Symmetric Tree (Mirror Image of itself)

15 June 2025 09:54 AM

Given a **binary tree**, the task is to check whether it is a **mirror** of itself.

Example:

Input: *root[] = [1, 2, 2, 3, 4, 4, 3]*

Output: *True*

Explanation: *Tree is mirror image of itself i.e. tree is symmetric.*

Input: *root[] = [1, 2, 2, N, 3, N, 3]*

Output: *False*

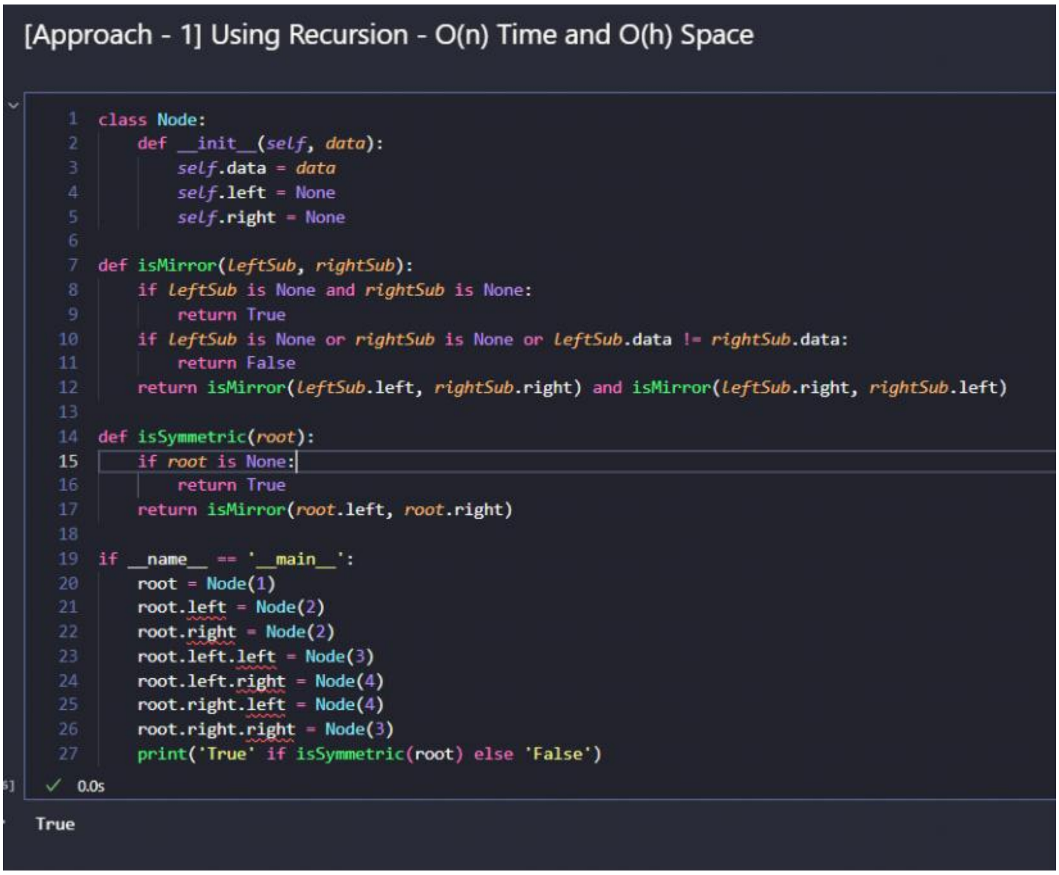
Explanation: *Tree is not mirror image of itself i.e. tree is not symmetric.*

Try it on GfG Practice

Table of Content

[Approach - 1] Using Recursion - O(n) Time and O(h) Space

The idea is to ***recursively*** *compare the* ***left*** *and* ***right*** *subtrees of the root. For the tree to be* ***symmetric****, the* ***root*** *values of the* ***left*** *and* ***right*** *subtrees must match, and their corresponding* ***children*** *must also be mirrors.*



[Approach - 1] Using Recursion - O(n) Time and O(h) Space

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def isMirror(leftSub, rightSub):
    if leftSub is None and rightSub is None:
        return True
    if leftSub is None or rightSub is None or leftSub.data != rightSub.data:
        return False
    return isMirror(leftSub.left, rightSub.right) and isMirror(leftSub.right, rightSub.left)

def isSymmetric(root):
    if root is None:
        return True
    return isMirror(root.left, root.right)

if __name__ == '__main__':
    root = Node(1)
    root.left = Node(2)
    root.right = Node(2)
    root.left.left = Node(3)
    root.left.right = Node(4)
    root.right.left = Node(4)
    root.right.right = Node(3)
    print('True' if isSymmetric(root) else 'False')
```

✓ 0.0s

