Rajasekhar Reddy Kolagotla

Other

PDF generated at: 28 Jul 2025 06:07:49 UTC

View this report on HackerRank

Score

68.8% • 55 / 80

scored in TIP102: Unit 8 Version A (Standard) - Summer 2025 in 38 min 13 sec on 27 Jul 2025 22:27:23 PDT

Candidate Information

Email	rajasekhar1131997@gmail.com
Test	TIP102: Unit 8 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	27 Jul 2025 22:27:23 PDT
Time taken	38 min 13 sec/ 90 min
Personal Member ID	126663
Email Address with CodePath	rajasekhar1131997@gmail.com
Github username with CodePath	Rajasekhar1131997
Invited by	CodePath

Skill Distribution

There is no associated skills data that can be shown for this assessment

Tags Distribution

There is no associated tags data that can be shown for this assessment

Questions

Coding Questions • 40 / 60

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	1	Inorder Traversal of Binary Tree Coding	6 min 25 sec	-	20/20	-
✓	2	Binary Search Tree (BST) Insertion Coding	2 min 26 sec	-	20/20	-

✗	3	Root Equals Sum of Descendants Coding	21 min 23 sec	-	0/20	-

Multiple Choice + Debugging • 15 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	4	Debug this code Coding	2 min 41 sec	-	5/5	-
✓	5	What is the time complexity of mystery_function()? Multiple Choice	1 min 10 sec	-	5/5	-
✗	6	Which of the following best represents the values of the tree? Multiple Choice	2 min 17 sec	-	0/5	-
✓	7	Which of the following arrays best represents its preorder traversal? Multiple Choice	1 min 30 sec	-	5/5	-

1. Inorder Traversal of Binary Tree

✓ Correct

Coding

Question description

Given the `root` of a binary tree, return a list of integers representing the inorder traversal of its nodes' values.

Example 1:
Input: root = [1,null,2,3]

Output: [1, 3, 2]

Example 2:
Input: root = []
Output: []

Example 3:
Input: root = [1]
Output: [1]

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def inorder_traversal(root):
    if not root or root is None:
        return []
```

```
20      return inorder_traversal(root.left) + [root.val] +
    inorder_traversal(root.right)
21
22  def build_tree_from_list(values):
23      if not values:
24          return None
25
26      root = TreeNode(values[0])
27      queue = [root]
28      i = 1
29      while i < len(values):
30          current = queue.pop(0)
31          if values[i] is not None:
32              current.left = TreeNode(values[i])
33              queue.append(current.left)
34          i += 1
35          if i < len(values) and values[i] is not None:
36              current.right = TreeNode(values[i])
37              queue.append(current.right)
38          i += 1
39
40      return root
41
42  if __name__ == '__main__':
43      outfile = open(os.environ['OUTPUT_PATH'], 'w')
44      input_data = sys.stdin.read().strip()
45
46      input_data = input_data.splitlines()
47
48      for data in input_data:
49          if data.strip() == "":
50              continue
51
52          data = data.replace('null', 'None')
53          tree_list = ast.literal_eval(data)
54
55          root = build_tree_from_list(tree_list)
56          result = inorder_traversal(root)
57          outfile.write(str(result) + '\n')
58      outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

Basic Case	Easy	Hidden	Success	0	0.0449 sec	11 KB
Empty Tree	Easy	Hidden	Success	0	0.0474 sec	10.8 KB
Single Node Root	Easy	Hidden	Success	0	0.0295 sec	10.8 KB
Left-Skewed Tree	Easy	Hidden	Success	0	0.0334 sec	10.9 KB
Complete Binary Tree	Easy	Hidden	Success	0	0.0285 sec	10.9 KB
Tree with `None` nodes	Easy	Hidden	Success	0	0.0553 sec	11 KB
Tree with Single Child Nodes	Easy	Hidden	Success	0	0.0308 sec	10.9 KB
Large Tree	Easy	Hidden	Success	0	0.0319 sec	10.8 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0292 sec	11 KB

No comments.

2. Binary Search Tree (BST) Insertion

Correct

Coding

Question description

Given the `root` of a binary search tree and a `value`, insert a new node with `value` into the BST. Return the root of the BST after the insertion.

