



# Solution Review: Find the Height of a BST

## We'll cover the following

- Solution: recursively finding the heights of the left and right sub-trees
- Time Complexity

## Solution: recursively finding the heights of the left and right sub-trees

#

main.py

BinarySearchTree.py

Node.py

```
1 from Node import Node
2 from BinarySearchTree import BinarySearchTree
3
4
5 def findHeight(root):
6     if root is None: # check if root exists
7         return -1 # no root means -1 height
8     else:
9         max_sub_tree_height = max(
10             findHeight(root.leftChild),
11             findHeight(root.rightChild)
```

```

11     findHeight(root.rightChild)
12     ) # find the max height of the two sub-tree
13     # add 1 to max height and return
14     return 1 + max_sub_tree_height
15
16
17 BST = BinarySearchTree(6)
18 BST.insert(4)
19 BST.insert(9)
20 BST.insert(5)
21 BST.insert(2)
22 BST.insert(8)
23 BST.insert(12)
24 BST.insert(10)
25 BST.insert(14)
26
27
28 print(findHeight(BST.root))

```



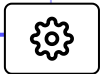
Here, we return -1 if the given node is **None**. Then, we call the `findHeight()` function on the left and right subtrees and return the one that has a greater value plus 1. We will not return 0 if the given node is **None** as the leaf node will have a height of 0.

## Time Complexity#

The time complexity of the code is  $O(n)$  as all the nodes of the entire tree have to be traversed.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ



[← Back](#)

Challenge 4: Find the Height of a BST

Challenge 5: Find Nodes at "k" distanc...

 Completed[Report an Issue](#)