National University of Singapore
School of Computing
IT5003: Data Structure and Algorithm
Semester I, 2019/2020

Tutorial x Lab 5
Binary Tree & BST

General idea: Tutorial type questions will be listed first in this document. Some tutorial question will then be implemented in the lab portion (later part of this document). Lab exercise(s) to be submitted on Coursemology will be marked [SUBMISSION] at the front.

1. Suppose you have a **binary tree T** containing only distinct items. The pre-order sequence of **T** is { 8, 7, 5, 28, 4, 9, 18, 17, 16 }, and the in-order sequence of **T** is {5, 7, 4, 28, 9, 8, 18, 16, 17}.

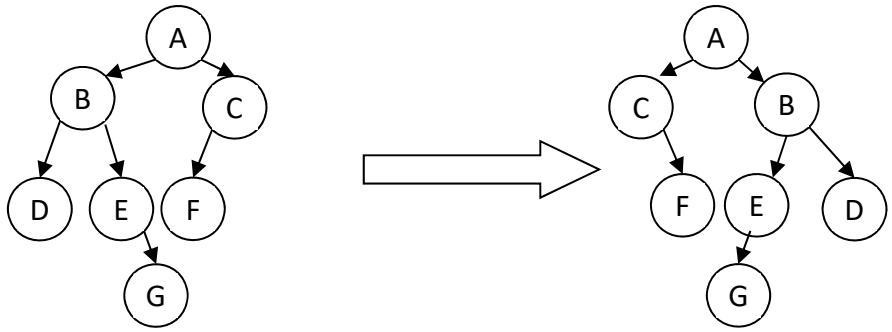
   a. Can you reconstruct a **unique T** from these two traversal sequences? If yes, reconstruct it (show a drawing of the tree).

   b. Give a general construction algorithm if (a) is possible.

```
def construct binaryTree( preOrder, inOrder ):
""" [preOrder] and [inOrder] are both Python lists."""
```

   c. If you are given the pre-order and **post-order** sequences of a binary tree, can you construct a **unique** binary tree? Explain your answer.

2. Two binary trees are mirror images of each other if their roots and left and right subtrees are reflected across a vertical line as shown. Write a **recursive method** to create a mirror image of a given binary tree pointed by **T**.




```
def flipTree( T ):
""" Return the mirror image of T. Note that you should create
new TreeNode object instead of changing the existing tree
nodes. """
```

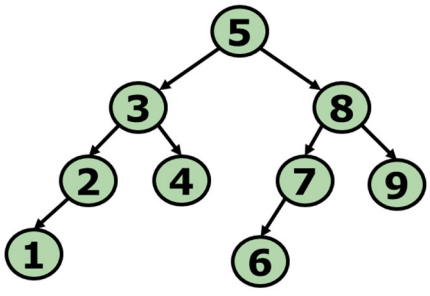
3. [BST basic Operations]

   a. Draw the binary search tree after inserting the following integers in sequence: 20, 10, 30, 5, 7, 25, 15, 37, 12, 18 and 40.

   b. What are the corresponding output using (i) pre-order, (ii) in-order, (iii) post-order, (iv) level-order traversal?

   c. Draw the new binary search tree after the following operations: delete 30, delete 10, delete15 (You should follow the deletion algorithm in the lecture notes). Is the new tree a complete binary tree?

4. [Balanced BST from Sorted List]
   Suppose you are give **L**, a list of number in ascending order, give an algorithm to construct a **balanced BST** with the values in L. Whenever there is an uneven split of nodes between the left / right subtree, we will let left subtree to have 1 more node the right subtree. For example, given L = [1, 2, 3, 4, 5, 6, 7, 8, 9], the BST constructed should be:



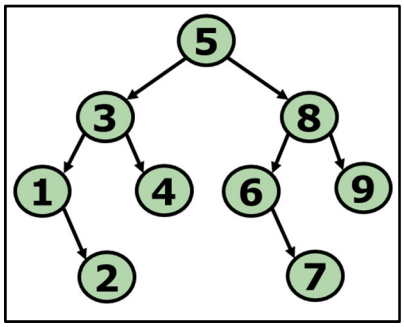
~~~ Lab Questions ~~~

1. [**Pro-Gardener** - **Submission**] Let us implement the following **two** functions for constructing a **BST** (note: not binary tree, but binary **search tree**) in different ways. For this question, we have provided the BSTMod class, which is very similar to BSTRef class discussed in lecture with **a few simplifications:**
    - Base class removed, so that it is a standalone simple class.
    - Non-essential method like Deletion, findMin methods are removed.
    - The traversal methods return the **sequence of traversal as a list** instead of a "pretty string".

    Part A – Build BST from Preorder list

