CSSE 230 – Data Structures and Algorithms

Exam 2 Winter, 2016-17 Name: _____KEY_____ **Section: 01 02**

You may use a calculator, a writing implement, and a 1-sided 8.5" x 11" note page. You must turn in this written part before you use your computer, textbook, or any other resources.

You may not communicate with any person other than your instructor about this exam until allowed by your instructor. You may not use other electronic devices such as phones or any device with headphones or earbuds.

Part 1 scores (for instructor use):

Problem	Possible	Score
1	18	
2	6	
3	12	
4	8	
5	6	
6	15	
Total	**65**	

Computer part = 55 points

Total	120	

Part 2 (computer) rules

You may use any printed or written materials that you brought with you, code that you wrote yourself or with a partner before the exam, anything on the CSSE230 Moodle and web sites, and the Java API documentation (on your laptop or at oracle.com).

You may not otherwise search the internet or communicate with any person other than your instructor or a student assistant. You may not communicate with anyone else about this exam until allowed by your instructor. You may not use other electronic devices such as phones or any device with headphones or earbuds.

You will sign a statement to certify that you neither received help from others nor gave it on either part of the exam.

You must actually get this problem working on your computer. All or most of the credit for this problem will be for code that actually works.

1. (18 points) T/F/IDK. Below you will find several statements. A statement is true (T) if it is always true. It is false (F) if there is at least one counterexample (sometimes false). You may also choose IDK to indicate that you do not know the answer. **Point values:** Correct answer: 2, incorrect answer: -1, IDK: 1, blank: 0. **Circle at most one answer for each part.**

   a. (T) F IDK Some red-black trees are not height-balanced.

   b. T (F) IDK In an AVL tree where the root has balance code Code.RIGHT, inserting something smaller than root.element must either cause a rotation or cause the root's balance code to shift to Code.SAME.

   c. T (F) IDK In a postorder traversal of a binary search tree, the final node visited has the smallest value in the tree.

   d. (T) F IDK A hash table with linear probing and a binary heap are both examples of array-based data structures.

   e. (T) F IDK If a tree satisfies both BST and (min)-heap-order properties, then it is a degenerate tree (that is, a linked list).

   f. T (F) IDK In a binary tree, a preorder iterator can be implemented with a Stack; replacing the Stack data structure and methods with those of a Queue will turn it into an in-order iterator.

   g. T (F) IDK A hash table can simultaneously achieve worst-case O(1) lookup and worst-case O(1) insertion.

   h. (T) F IDK An unsorted array of size n can be turned into a heap in O(n) time.

   i. T (F) IDK A binary search tree achieves worst-case O(log n) insertion, deletion, and search.

2. (6 points) Match up each hashing collision-handling method
   A. Linear probing
   B. Quadratic probing
   C. Chaining

   with the true statement about it when the table has load factor in the range $0.5 < \lambda < 1.0$.

   i. insert() will work, efficiently. ___**C**___

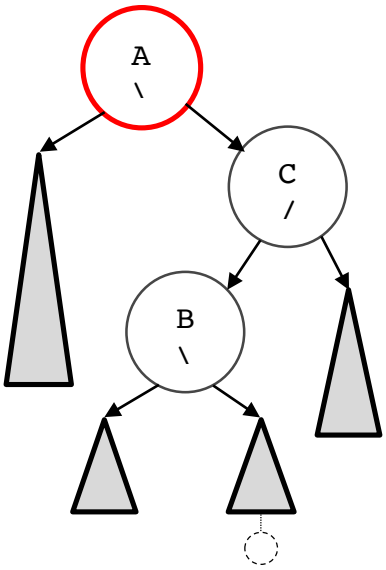
   ii. insert() may not work at all. ___**B**___

   iii. insert() will work, but is very inefficient. ___**A**___

3. (12 points) EditorTree problem: balance and rank updates after a rotation.
Consider a node A, with C = A.right, and B = C. left, as shown.
Suppose we **insert** a node in the **right subtree of B**, and the lowest imbalance this causes is at A, forcing a double-left rotation at A.
The picture shows the "mid-insertion" situation on our way back up the tree: we have already updated B's and C's balance codes and ranks to reflect the insertion, but upon reaching A we found the balance code \ could not be shifted further right, so need to rotate.

Complete the table by writing the new ranks (in terms of R_A, R_B, R_C) and balance codes, after rotation.

Mid-insertion:
On the way up the tree.
Just found imbalance at A.
(Subtrees not drawn to scale)



Node	Mid-insertion rank (just before rotation)	Rank after rotation	Mid-insertion balance code	New balance code
A	R_A	R_A	\	/
B	R_B	$R_A + R_B + 1$	\	=
C	R_C	$R_C - R_B - 1$	/	=

4. (8 points) Short answer.

   a. Given a heap of size *n* represented by an array `arr` with the 0th entry unused (so the root is at `arr[1]`), what is the index of the first (lowest-index) leaf? [Be precise—you may need to use a mathematical floor/ceiling.]

floor(*n* / 2)+1

   b. Suppose we insert the following names of people currently aged 31 into a hash table, using Java's String.hashCode() method, where the underlying array is of capacity 31.

`"Cristiano Ronaldo"`	`"Keira Knightly"`
`"Michael Phelps"`	`"Anna Kendrick"`
`"Allyson Felix"`	`"Matt Ryan"`
`"Bruno Mars"`	`"Chris Paul"`
`"Carly Rae Jepson"`	`"Nate Chenette"`

   Name a pair of people who collide in the hash table.