Example 1:
Input: root = [4,2,7,1,3], val = 5
Output: [4,2,7,1,3,5]

Example 2:
Input: root = [40,20,60,10,30,50,70], val = 25
Output: [40,20,60,10,30,50,70,None,None,25]

Candidate's Solution

Language used: Python 3

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def insert_into_bst(root, val):
    # Write your code here
    if not root or root is None:
        return TreeNode(val)
    if val < root.val:
        root.left = insert_into_bst(root.left, val)
```

```
    else:
        root.right = insert_into_bst(root.right, val)
    return root

def build_tree(values):
    if not values:
        return None
    root = TreeNode(values[0])
    for val in values[1:]:
        insert_into_bst(root, val)
    return root

def bst_to_list(root):
    if not root:
        return []
    result = []
    queue = [root]
    while queue:
        node = queue.pop(0)
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    # Remove trailing 'None' values
    while result and result[-1] is None:
        result.pop()
    return result

if __name__ == '__main__':
    input_data = sys.stdin.read().strip().splitlines()

    for data in input_data:
        tree_data, val = data.split('],')
        tree_data += ']'
        val = int(val.strip())

        tree_list = ast.literal_eval(tree_data)

        root = build_tree(tree_list)
        result = insert_into_bst(root, val)

        print(bst_to_list(result))
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Insertion Case	Easy	Hidden	Success	0	0.0281 sec	10.9 KB
Insertion at the Root	Easy	Hidden	Success	0	0.0266 sec	11 KB
Insertion in a Single-Node Tree	Easy	Hidden	Success	0	0.0283 sec	11 KB
Insertion Causing Multiple Levels	Easy	Hidden	Success	0	0.034 sec	11 KB
Insertion at a Leaf Node	Easy	Hidden	Success	0	0.0306 sec	11 KB
Insertion into a Full Binary Tree	Easy	Hidden	Success	0	0.0365 sec	11 KB
Insertion into a Tree with Duplicates	Easy	Hidden	Success	0	0.0324 sec	11 KB
Insertion of a Minimum Value	Easy	Hidden	Success	0	0.0286 sec	10.8 KB
Insertion of a Maximum Value	Easy	Hidden	Success	0	0.0384 sec	11 KB

Large Input Tree	Easy	Hidden	Success	0	0.0445 sec	10.9 KB
Very Large Value Insertion	Easy	Hidden	Success	0	0.0387 sec	10.9 KB
Very Small Value Insertion	Easy	Hidden	Success	0	0.0427 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0277 sec	10.9 KB

No comments.

3. Root Equals Sum of Descendants

Incorrect

Coding

Question description

Given the `root` of a binary tree, write a function that returns `True` if the value of `root` is equal to the sum of the values of all its descendants. Return `False` otherwise.

Example 1:
Input: root = [10, 4, 6]
Output: True
Explanation: 10 = 4 + 6

Example 2:
Input: root = [5, 3, 1, 1, 1, 1, 1]
Output: False
Explanation: 5 != 3 + 1 + 1 + 1 + 1 + 1

Candidate's Solution Language used: Python 3

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left = left
        self.right = right
#!/bin/python3

import math
import os
import random
import re
import sys
import ast

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def root_equals_sum_of_descendants(root):
    # Write your code here
    def subtree_sum(root):
        if not root:
            return 0
        return root.val + subtree_sum(root.left) + subtree_sum(root.right)
    if not root or root is None:
        return False
    # if root.left is None or root.right is None:
    #     return False
    # if root.val == 0:
    #     return True
    # left_sum = subtree_sum(root.left)
```

```
    # right_sum = subtree_sum(root.right)
    total = subtree_sum(root)

    return root.val == (total - root.val)
def list_to_tree(lst):
    if not lst:
        return None
    root = TreeNode(lst[0])
    queue = [root]
    i = 1
    while i < len(lst):
        current = queue.pop(0)
        if i < len(lst) and lst[i] is not None:
            current.left = TreeNode(lst[i])
            queue.append(current.left)
        i += 1
        if i < len(lst) and lst[i] is not None:
            current.right = TreeNode(lst[i])
            queue.append(current.right)
        i += 1
    return root

if __name__ == '__main__':
    outfile = open(os.environ['OUTPUT_PATH'], 'w')
    input_data = sys.stdin.read().strip().splitlines()

    for data in input_data:

        root_list = ast.literal_eval(data)
        root = list_to_tree(root_list)

        result = root_equals_sum_of_descendants(root)

        outfile.write(str(result) + '\n')
    outfile.close()
def list_to_tree(lst):
    if not lst:
        return None
    root = TreeNode(lst[0])
    queue = [root]
    i = 1
    while i < len(lst):
        current = queue.pop(0)
        if i < len(lst) and lst[i] is not None:
            current.left = TreeNode(lst[i])
            queue.append(current.left)
```

