Code Challenge #3 Find Closest Value In BST (Easy)

Write a function that takes in a Binary Search Tree (BST) and a target integer value and returns the closest value to that target value contained in the BST.

You can assume that there will only be one closest value.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

Sample Input

```
tree =   10
       /     \
      5       15
    /   \   /   \
   2     5 13   22
 /           \
1             14

target = 12
```

Sample Output

```
13
```

Solution #1

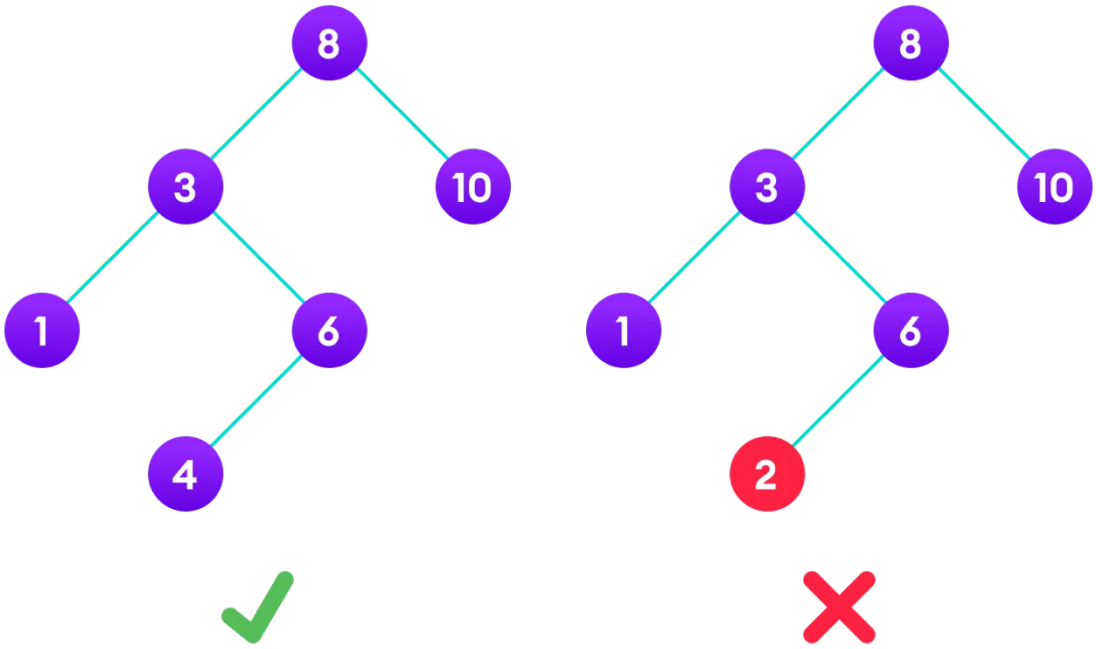
```
1. function findClosestValueInBst(tree, target) {
2.   return findClosestValueInBstHelper(tree, target, tree.value)
3. }
4.
5.
6. function findClosestValueInBstHelper(tree, target, closest) {
7.  let currentNode = tree;
8.  while (currentNode !== null) {
9.   if (Math.abs(target - closest) > Math.abs(target -
   currentNode.value)) {
```

```
10.                     closest = currentNode.value;
11.                     }
12.          if (target < currentNode.value) {
13.                 currentNode = currentNode.left;
14.          } else if (target > currentNode.value) {
15.                 currentNode = currentNode.right;
16.          } else {
17.                 break;
18.          }
19.         }
20.         return closest;
21. }
22.
```

Explanation

In order to find the closest value to the target value in the BST. We must go through each items aka the nodes in the BST. A binary search tree is called a binary search tree because it can have a maximum of two children. Searching for an item in the binary search tree can be accomplished in O(log(n)) because the numbers in the node are stored in an ordered fashion. In a binary search tree the root node splits the numbers to the left of it as numbers lower than it and the numbers to the right are numbers bigger than it.

Example of a Binary Search Tree



* The binary tree on the right isn't a binary search tree because the right subtree of node "3" contains a value smaller than it.

Code Solution Breakdown

We solve this problem using a helper function which we call within the main function. This helper function takes in three arguments. The tree, target and closest value. The first thing we do is pass the tree into a variable called currentNode. We do this because we will start off by starting with the root node of the tree. We will then use a while loop to keep the loop running as long as the currentNode is not equal to null. Null values in a binary tree indicates you reached the end of the Binary Tree. We will then use if statements to check if the difference using Math.abs to see if the target – closest is greater than target – currentNode.value. If it is then we assign the closest value to the currentNode.value because it is the smallest difference of the target value. * This won't be the case because the closest value is the same as the currentNode.value which is both the root value. We will move on to the next if statement which will check to see if the target value is less than the currentNode.value. If it is we move the currentNode value to the left by assigning the currentNode to currentNode.left. We move to the left because target is smaller than CurrentNode value so any value close to it will be on the left side of the BST due to the structure of a BST (left

values are smaller than the root). If the target is greater than the currentNode.value we move to the right and assign the currentNode to the currentNode.right value. If the target is bigger than the currentNode than the closest value will be to right since the numbers to the right are bigger numbers (bigger numbers will have a smaller difference to the target value in this case). If none of these conditions apply than we use a break to break the code since it means that the current value is equal to the closest value. The loop will continuously run unless the targets and closest value match or we reach a node of null which means we reached the end of the BST. We can finally go to the end of the code where we return the closest.