```
def buildBSTfromPreorder( L ):
""" [L] is a list of number organized as a pre-order traversal
of a BST. Rebuild and return the BST from [L]. """
```

    For example:



    The preorder sequence is [5, 3, 1, 2, 4, 8, 6, 7, 9]. Using this sequence, you can rebuild the BST from scratch. Note that you are **not allowed** to use the `insert()` method, i.e. you need to learn how to link up the left / right subtree through the references directly.

    Note that we design this as a method of the BSTMod class, i.e. calling this method will rebuild (and destroy current content of the BST). The actual implementation will be a recursive helper method as shown in the skeleton code.

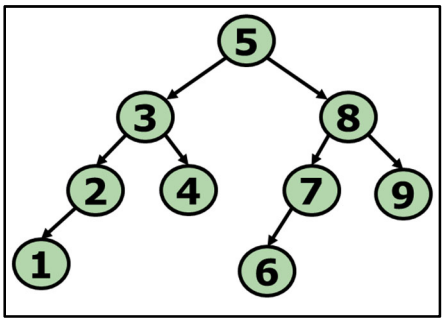
Part B – Build Balanced BST from any list

```
def buildBalancedBST( L ):
""" [L] is a list of numbers in random order. Build and return
a balanced BST from [L]. """

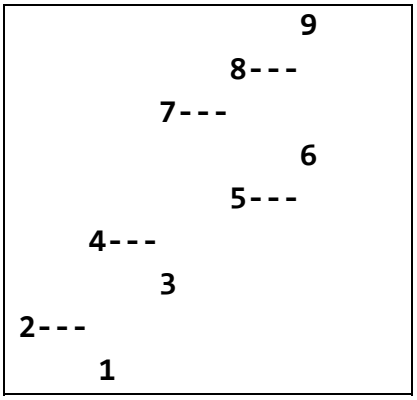
Restriction: You are NOT allowed to sort [L] and can only make
use of the BST methods to help.
"""
```

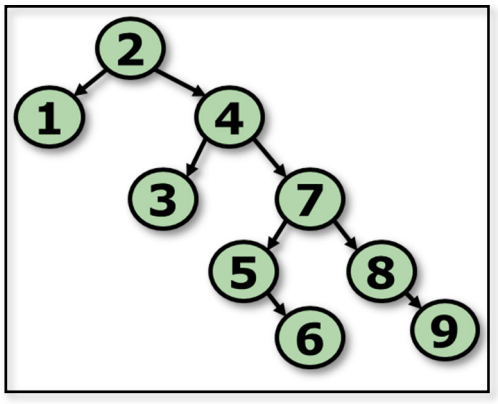
Whenever there is an uneven split of nodes between the left / right subtree, your code should ensure left subtree will have 1 more node the right subtree. Similar to part (a), your code needs to link the `TreeNodes` directly during construction.

For example, given the random sequence [2, 4, 7, 8, 1, 5, 6, 3, 9], your code should return a BST that looks like:



2. [Binary Tree / BST] As shown in the lecture, you can get creative with the tree printing other than the standard traversal. Add a "`prettyPrint()`" method to the Binary Tree / BST class (e.g. the `BSTRef` from Lecture 9) to print the tree in the following fashion:





You can see that the print out is essentially a 90 degree rotated version of the actual tree! The "---" are added so that it is easier to see the left / right side of a subtree.

3. The `BSTRef` class given ("`L9-BST-Code.zip`") does not have the `delete()` method. Use the "`insert()`" method to learn how to translate the pseudo code into actual Python code. Then, implement the "`delete()`" method accordingly.

We will evaluate both correctness and programming style of your submitted code:

a. [Correctness 70%]: The code works according to the functional requirement, e.g. output value, output format, side effect, efficiency etc.

b. [Programming Style 30%]: Reuse own code whenever appropriate, modularity, good variable / method naming convention, comments (only if appropriate)