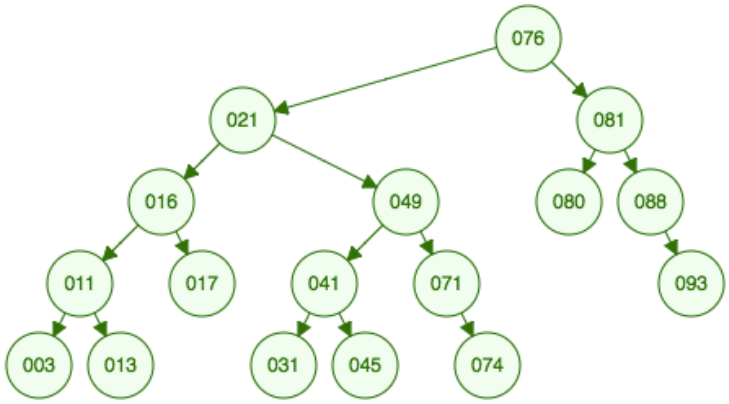
_____ collides with _____

Phelps & Mars, or Jepson & Ryan

5. (6 points) A friend of yours has invented new "self-balancing" rules for a binary search tree—she calls them Low Vacancy Always (LVA) trees. Fascinated, you power through an induction proof and find that the minimum-size LVA tree of height $H \geq 2$ has size $N = H^2$. Given this result, what guarantee can you give on the running time of the contains() operation on an LVA tree of size N?

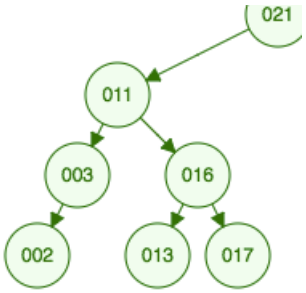
contains() should be O(height). The result shows that if the height is H, then $N \geq H^2$, implying that $H \leq \sqrt{N}$. Therefore, contains will be O(sqrt(N)).

6. (15 points) Consider the following AVL tree. **All parts will all start with this original tree**. For each answer, draw the tree that results after the given node is inserted and the tree is rebalanced. If only a small part of the tree is affected by the rotation and you clearly show all the effects of the rotation, you don't have to re-draw the whole tree. Also write which type of rotation is needed (like **SL@50**).
-2 for minor error, -4 for major one. 1 point for rotation type.

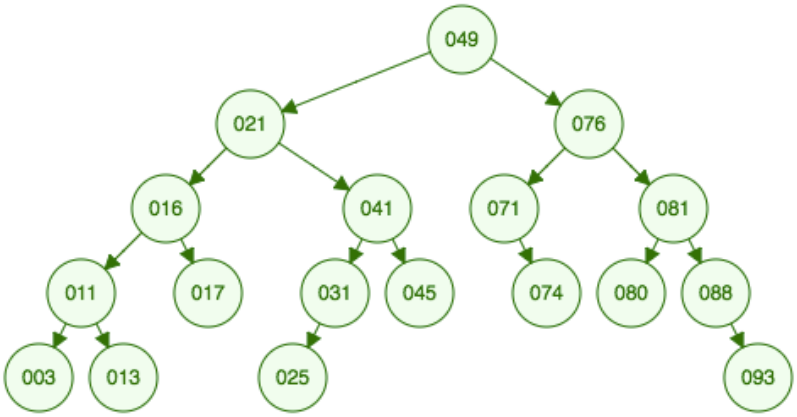


a. (5 pts) Insert 72 into the original tree and rebalance. Rotation: ____ **DL@71**_____

b. (5 pts) Insert 2 into the original tree and rebalance. Rotation: ____**SR@16**_____



c. (5 pts) Insert 25 into the original tree and rebalance. Rotation: ____**DR@76**_____



Programming:

1) In BinarySearchTree:

```
public int indexOf(Integer value) {
	return this.root.indexOf(value);
}
```

In BinaryNode:

```
public int indexOf(Integer value) {
	if (this == NULL_NODE) {
		throw new NoSuchElementException();
	}
	if (this.data.compareTo(value) > 0) {         // value is to the left
		return left.indexOf(value);
	} else if (this.data.compareTo(value) < 0) {  // value is to the right
		return (left.size() + 1 + right.indexOf(value));
	} else {                                      // found value
		return left.size();
	}
}

public int size() {
	if (this == NULL_NODE) {
		return 0;
	}
	return (left.size() + right.size() + 1);
}
```

2) In BinarySearchTreeWithColor:

```
public boolean isRedBlack() {
	boolean rootIsBlack = (root.color == Color.BLACK);
	Info info = root.isRedBlackHelper();
	return (rootIsBlack && info.isRedBlackSubtree);
}

public class Info {
	boolean isRedBlackSubtree; // true if subtree satisfies no-double-reds and black-balance
	int blackHeight;
	Color color;
	public Info(boolean isRBS, int blackHt, Color col) {
		this.isRedBlackSubtree = isRBS;
		this.blackHeight = blackHt;
		this.color = col;
	}
}
```

In RedBlackNode:

```
public Info isRedBlackHelper() {
	if (this == NULL_NODE) {
		return new Info(true,-1,Color.BLACK);
	}
	Info lInfo = left.isRedBlackHelper();
	Info rInfo = right.isRedBlackHelper();
	boolean isRBSubtree = lInfo.isRedBlackSubtree && rInfo.isRedBlackSubtree &&
			(this.color == Color.BLACK ||
			 (lInfo.color == Color.BLACK && rInfo.color == Color.BLACK)) &&
			lInfo.blackHeight == rInfo.blackHeight;
	int bHt = lInfo.blackHeight;
	bHt += (this.color == Color.BLACK) ? 1 : 0;
	return new Info(isRBSubtree,bHt,this.color);
}
```