CSC/DCSCI 2720: Data Structures
Lab 11

Instructor: Shiraj Pokharel

Due : 48 hours after release
Late Submission deadline (with 25% penalty) 24 hours after due date

Answer the below questions.
You must submit your responses as a SINGLE Jupyter Notebook, where each program is put in separate Jupyter Notebook cells within that SINGLE Jupyter Notebook. Do **NOT** submit Colab links.
Failure to comply with this simple requirement will result in a score of Zero. Please, be careful not to be assigned a Zero score this way.

Few Rules to be followed, else will receive a score of ZERO

(1) Your submissions will work exactly as required.
(2) Your files shall not be incomplete or worse corrupted such that the file does not compile at all. Make sure you submit a file that compiles.
(3) Your submission will show an output. Should you recive a Zero for no output shown do not bother to email me with "but the logic is perfect" !

Note that your program's output must **exactly** match the specs(design , style) given here for each problem to pass the instructor's test cases .
Design refers to how well your code is written (i.e. is it clear, efficient, and elegant), while *Style* refers to the readability of your code (commented, correct indentation, good variable names).

PROBLEM STATEMENT :
In today's Lab we will explore two specific ways to perform a validation check of whether a Binary Tree is actually a **Binary Search Tree (BST)**.

Each implementation will be a separate "cell" in your JuPyter Notebook.

[1] **50 Points** Perform a check of constraints on node values for each sub-tree, just the way we discussed in the Lecture. Please remember what we discussed

in the Lecture - that - for a Binary Tree to qualify as a **Binary Search Tree**, we must peform the node values check of-course but also must not forget the value checks at the sub-tree levels.

[2] **50 Points** Perform the BST check by doing an **In-Order Traversal** of the Binary Tree as discussed in the Lecture. Since we know that an in-order traversal of a BST results in nodes being processed in ascendingly sorted order, as soon as there is a violation of ascendingly sorted order we would know that the tree provided is not a BST.

```
# Class to represent Tree node
class Node:
  # A function to create a new node
  def __init__(self, key):
    self.data = key
    self.left = None
    self.right = None
```

The root element of the Binary Tree is given to you. Below is an illustrated sample of Binary Tree nodes for your reference, which in-fact is the same example we discussed in the lecture.

```
root = Node(4)
root.left = Node(2)
root.right = Node(6)
root.left.left = Node(1)
root.left.right = Node(3)
root.right.left = Node(5)
root.right.right = Node(7)
```

Your code will need to return a boolean : True or False.
When you follow the validation process specified - the complexity of the solution will be as below.

Time Complexity: $O(n)$
Space Complexity: $O(n)$

The linear space complexity would come from the recursion (AKA "recursion stack") or an explicit stack in-case of iterative traversal method you employ to validate the Tree.
Submissions that don't meet the linear Time and Space complexities will only receive 50% credit.

Very Very Important :

(1) Your code should be well commented which explains all the steps you are performing to solve the problem. A submission without code comments will immediately be deducted 15 points !

(2) As a comment in your code, please write your test-cases on how you would test your solution assumptions and hence your code.
A submission without test cases (as comments) will immediately be deducted 15 points ! Please Remember : Although, written as comments - You will address your test cases in the form of code and not prose :)