```
True
```

```
# Python program to check if a given
# Binary Tree is symmetric

class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
```

```
# Recursive helper function to check if two subtrees are mirror images
def isMirror(leftSub, rightSub):

    # Both are null, so they are mirror images
    if leftSub is None and rightSub is None:
        return True

    # One of them is null, so they aren't mirror images
    if leftSub is None or rightSub is None or leftSub.data != rightSub.data:
        return False

    # Check if the subtrees are mirrors
    return isMirror(leftSub.left, rightSub.right) and \
           isMirror(leftSub.right, rightSub.left)

def isSymmetric(root):

    # If tree is empty, it's symmetric
    if root is None:
        return True

    # Check if the left and right subtrees are mirrors of each other
    return isMirror(root.left, root.right)

if __name__ == "__main__":
    # Creating a sample symmetric binary tree
    #        1
    #       / \
    #      2   2
    #     / \ / \
    #    3  4 4  3
    print("true" if isSymmetric(root) else "false")
```

Output

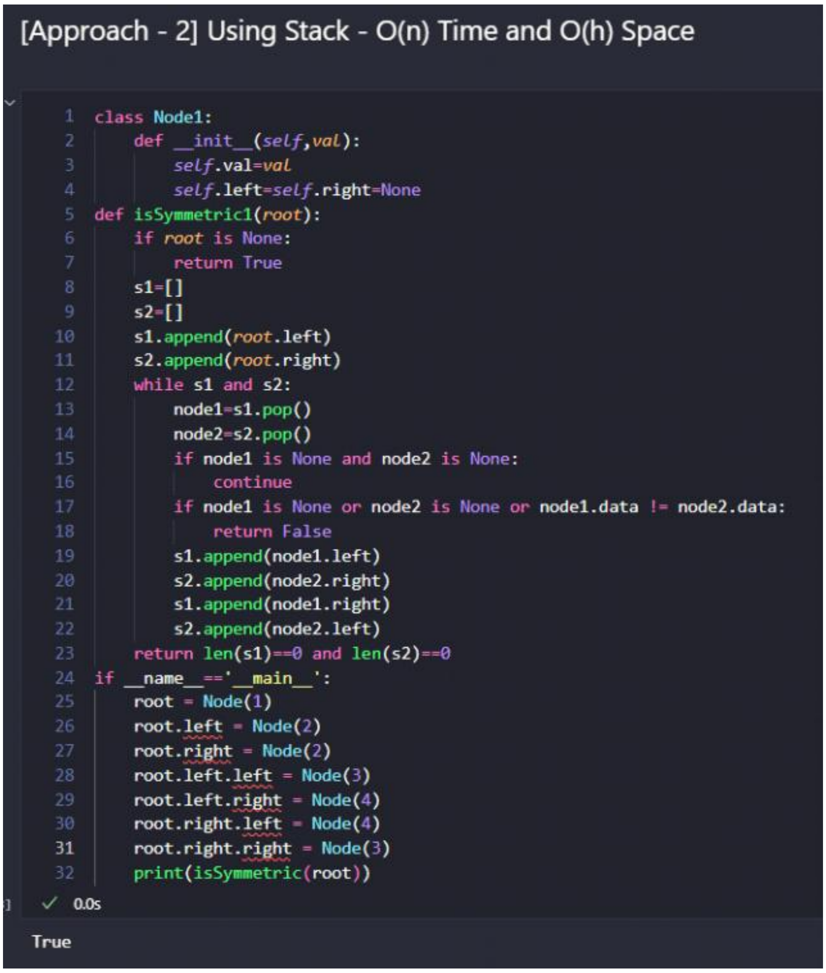
```
true
```

[Approach - 2] Using Stack - O(n) Time and O(h) Space

The idea is to use ***two*** *[**stack**](stack) to check if a binary tree is* ***symmetric****. One stack is for the* ***left*** *side of the tree, and the other is for the* ***right*** *side. By comparing nodes from both* ***stack*** *at each level, we can check if the* ***left*** *and* ***right*** *sides are mirror images of each other.*

Step-by-Step Implementation:

- Create a **two stacks**, say **s1** and **s2** and push the **left child** of the root node in **s1** and **right child** of the root node into **s2**.
- While both the stack are not empty, repeat the following steps:
  - Pop two nodes from the stack, say **node1** and **node2**.
  - If both **node1** and **node2** are null, continue to the next iteration.
  - If one of the nodes is null and the other is not, **return false** as it is not a mirror.
  - If both nodes are not null, compare their values. If they are **not equa**l, return false.
  - Push the left child of node1 and the right child of node2 onto the stack.
  - Push the right child of node1 and the left child of node2 onto the stack.
- If the loop completes successfully without returning false, **return true** as it is a mirror.



[Approach - 2] Using Stack - O(n) Time and O(h) Space

