;

z.right = c;

node.left = a;

node.right = z;

private boolean nodeInTree (BinaryNode node, BinaryNode root) {

if (node.parent == root) {

return true;

}

if (node.parent == null) {

return false;

}

this.nodeInTree(node.parent, root);

}

/\*\*

\* 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**;

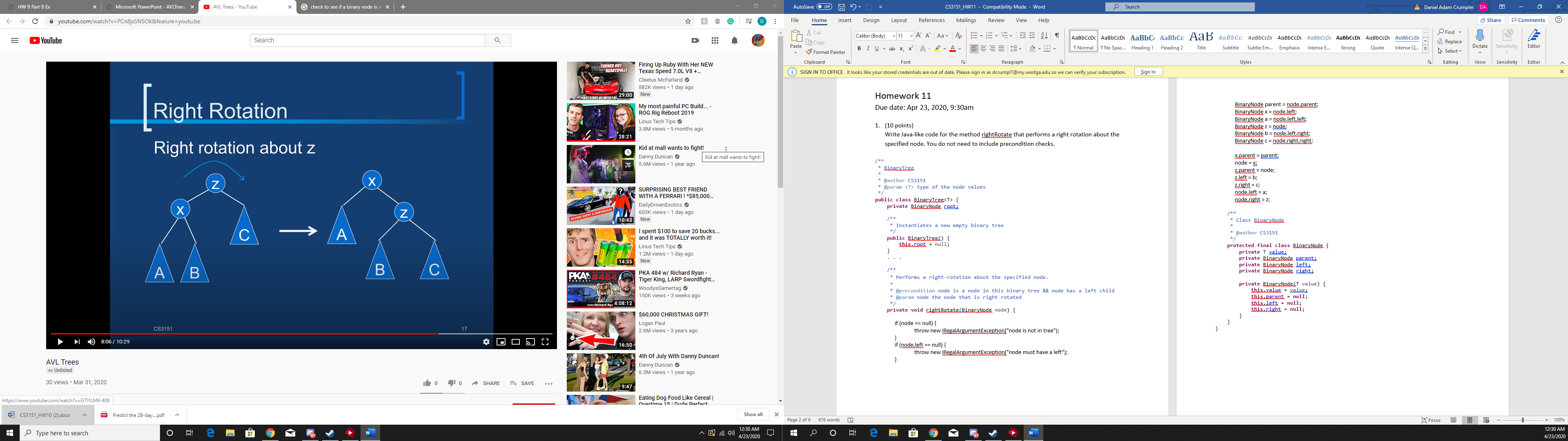
**this**.left = **null**;

**this**.right = **null**;

}

}

}



1. (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.

1. Insert “dog” into the following AVL tree:

hen

cow

pig

goose

rooster

horse

cat

dog

1. Insert “duck” into the following AVL tree:

hen

donkey

pig

cow

goose

rooster

horse

cat

duck

dog

1. Insert “dog” into the following AVL tree:

hen

duck

pig

goose

cow

rooster

horse

cat

donkey

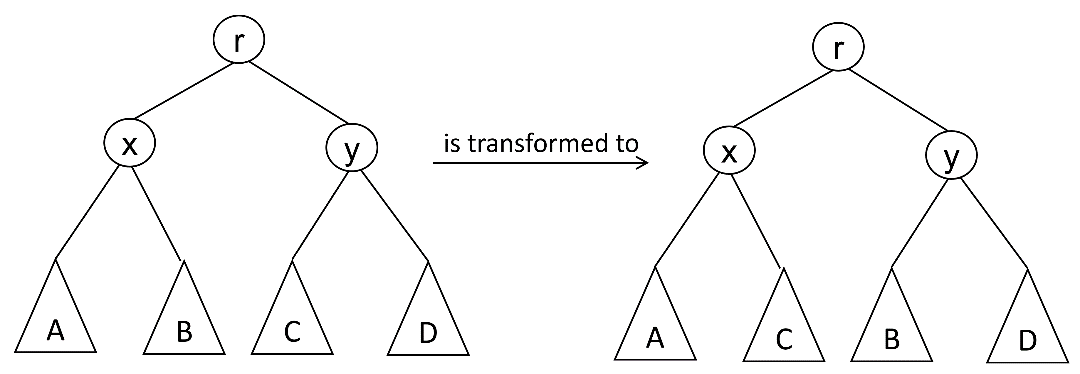
dog

1. (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.

1. Swap subtrees B and C:



Does the operation maintain the binary search tree property? **Yes**

Does the operation maintain the height-balance? **Yes**

1. Move x and y and subtree C:

A close up of a map

Description automatically generated

Does the operation maintain the binary search tree property? **Yes**

Does the operation maintain the height-balance?  **No**

1. (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”)); **(movie)**

events.add(new Event(18, “5k race”)); **(5k race, movie)**

events.add(new Event(22, “art festival”)); **(5k race, art festival, movie)**

event = events.remove(); **(art festival, movie)**

event = events.remove(); **(movie)**

events.add(new Event(14, “dinner party”)); **(dinner party, movie)**

events.add(new Event(15, “fireworks”)); **(dinner party, fireworks, movie)**

events.add(new Event(17, “concert”)); **(dinner party, fireworks, concert, movie)**

event = events.remove(); **(fireworks, concert, movie)**

event = events.remove(); **(concert, movie)**

events.add(new Event(24, “wedding”)); **(concert, wedding, movie)**

events.add(new Event(23, “conference”)); **(concert, conference, wedding, movie)**

event = events.remove(); **(conference, wedding, movie)**

event = events.remove(); **(wedding, movie)**

event = events.remove(); **(movie)**

Event names in the order the corresponding events are removed:

**5k Race**

**Art Festival**

**Dinner Party**

**Fireworks**

**Concert**

**Conference**

**Wedding**

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.