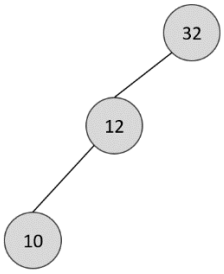
COSC 222: Data Structures
Lab 5

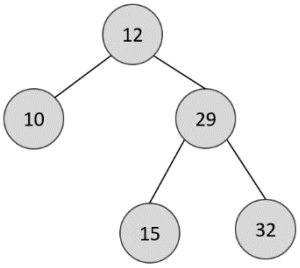
Question 1 [5 marks]

This question does not require coding. You will need to submit a PDF file with your answers. This can be handwritten and scanned, drawn on a tablet, or created using a diagramming program and word processor. Name your file **Lab5Question1.pdf**.

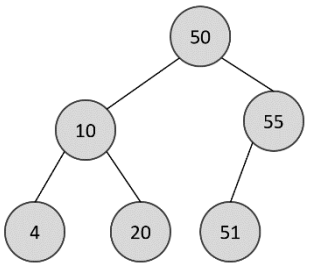
a. [2.5 marks] We have the following eight trees:



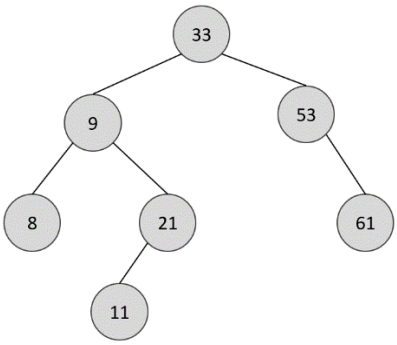
Tree a



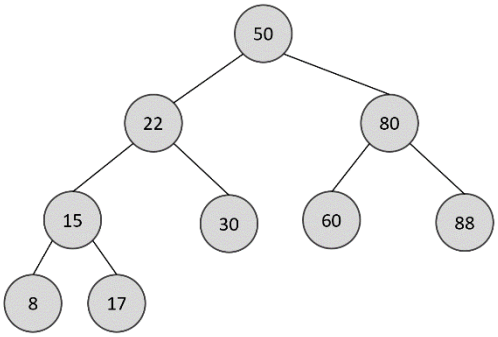
Tree b



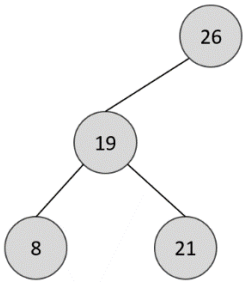
Tree c



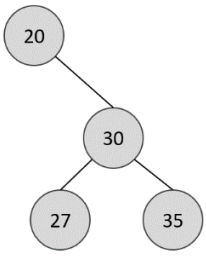
Tree d



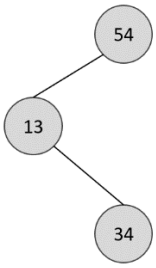
Tree e



Tree f



Tree g



Tree h

Write the tree name (e.g., Tree a, b, c, d) of the all the trees that satisfy the following tree properties:

Full tree: _____

Complete tree: _____

Perfect tree: _____

Degenerate tree: _____

Valid AVL tree: _____

b. [2.5 marks] Let's assume that we have an initially empty AVL tree. Now insert the following keys: 20, 26, 21, 27, 23, 22, 28 (in that order) into the tree. Show the tree after inserting **each key**. If you need to apply a rotation, please explicitly mention the **case** that we discussed in the class and the **rotation** that you performed to balance the tree.

Question 2 [15 marks + 2 marks (BONUS)]

Use the sample java files provided in the **Lab5.zip** file. Begin with the file `Lab5Q2Test.java` file. It contains a simple program that tests some Binary Search Tree (BST) methods. You need to complete the methods inside the `BST.java` file.

First check the `TreeNode.java` file which is similar to the `Node` class seen in the class lectures. You may want to add more instance variables and (get/set) methods to run the program.

Now check the `BST.java` file and complete the following methods:

- [2 marks] `insert()`: This is similar to the `add` method (recursive) discussed in the class. Instead of using recursive code, use *iterative* (e.g., loop) code to complete the method so that it inserts a key/item to a BST.