```
class Node1:
    def __init__(self,val):
        self.val=val
        self.left=self.right=None
def isSymmetric1(root):
    if root is None:
        return True
    s1=[]
    s2=[]
    s1.append(root.left)
    s2.append(root.right)
    while s1 and s2:
        node1=s1.pop()
        node2=s2.pop()
        if node1 is None and node2 is None:
            continue
        if node1 is None or node2 is None or node1.data != node2.data:
            return False
        s1.append(node1.left)
        s2.append(node2.right)
        s1.append(node1.right)
        s2.append(node2.left)
    return len(s1)==0 and len(s2)==0
if __name__=='__main__':
    root = Node(1)
    root.left = Node(2)
    root.right = Node(2)
    root.left.left = Node(3)
    root.left.right = Node(4)
    root.right.left = Node(4)
    root.right.right = Node(3)
    print(isSymmetric(root))
```

```
✓ 0.0s

True
```

```
# Python program to check if a given
# Binary Tree is symmetric

class Node:
    def __init__(self, val):
        self.data = val
        self.left = self.right = None

# Function to check if the binary tree is symmetric
def isSymmetric(root):
    if root is None:
        return True

    # Two stacks to store nodes for comparison
    s1 = []
    s2 = []

    # Initialize the stacks with the
    # left and right subtrees
    s1.append(root.left)
    s2.append(root.right)

    while s1 and s2:

        # Get the current pair of nodes
        node1 = s1.pop()
        node2 = s2.pop()

        # If both nodes are null, continue to the next pair
        if node1 is None and node2 is None:
            continue

        # If one node is null and the other is not,
        # or the nodes' data do not match
        # then the tree is not symmetric
        if node1 is None or node2 is None or node1.data != node2.data:
            return False
        # Push children of node1 and node2 in opposite order
        # Push left child of node1 and right child of node2
        s1.append(node1.left)
        s2.append(node2.right)

        # Push right child of node1 and left child of node2
        s1.append(node1.right)
        s2.append(node2.left)

    # If both stacks are empty, the tree is symmetric
    return len(s1) == 0 and len(s2) == 0

if __name__ == "__main__":

    # Creating a sample symmetric binary tree
    #        1
    #       / \
    #      2   2
    #     / \ / \
    #    3  4 4  3
    root = Node(1)
    root.left = Node(2)
    root.right = Node(2)
    root.left.left = Node(3)
    root.left.right = Node(4)
    root.right.left = Node(4)
    root.right.right = Node(3)

