Objectives: You will gain experience BST performance and implementation

To start the lab: Download and unzip the file: http://www.cs.uni.edu/~fienup/cs1520f19/labs/lab9.zip

Part A: Consider the Binary Search Tree (BST) below. For each node in a BST, all values in the left-subtree are < the node and all values in the right-subtree are > the node.

<table>
<thead>
<tr>
<th>Level</th>
<th>Height of a BST is the max. level so the height is 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

a) Review section 6.5.2 on Tree Traversals to determine the order nodes are processed in each tree traversal.
   - What is the order of node processing in a preorder traversal of the above BST?
   - What is the order of node processing in an inorder traversal of the above BST?
   - What is the order of node processing in a postorder traversal of the above BST?

b) Starting with an empty BST, what would be the shape of the BST after put’s for keys: 50, 60, 30, 70, 90, 40, and 65?

After you have answered the above questions, raise your hand and explain your answers.
Part B: Run the `timeBinarySearchTree.py` program that:
- creates a list, `evenList`, that holds 3,000 sorted, even values (e.g., `evenList = [0, 2, 4, 6, 8, ..., 5996, 5998]`)
- puts (adds) all the `evenList` items into an initially empty `BinarySearchTree` object, `bst`
- times the searches (in) `bst` for target values 0, 1, 2, 3, 4, ..., 5998, 5999 so half of the searches are successful and half are unsuccessful

a) How long does it take to search for target values of 0, 1, 2, 3, 4, ..., 5998, 5999?

b) Explain why these searches take so long. (Hint: consider the shape of the `BinarySearchTree` `bst`)

c) Uncomment the “`shuffle(evenList)`” which randomizes the items in `evenList` before adding them to the `BinarySearchTree` `bst`. Now how long does it take to search for target values from 0, 1, 2, 3, 4, ..., 5998, 5999?

d) Explain why these searches take so little time.

e) What is the search time with the `timeOpenAddrHashDictSearch.py` program? Why is it faster?

Part C: a) Complete the recursive `height` method in the `BinarySearchTree` class. Model it after the postorder traversal, since the height of the whole BST can be determined after you know the height of the left-subtree and height of the right-subtree. For example if the left-subtree has a height of say 8 and the right-subtree has a height of 5, then the overall height including the root is 9 (i.e., one more than the tallest subtree’s height). For the base case of the recursion, if we define the empty subtree’s height to be -1 (i.e., `subtreeRoot` points to `None` since it has no `TreeNode` to point at), then the recursive definition still works for a leaf node which should have a height of 0.

b) Uncomment the call to the `height` method at the end of the `timeBinarySearchTree.py` program. What is the height of `bst` if we are shuffling the `evenList`?

c) What would be the shortest possible height for a binary tree with 3,000 items?

After you have completed the `height` method and answered the above questions, raise your hand and explain your answers.