- [2 marks] `recPrintTreeDesc()`: there is already a `printTree()` method in this class which shows all the nodes in non-descending order. Now complete the `recPrintTreeDesc()` method to show the nodes in descending order (use recursion). [Hints: for in-order traversal (in ascending order), we traverse the left subtree, then visit the root, and then traverse the right subtree. To visit the items in descending order, we switch the left and right traversals in the recursive algorithm]

- [2 marks] `recursivePrintLeafs():` write a recursive solution to show the leaf nodes of the BST. From the class lecture, we know that leaf nodes do not have both left and right child. For such nodes (i.e., leaf nodes), we can just print the node values. [Hints: check if the node is null, check if the left and right subtrees are null, recursively call the left and right subtrees]

- [2 marks] `recursiveInternalNodes()`: write a recursive solution to show the internal nodes (i.e., non-leaf nodes) of the BST. This is very similar to `recursivePrintLeafs()`; however, we will only show key value if the left or right child of a node is not null.

- [2 marks] `recursiveCountNodes()`: complete this method to return the number of nodes in the subtree rooted at a given key value (i.e., first search the `key` value to find the root). Note that the program should count itself (i.e., subtree root) and all its descendants. If the root is not found in the tree, the program should display 0 node.

- [1 mark] isPerfectTree() : Complete this method to return true if the tree is **Perfect**, return false otherwise. Refer to lecture 8, part 3, slide 39.

- [1 mark] isFullTree() : Complete this method to return true if the tree is **Full**, return false otherwise. Check lecture8 tree - part 3 slide 37.

- [1 mark] isCompleteTree(): Complete this method to return true if the tree is **Complete**, return false otherwise. Check lecture 8 tree - part 3 - page 38. [Hints: if the depth of a tree is d and the total number of nodes is n, then n should always be n >= 2d + 1 for the left subtrees, and n>= 2d + 2 for the right subtrees in each level of the tree].

- [1 mark] getHeight(): Complete this method to return the height of the tree.

- [1 mark] isDegenerateBinaryTree() : Complete this method to return true if the tree is a **Degenerate Binary Tree**, return false otherwise. Hint: Check the lecture and observe the relation between the number of nodes and the height of the tree.

- [2 marks (BONUS)] Method `findSmallest()`: complete the method to find the minimum value stored in the subtree rooted at a given key value (i.e., subtree root value). You can assume that the root is always present in the tree.

Sample Output:

```
Inorder traversal: 8 9 10 11 30 40 52 55 61 62
Tree in Descending Order: 62 61 55 52 40 30 11 10 9 8
Leaf nodes are: 8 40 55 62
Internal nodes are: 52 30 11 10 9 61
Number of nodes: 10
Smallest node in the subtree rooted 10: 8
Number of nodes in the subtree rooted 10: 3
-------- -------- --------
Height of tree: 2
Height of tree: 3
Height of tree: 3
Height of tree: 4
-------- -------- --------
Tree is NOT Pefect
Tree is NOT Pefect
Tree is NOT Pefect
Tree is NOT Pefect
-------- -------- --------
Tree is Complete
```

```
Tree is NOT Complete
Tree is Complete
Tree is NOT Complete
-------- -------- --------
Tree is NOT Full
Tree is Full
Tree is NOT Full
Tree is NOT Full
-------- -------- --------
Tree is NOT Degenerate
Tree is NOT Degenerate
Tree is NOT Degenerate
Tree is Degenerate
```

Submission instruction:

- Create a folder called "Lab5_<*student_ID* >" where <*student_ID* > is your student number/ID. Only include the java files in the folder. **DO NOT** include any other files (e.g., .class, .java~ or any other files)
    - For Question 1, only include **Lab5Question1.pdf** file.
    - For Question 2, include Hand in your **Lab5Q2Test.java**, **BST.java** and **TreeNode.java** files.
- Make a **zip file** of the folder and upload it to Canvas.
- To be eligible to earn full marks, your Java programs **must compile and run** upon download, without requiring any modifications.
- These assignments are your chance to learn the material for the exams. Code your assignments independently