    print(isSymmetric(root))
```

Output

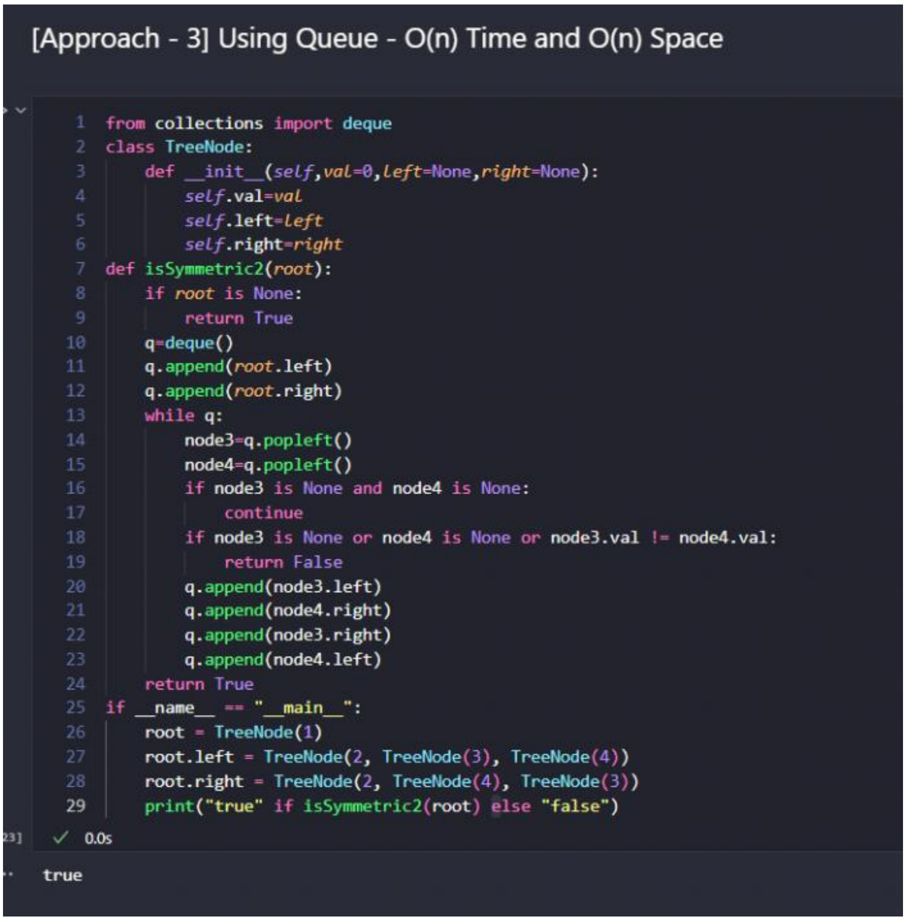
```
true
```

[Approach - 3] Using Queue - O(n) Time and O(n) Space

*The basic idea is to check if the left and right subtrees of the **root** node are **mirror images** of each other. To do this, we perform a level-order traversal of the binary tree using a queue. Initially**,** we push the root node into the queue **twice.** We dequeue two nodes at a time from the **front** of the queue and check if they are mirror images of each other.*

Step-by-Step implementation:

- If the root node is **NULL**, **return true** as an empty binary tree is considered **symmetric**.
- Create a **queue** and push the left and right child of **root** node into the queue.
- While the queue is not empty, **dequeue** two nodes at a time, one for the **left subtree** and one for the **right subtree**.
  - If both the **left** and **right** nodes are **NULL**, continue to the next iteration as the subtrees are considered mirror images of each other.
  - If either the **left** or **right** node is **NULL**, or their data is not equal, **return false** as they are not mirror images of each other.
  - Push the **left** and **right** nodes of the left subtree into the **queue**, followed by the right and left nodes of the right subtree into the queue.
- If the queue becomes **empty** and we have not returned false till now, **return true** as the binary tree is symmetric.



[Approach - 3] Using Queue - O(n) Time and O(n) Space

```python
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 isSymmetric2(root):
    if root is None:
        return True
    q=deque()
    q.append(root.left)
    q.append(root.right)
    while q:
        node3=q.popleft()
        node4=q.popleft()
        if node3 is None and node4 is None:
            continue
        if node3 is None or node4 is None or node3.val != node4.val:
            return False
        q.append(node3.left)
        q.append(node4.right)
        q.append(node3.right)
        q.append(node4.left)
    return True
if __name__ == "__main__":
    root = TreeNode(1)
    root.left = TreeNode(2, TreeNode(3), TreeNode(4))
    root.right = TreeNode(2, TreeNode(4), TreeNode(3))
    print("true" if isSymmetric2(root) else "false")
```

```
23]  ✓  0.0s

true
```

```python
from collections import deque

# Definition for a binary tree node
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# Function to check if the binary tree is symmetric
def isSymmetric(root):
    if root is None:
        return True

    # Use a queue to store nodes for comparison
    q = deque()

    # Initialize the queue with the left and right subtrees
    q.append(root.left)
    q.append(root.right)

    while q:
        node1 = q.popleft()
        node2 = q.popleft()

        # If both nodes are None, continue
        if node1 is None and node2 is None:
            continue

        # If only one is None or values don't match, it's not symmetric
        if node1 is None or node2 is None or node1.val != node2.val:
            return False

