CSSE 230 – Data Structures and Algorithm Analysis

Exam 2

Spring term, 2022-2023

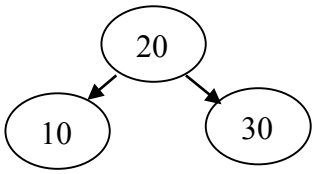
Your name: _____

Instructions:

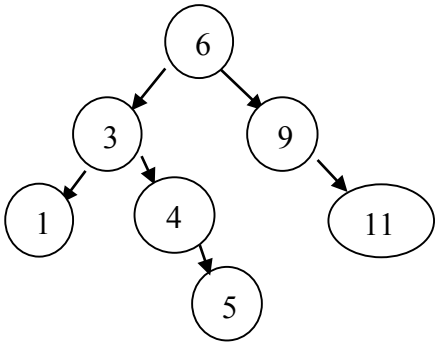
- All the work you turn in must be your own.
- You must not use any forms of communication or cooperation.
- This is an open book exam:
    - For any portion, you may look at the book whether hardcopy or online.
    - You may look at our CSSE 230 web-site and any page linked directly from it.
    - You may not search the web and you may not use ChatGPT or like technologies during the exam.
    - You may look at any of the Java documentation, in particular that for data structures.
    - For the paper part, you may NOT run any applets nor any code.
    - For the on-the-computer part, you may NOT copy any of the code on your computer.

Problem	Points	Your score
1	**12**	
2	**18**	
3	**12**	
4	**8**	
reverse()	**25**	
isAVLTree()	**25**	
Total	**100**	

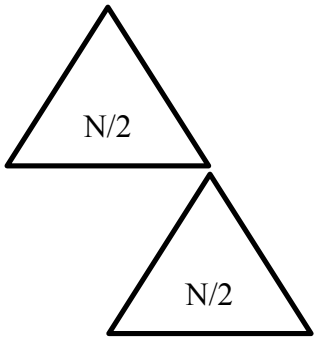
1) a) [6 pts] Show the steps and rotations required to insert the following elements into the AVL tree below: 25, 27, 22, 26 in this order.



b) [6 pts] Show the steps and rotations required to remove nodes 3, 9, 11 in that order, from the AVL tree below. If you need to remove a node with two children, please copy the largest value of its left subtree.



2) Consider the following data structure. Instead of inserting *N* elements into an AVL tree, insert just *N/2* elements each into two separate AVL trees. Once finished, combine both AVL trees by connecting the root of one of the AVL trees to the right child of the largest element of the other AVL tree. Here is an image of the proposed data structure.



a) [2 pts] What is the fundamental flaw of the proposed data structure?

b) [8 pts] Consider the following alternate proposal. Again, create two AVL trees with *N/2* elements each. Then, merge the trees by inserting the elements of the second tree into the first one, using the AVL tree `insert` method. Analyze the **worst-case** runtime of this algorithm. Please express your answer in terms of *N*, the number of elements to be inserted into the data structure. Consider the:
- cost of building the initial two trees, the
- cost of reading off the elements from one of the trees and then the
- cost of inserting those elements into the other tree.

Express your answer using approximate data or using big-Oh notation. Explain your answers.

c) [8 pts] Suppose, we are using a BST instead of an AVL tree. What is the **worst-case** runtime of the data structure proposed in (b), given a BST? Again, justify your answer.

3) Consider the following method from the **`BinarySearchTree`** class. Its objective is to take a BinarySearch tree and balance it, using an AVL tree.

```
public void balanceTree(){
a:___ AVLTree<T> t = new AVLTree<T>();
b:___ Iterator<T> i = this.iterator();
c:___ while (i.hasNext()) t.insert(i.next());
d:___ this.root = null;
e:___ Iterator<T> j = t.PreOrderiterator();
f:___ while (j.hasNext()) this.insert(j.next());
}
```

a) [8 pts] **Worst case.** Perform the analysis of the approximate number of times an element gets accessed. Then perform the big-Oh analysis. Show your work and express it in terms of N, the size of the tree. Please indicate the runtime of each of the lines a-f.

b) [4 pts] **Alternate implementation**. Propose an implementation of `balanceTree` that is more efficient or argue that the above code is based on the most efficient algorithm.

4) Consider the following binary heap implementation of a priority queue in which items with lower numbers have higher priority, i.e. it is a min-queue.

[4 pts] Insert the following sequence of numbers into it: 4-3-2. Show your work.

/	3	4	5	6	6			

/								

/								

/								

b) [4 pts] Call poll() three times on the following min-queue. Show your work.

/	1	3	2	7	6	5	4	8

/								

/								

/								

On-the-Computer part

- Download the **`Exam2`** project located on our Moodle site located in the *Exam 1* section. Before you do that, please submit the written portion to the specified Moodle drop box.
- All code you write should be **correct, efficient,** and use **good style.** However, no documentation is required, because of time constraints.
- All code you submit **MUST** be your own work.
- Please use the provided unit test-cases, as well as your own test-cases to ensure all functional and performance requirements are met.
- The points given by the unit test-cases are assuming an honest effort. Writing code that satisfies just the test cases will result in zero points given. Partial credit will be given where appropriate.
- Implement the [25 pts] *reverse()* and [25 pts] *isAVLTree()* methods as specified in the Java Docs of the **`Exam2`** starter code.
- Submit your **BinarySearchTree.java** file to the **Exam2** drop box on our Moodle course site.
- Recall that you need to add JUNIT 4 to the build path. In the Testing.java file, hover the cursor over the "org.junit…" import, then click on "Fix project setup" in the window that pops up and then "Add JUnit 4 to build path."

```
/**
* This procedure reverses a BST, so that it is now ordered from largest
* to smallest element. This procedure produces a tree that is symmetric to
* the old tree. For example, if the current tree is as shown on the left
* side, then this method will turn that tree into the one shown on the
* right side.
*
*                   15                          15
*                 /    \                      /    \
*               10     20                   20     10
*                 \                           /
*                  12                        12
*
* This method runs in linear time. You may not insert the elements into a
* new tree.
*/

public void reverse() {

/**
* This procedure determines whether a BST in fact satisfies the
* requirements of an AVL tree. A BST satisfies this requirement if for
* every node in the tree, the difference in height of its children is at
* most one. This method runs in linear time. You may not add a height
* field to the BST or Node classes. You may not use a height method
* either. You need to traverse the tree assess this property.
*
* @return This method returns true, if the BST forms an AVL tree and false
* otherwise.
*/
public boolean isAVLTree(){
```