```cpp
/*// ------------------------
Justin Dang
Student ID: 1148267

Note that (numbers) (ex. (1), (2)) are cited documents below.
// ----------------------------------------------------------------
The following code function:
Create a binary tree that is sorted

With the tree being sorted, we are allowed to search the tree with a the average case being O(logn) and the
worst case being O(n).(2)

// ----------------------------------------------------------------
Trees(data structure):
Similar to Linked Lists, we do not have random access. As a result we find ourselves using a
linear search to traverse linked lists which are not ideal for some situations.

Trees help create a more efficient method for traversal of a linked list.

Trees come in different shapes and sizes, a binary tree can only have two paths or nodes per previous existing ndoe.

Important things to know about a tree: ( "words" are definitions listed below )

Root:   The first/parent node of our tree. Similar to linked lists this will act as our front and access point to the
        rest of the tree.

Path:   There can be multiple paths in a tree. A binary tree can have two paths per node, A N tree can have N paths, etc.
        The length of a path is determined by the distance from the "root" to a "leaf".

Height: The "height" of a tree can be determined by the length of the longest "path". There are many methods to calculate
        the "height" of a tree, but in our case we use our "root" as level 0, next node in our tree as "level" 1, and so on
        adding 1 to represent the next "level" in our tree.

Level:  A tree will have levels within it similar to the height of a tree. Levels are directly proportional to the
        height of a tree.

Leaf:   The last node in a "path". A leaf is not a "root" and is the last node in a "path" that signifys the end
        of a tree.

For more information or clarification: https://docs.google.com/document/d/1RmGQzEHF10mUlvhPCtBlHsUtp-LslpHk6qRUoqIMTBk/edit

```

```
// -------------------------------------------------------------
Cited Work:

Reference Document(1): BinaryTrees.cpp provided by prof.

Binary Search Tree Wikipedia(2): https://en.wikipedia.org/wiki/Binary_search_tree

Tree Traversal(3): https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/
*/// ------------------------
#include <iostream>
using namespace std;

// Blueprint for each node in our binary tree
class Node {
public:
    int data;                              // Data(int) stored in for this node
    Node* left, * right;                   // Addresses to the left or right node

    Node() { left = right = 0; }           // sets the address to the left or right node to null(to symbolize a leaf)

    Node(int d, Node* l = 0, Node* r = 0) { // assigns data and addresses
        this->data = d;
        left = l;
        right = r;
    }
};

class binaryTree {
public:
    Node* root;

    binaryTree() { root = 0; }

    /*
    Adds a node to our binary tree.

    Organizes our nodes by data(int) size:
    1. if larger  -> we proceed down the right side of our tree
    2. if smaller -> we proceed down the left side of our tree

    The node is then placed at the end as a leaf of our tree
    */
    void addNode(int data) {
        Node* newNode = new Node(data);

        // Check if the tree is empty(no root), if so simply set the root to our newNode
        if (root == 0) {
            root = newNode;
```

```
        }

        Node* temp = root; // Since we dont want to alter/manipulate our root, we create a copy to traverse our list

        while (true) {
            if (data < temp->data) {      // Starting at the root, we check if newNode has a smaller int than our root,
                                          // if so we proceed down the left side of our tree

                if (temp->left == 0) {    // We then check if our root has left node already in place
                    temp->left = newNode; // If not we set our root's left node to our newNode~~~~~~~~~~~~~~~~~~~~~
                    break;
                }
                else
                    temp = temp->left;    // If our left node is taken, then we continue down the list
            }
            else if (data > temp->data) { // else if our newNode is larger than the in in our root, we proceed down the
                                          // right side of our tree

                if (temp->right == 0) {   // Check if our root has a right node in place
                    temp->right = newNode;// if not we set our root's right node to our newNode~~~~~~~~~~~~~~~~~~
                    break;
                }
                else
                    temp = temp->right;   // If our right node is taken, then we continue down the list
            }
            else if (data == temp->data) {// No duplicates should be entered as written in ref doc(1)
                break;
            }
        }
    }
    /*
    prints tree in LVR
    */
    void print(Node* currentNode) {
        if (currentNode != 0) {
            print(currentNode->left);                   // traverse left subtree           L
            cout << currentNode->data << " ";           // evaluate (print) current node   V
            print(currentNode->right);                  // traverse right subtree          R
```

```
        }
    }
};

int main()
{
    binaryTree binTree1;
    for (int x = 0; x < 7; x++) {
        cout << "Adding (" << x << ") to the tree. . .\n";
        binTree1.addNode(x);
    }
    cout << "Binary Tree in LVR: ";
    binTree1.print(binTree1.root);
    cin.get();
}
```