# CS 5012
# Homework 6.8: Binary Search Trees
Due date: Sunday, March 21, 2021

**LEARNING OBJECTIVES:**
- Implement a Binary Search Tree (BST) in Python
- Implement the 3 tree traversal methods in Python
- Design and implement a `find` ("search") method in Python that finds a particular node (based on the data value). This method relies on an understanding of the tree traversal methods and the way nodes are added to the BST
- Design and implement a `size` method in Python. This method relies on an understanding of the tree traversal methods

**GRADING:**
- A maximum of **100 points** can be obtained on this homework assignment.
- There are five (5) methods to write and each method will be worth 20 points.

**SUBMITTING:**
- Submit on Collab
- Submit your Python (.py) file(s) along with a text document showing the output of the provided test cases
- You **may work in pairs** on this homework
- At the top of your document be sure to include your **name** and **computing ID**
  - If you worked in pairs, be sure to include **BOTH** names and computing IDs
- The submission deadline is **11:59pm** on the date the assignment is due, mentioned above

**PROBLEM DESCRIPTION:**
You are given some skeleton Python code that consists of two classes: a `Node` class and a `BinarySearchTree` class. The `Node` class describes the structure of a node in the tree: each node has a data item and can have a left and right child. The `BinarySearchTree` class is responsible for tree-level methods such as `buildBST`, inserting a data value in the right place/node in the BST tree (we populate the tree given a list data values through `main`), and the tree traversal methods. Your task is to write a number of methods in the `BinarySearchTree` Class.

**Write the following methods:**

(1) **find**

This method takes in a target (data value) and returns True or False depending whether or not the target node was found.

Use the following header:

```
def find(self, target):  # Find a node with the 'target' value in the BST
```

(2) **size**

This method counts the number of nodes in a tree. It should return -1 if the tree is empty (no nodes added), otherwise it should return the number of nodes in the tree. *Note: Considering you've seen how to write a combined tree-wide and node-level implementation, we explicitly ask to do this in the BST class.*

Use the following header:

```
def size(self, node):  # Counts the number of nodes in the BST
```

(3) **inorder**
This method visits (and prints) each node in the tree using the in-order tree traversal algorithm. The output should be a list of the nodes visited (data values printed out).
Use the following header:

```
def inorder(self, node):  # Performing in-order tree traversal
```

(4) **preorder**
This method visits (and prints) each node in the tree using the pre-order tree traversal algorithm. The output should be a list of the nodes visited (data values printed out).
Use the following header:

```
def preorder(self, node):  # Performing pre-order tree traversal
```

(5) **postorder**
This method visits (and prints) each node in the tree using the post-order tree traversal algorithm. The output should be a list of the nodes visited (data values printed out).
Use the following header:

```
def postorder(self, node):  # Performing post-order tree traversal
```

Once you've finished writing your methods run the following lines of code (**test cases**) to demonstrate that your methods work:

```
tree = BinarySearchTree()
treeEmpty = BinarySearchTree()  # Empty tree
arr = [8,3,1,6,4,7,10,14,13]    # Array of nodes (data items)
for i in arr:                   # For each data item, build the Binary Search Tree
    tree.buildBST(i)

print('What\'s the size of the tree?')
print(tree.size(tree.root))
print('What\'s the size of the tree?')
print(treeEmpty.size(treeEmpty.root))
print("")
print ('In-order Tree Traversal:')
tree.inorder(tree.root)         # Perform in-order tree traversal, and print
print("")
print ('Pre-order Tree Traversal:')
tree.preorder(tree.root)        # Perform pre-order tree traversal, and print
print("")
print ('Post-order Tree Traversal:')
tree.postorder(tree.root)       # Perform post-order tree traversal, and print
print("")
print ('Find 7:', end=" ")
print(tree.find(7))
print('Find 5:', end=" ")
print(tree.find(5))
print('Find 30:', end=" ")
print(tree.find(30))
```