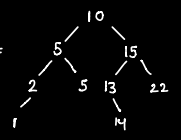
Find Closest Value in BST

Input:

tree =



target = 12

Output: 13

Input: Binary Search Tree
Target value

Output: return closest value to the target value contained in the BST

Assume: There will only be one closest value

Iterative Solution:

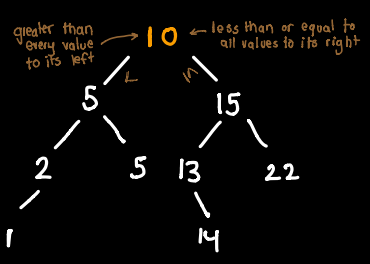
```
// Average: O(log(n)) time | O(1) space
// Worst: O(n) time | O(1) space
function findClosestValueInBst(tree, target) {
  let winningNode = tree;
  let currNode = tree;
  let winningDiff = 0;
  let currDiff = 0;

  while (currNode) {
    winningDiff = Math.abs(target - winningNode.value);
    currDiff = Math.abs(target - currNode.value);

    if (currDiff < winningDiff) {
      winningNode = currNode;
    }

    if (currNode.value < target) {
      currNode = currNode.right;
    } else {
      currNode = currNode.left;
    }
  }
  return winningNode.value
}
```

Declare a winningNode and currentNode value.
currNode ⟶ node that we are currently at
winningNode ⟶ node that is closest to our target
We set the curr/winning node to be the root of the tree
We then calculate the difference between the target and the winningNode/currNode. If the currNode difference is smaller than the winningNode difference, the currNode becomes the winningNode
We then traverse the tree and do this at each node
Once we reach a null node, we exit the while loop and return the winningNode



O(logn) time on avg since we get rid of half the BST at each iteration

O(n) time at worst since the tree could be one branch only

O(1) space since no more space gets used as input grows

Recursive Solution:

```
// Average: O(log(n)) time | O(n) space
// Worst: O(n) time | O(n) space
function findClosestValueInBst(tree, target) {
  return findClosestValueInBstHelper(tree, target, tree)
}

function findClosestValueInBstHelper(tree, target, winner) {
  if (tree === null) return winner.value
  let currNode = tree
  let winningNode = winner
  let currDifference = Math.abs(target - currNode.value)
  let winningDifference = Math.abs(target - winningNode.value)

  if (currDifference < winningDifference) {
    winningNode = currNode
  }

  if (currNode.value < target) {
    return findClosestValueInBstHelper(currNode.right, target, winningNode)
  } else if (currNode.value > target) {
    return findClosestValueInBstHelper(currNode.left, target, winningNode)
  } else {
    return winningNode.value
  }
}
```

With the use of a helper function, we can solve this problem recursively.
We initially pass in the the tree who's root node is the default currNode and winningNode (as done in the iterative solution)
After comparing the differences between currNode and winningNode and setting the winningNode appropiately, we compare the currNode's value to the target
We then recursively call the function again but with the new currentNode's left or right node
If the target is equal to the currNode's value, we return the currentNode (equal to the winning node)
If the currNode is null we return the winningNode's value.

Time: O(logn) average, O(n) worst } as mentioned above

Space: O(logn) average, O(n) worst } This because recursive solutions use the Stack data structure. The stack builds up to the height of the tree (logn average case and n worst case)