```
88          i += 1
89          if i < len(lst) and lst[i] is not None:
90              current.right = TreeNode(lst[i])
91              queue.append(current.right)
92          i += 1
93      return root
94
95  if __name__ == '__main__':
96      outfile = open(os.environ['OUTPUT_PATH'], 'w')
97      input_data = sys.stdin.read().strip().splitlines()
98
99      for data in input_data:
100
101         root_list = ast.literal_eval(data)
102         root = list_to_tree(root_list)
103
104         result = root_equals_sum_of_descendants(root)
105
106         outfile.write(str(result) + '\n')
107     outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case: True	Easy	Hidden	Success	0	0.0295 sec	10.8 KB
Basic Case: False	Easy	Hidden	Success	0	0.0333 sec	10.9 KB
Tree with Missing Child	Easy	Hidden	Success	0	0.0279 sec	11 KB
Tree with Multiple Levels	Easy	Hidden	Success	0	0.0281 sec	11 KB
Single Node Tree	Easy	Hidden	Success	0	0.0419 sec	10.9 KB

Empty Tree	Easy	Hidden	Wrong Answer	0	0.0312 sec	11 KB
Larger Tree	Easy	Hidden	Success	0	0.0316 sec	10.8 KB
Tree with All Left Children	Easy	Hidden	Success	0	0.044 sec	10.8 KB
Tree with Negative Values	Easy	Hidden	Success	0	0.0279 sec	10.9 KB
Tree with None in Non-Leaf Positions	Easy	Hidden	Success	0	0.0632 sec	10.5 KB
Balanced Tree	Easy	Hidden	Success	0	0.0338 sec	10.8 KB
Pass/Fail Case	Easy	Hidden	Wrong Answer	0	0.0286 sec	11 KB

No comments.

4. Debug this code

Correct

Coding

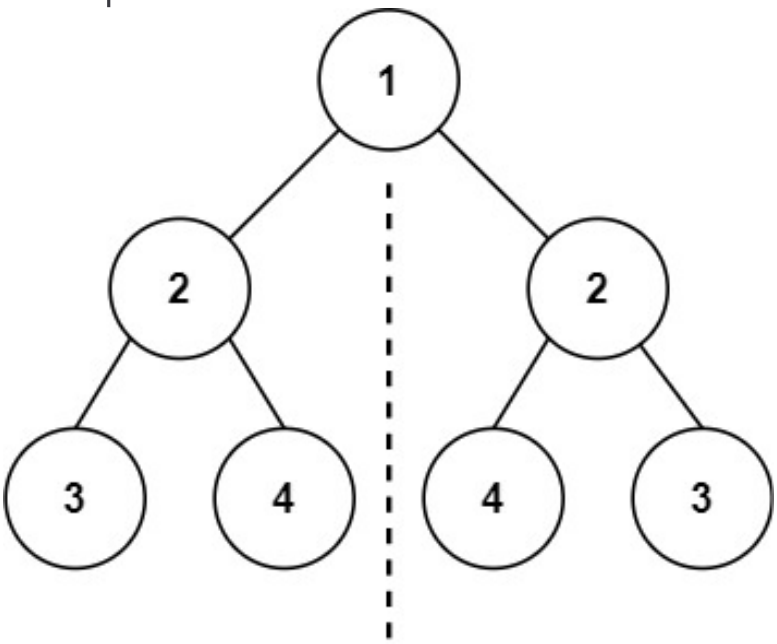
Question description

The following code incorrectly implements the function `is_symmetric()`. When implemented correctly, `is_symmetric()` accepts the `root` of a binary tree and

returns `True` if the tree is symmetric around its center and `False` otherwise.

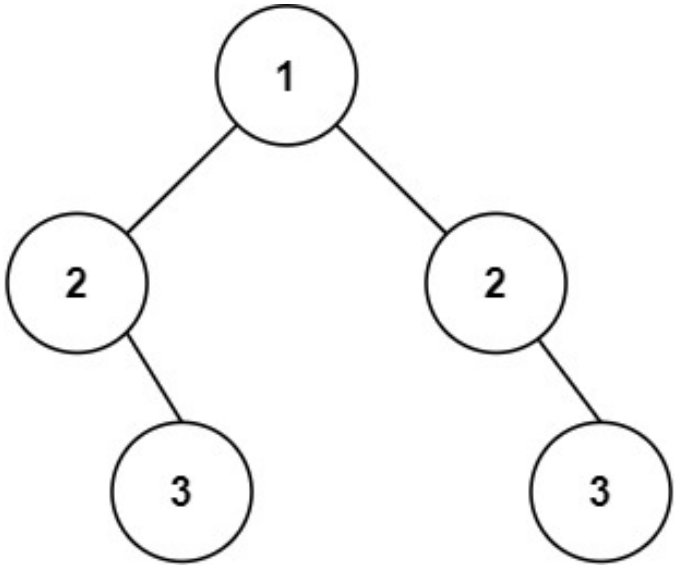
Identify any bug(s) within the given implementation and correct the code so that it successfully passes the provided test cases.

