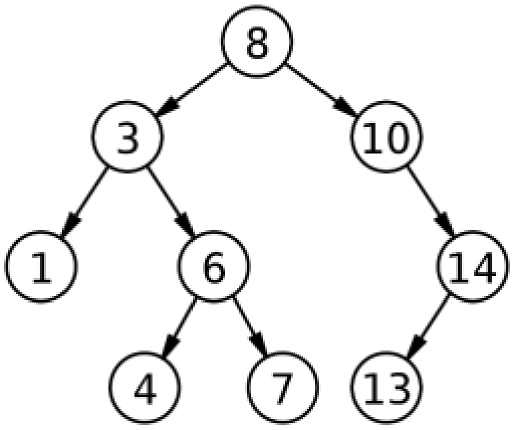
Add a New Element to a Binary Search Tree

This series of challenges will introduce the tree data structure. Trees are an important and versatile data structure in computer science. Of course, their name comes from the fact that when visualized they look much like the trees we are familiar with in the natural world. A tree data structure begins with one node, typically referred to as the root, and from here branches out to additional nodes, each of which may have more child nodes, and so on and so forth. The data structure is usually visualized with the root node at the top; you can think of it as a natural tree flipped upside down.

First, let's describe some common terminology we will encounter with trees. The root node is the top of the tree. Data points in the tree are called nodes. Nodes with branches leading to other nodes are referred to as the parent of the node the branch leads to (the child). Other more complicated familial terms apply as you might expect. A subtree refers to all the descendants of a particular node, branches may be referred to as edges, and leaf nodes are nodes at the end of the tree that have no children. Finally, note that trees are inherently recursive data structures. That is, any children of a node are parents of their own subtree, and so on. The recursive nature of trees is important to understand when designing algorithms for common tree operations.

To begin, we will discuss a particular type of a tree, the binary tree. In fact, we will actually discuss a particular binary tree, a binary search tree. Let's describe what this means. While the tree data structure can have any number of branches at a single node, a binary tree can only have two branches for every node. Furthermore, a binary search tree is ordered with respect to the child subtrees, such that the value of each node in the left subtree is less than or equal to the value of the parent node, and the value of each node in the right subtree is greater than or equal to the value of the parent node. It's very helpful to visualize this relationship in order to understand it better:



Now this ordered relationship is very easy to see. Note that every value to the left of 8, the root node, is less than 8, and every value to the right is greater than 8. Also notice that this relationship applies to each of the subtrees as well. For example, the first left child is a subtree. 3 is the parent node, and it has exactly two child nodes — by the rules governing binary search trees, we know without even looking that the left child of this node (and any of its children) will be less than 3, and the right child (and any of its children) will be greater than 3 (but also less than the structure's root value), and so on.

Binary search trees are very common and useful data structures because they provide logarithmic time in the average case for several common operations such as lookup, insertion, and deletion.

We'll start simple. We've defined the skeleton of a binary search tree structure here in addition to a function to create nodes for our tree. Observe that each node may have a left and right value. These will be assigned child subtrees if they exist. In our binary search tree, you will create a method to add new values to the tree. The method should be called `add` and it should accept an integer value to add to the tree. Take care to maintain the invariant of a binary search tree: the value in each left child should be less than or equal to the parent value, and the value in each right child should be greater than or equal to the parent value. Here, let's make it so our tree cannot hold duplicate values. If we try to add a value that already exists, the method should return `null`. Otherwise, if the addition is successful, `undefined` should be returned.

Hint: trees are naturally recursive data structures!

⚗	The `BinarySearchTree` data structure should exist.
⚗	The binary search tree should have a method called `add`.
⚗	The add method should add elements according to the binary search tree rules.
⚗	Adding an element that already exists should return `null`.