





Check if an Element is Present in a Binary Search Tree

Now that we have a general sense of what a binary search tree is let's talk about it in a little more detail. Binary search trees provide logarithmic time for the common operations of lookup, insertion, and deletion in the average case, and linear time in the worst case. Why is this? Each of those basic operations requires us to find an item in the tree (or in the case of insertion to find where it should go) and because of the tree structure at each parent node we are branching left or right and effectively excluding half the size of the remaining tree. This makes the search proportional to the logarithm of the number of nodes in the tree, which creates logarithmic time for these operations in the average case. Ok, but what about the worst case? Well, consider constructing a tree from the following values, adding them left to right: 10, 12, 17, 25. Following our rules for a binary search tree, we will add 12 to the right of 10, 17 to the right of this, and 25 to the right of this. Now our tree resembles a linked list and traversing it to find 25 would require us to traverse all the items in linear fashion. Hence, linear time in the worst case. The problem here is that the tree is unbalanced. We'll look a little more into what this means in the following challenges

	The <code>BinarySearchTree</code> data structure should exist.
	The binary search tree should have a method called <code>isPresent</code> .
	The <code>isPresent</code> method should correctly check for the presence or absence of elements added to the tree.
	<code>isPresent</code> should handle cases where the tree is empty.