Example 1:



Input: root = [1,2,2,3,4,4,3]
Output: true

Example 2:



Input: root = [1,2,2,null,3,null,3]
Output: false

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_symmetric(root):
    def is_mirror(t1, t2):
        if not t1 and not t2:
```

```
            return True
        if not t1 or not t2:
            return False
        return (t1.val == t2.val and
                is_mirror(t1.left, t2.right) and
                is_mirror(t1.right, t2.left))

    return is_mirror(root, root)
def list_to_tree(lst):
    if not lst:
        return None

    root = TreeNode(lst[0])
    queue = deque([root])
    i = 1

    while i < len(lst):
        node = queue.popleft()

        # Handle the left child
        if lst[i] is not None:
            node.left = TreeNode(lst[i])
            queue.append(node.left)
        else:
            node.left = None
        i += 1

        # Handle the right child
        if i < len(lst):
            if lst[i] is not None:
                node.right = TreeNode(lst[i])
                queue.append(node.right)
            else:
                node.right = None
            i += 1

    return root

if __name__ == '__main__':
    input_data = sys.stdin.read().strip()
    input_list = ast.literal_eval(input_data)

    root = list_to_tree(input_list)

    result = is_symmetric(root)
    print(result)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Pass/Fail Case	Easy	Hidden	Success	5	0.0439 sec	10.9 KB

No comments.

5. What is the time complexity of mystery_function()?

Correct

Multiple Choice

Question description

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.value = val
        self.left_child = left
        self.right_child = right

def mystery_function(node):
    if not node:
        return 0

    left_result = mystery_function(node.left_child)
    right_result = mystery_function(node.right_child)

    return max(left_result, right_result) + 1
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ O(1)
- ○ O(log n)
- ● O(n) ✓
- ○ O(n log n)

No comments.

6. Which of the following best represents the values of the tree?

Incorrect

Multiple Choice

Question description

Given the following code, which of the following best represents the values of the tree with root `root`.

```
class TreeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

a = Node('a')
x = Node('x')
y = Node('y')
e = Node('e')
m = Node('m')
```

```
p = Node('p')

a.left = x
a.right = y

x.left = e
x.right = m

y.right = p

root = a
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ <pre> <code> root / \ / \ x y / \ \ e m p</code></pre>
- ○ <pre> <code> a / \ / \ x y / \ \ e m p</code></pre>   ✓
- ○ <pre> <code> a / \ / \ x y / \ \ m e p</code></pre>
- ● **<pre> <code> a / \ / \ x y / \ / e m p </code></pre>   **

No comments.

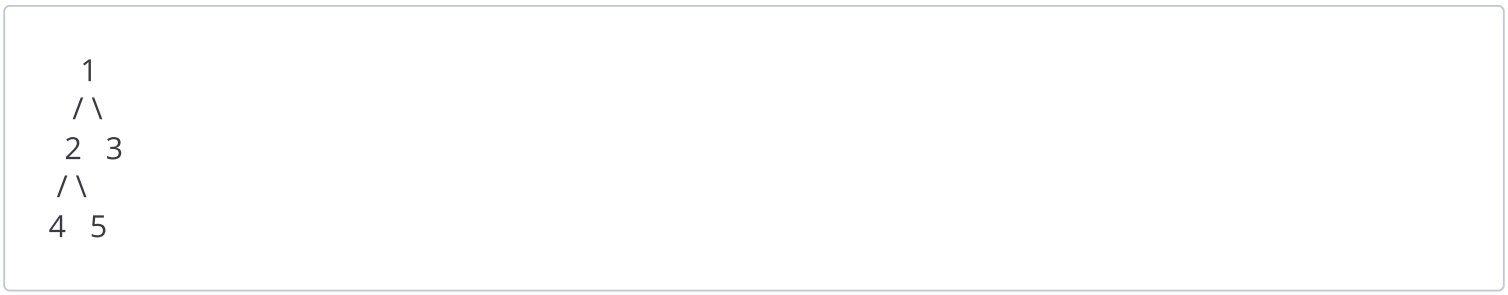
7. Which of the following arrays best represents its preorder traversal?

Correct

Multiple Choice

Question description

Given the following binary tree, which of the following arrays best represents its preorder traversal?



Candidate's Solution

Options: (Expected answer indicated with a tick)

- [ ] [4, 2, 5, 1, 3]
- [x] **[1, 2, 4, 5, 3]**
- [ ] [4, 5, 2, 3, 1]
- [ ] [1, 4, 2, 5, 3]

No comments.