        # Enqueue children in opposite order
        q.append(node1.left)
        q.append(node2.right)
        q.append(node1.right)
        q.append(node2.left)

    return True
```

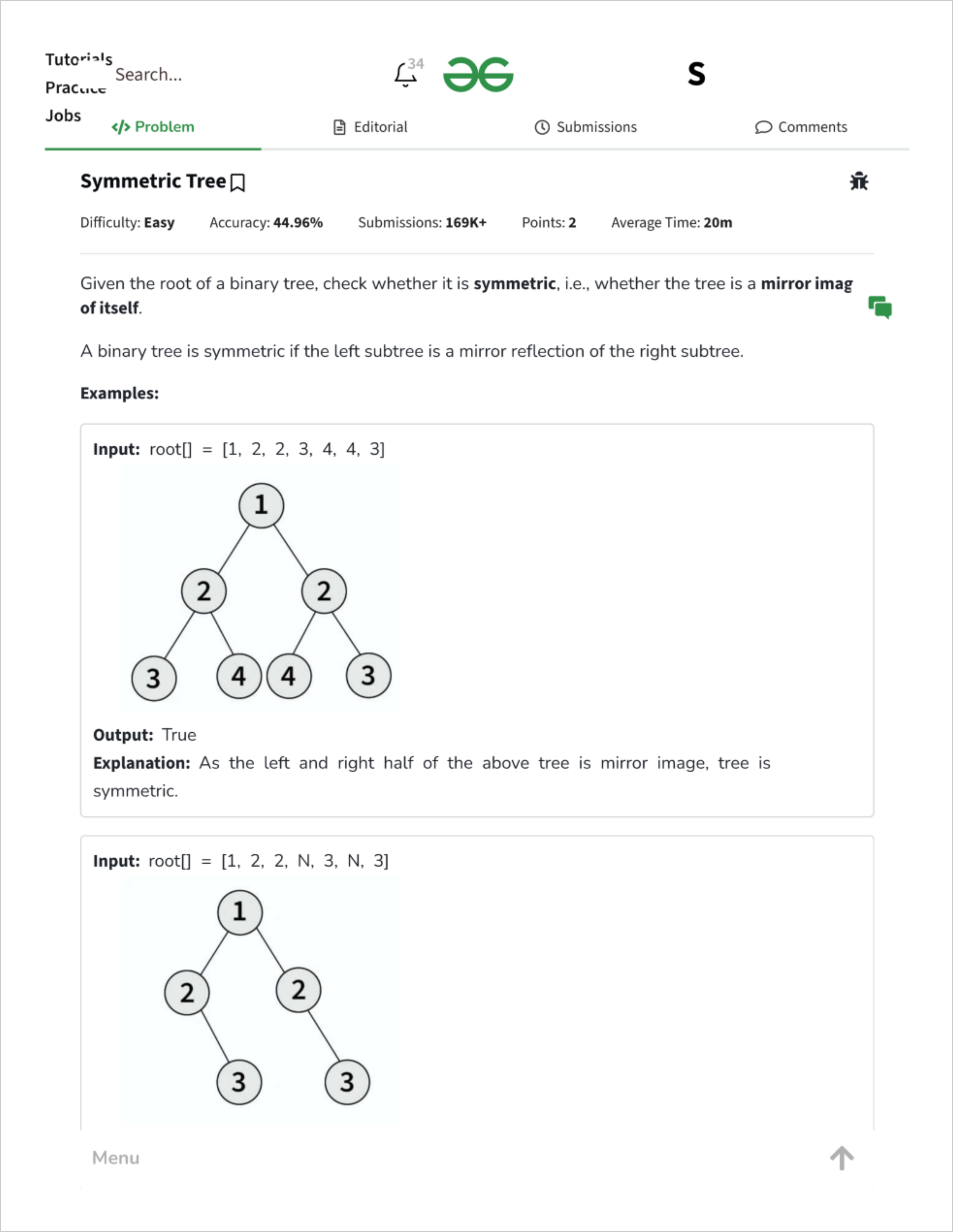
```
if __name__ == "__main__":

    # Example symmetric tree
    #        1
    #       /   \
    #     2     2
    #    / \   / \
    #   3   4 4   3

    root = TreeNode(1)
    root.left = TreeNode(2, TreeNode(3), TreeNode(4))
    root.right = TreeNode(2, TreeNode(4), TreeNode(3))

    print("true" if isSymmetric(root) else "false")
```

Output

```
True
```

completed

From <https://www.geeksforgeeks.org/dsa/symmetric-tree-tree-which-is-mirror-image-of-itself/>



Symmetric Tree

Difficulty: **Easy** Accuracy: **44.96%** Submissions: **169K+** Points: **2** Average Time: **20m**

Given the root of a binary tree, check whether it is **symmetric**, i.e., whether the tree is a **mirror imag of itself**.

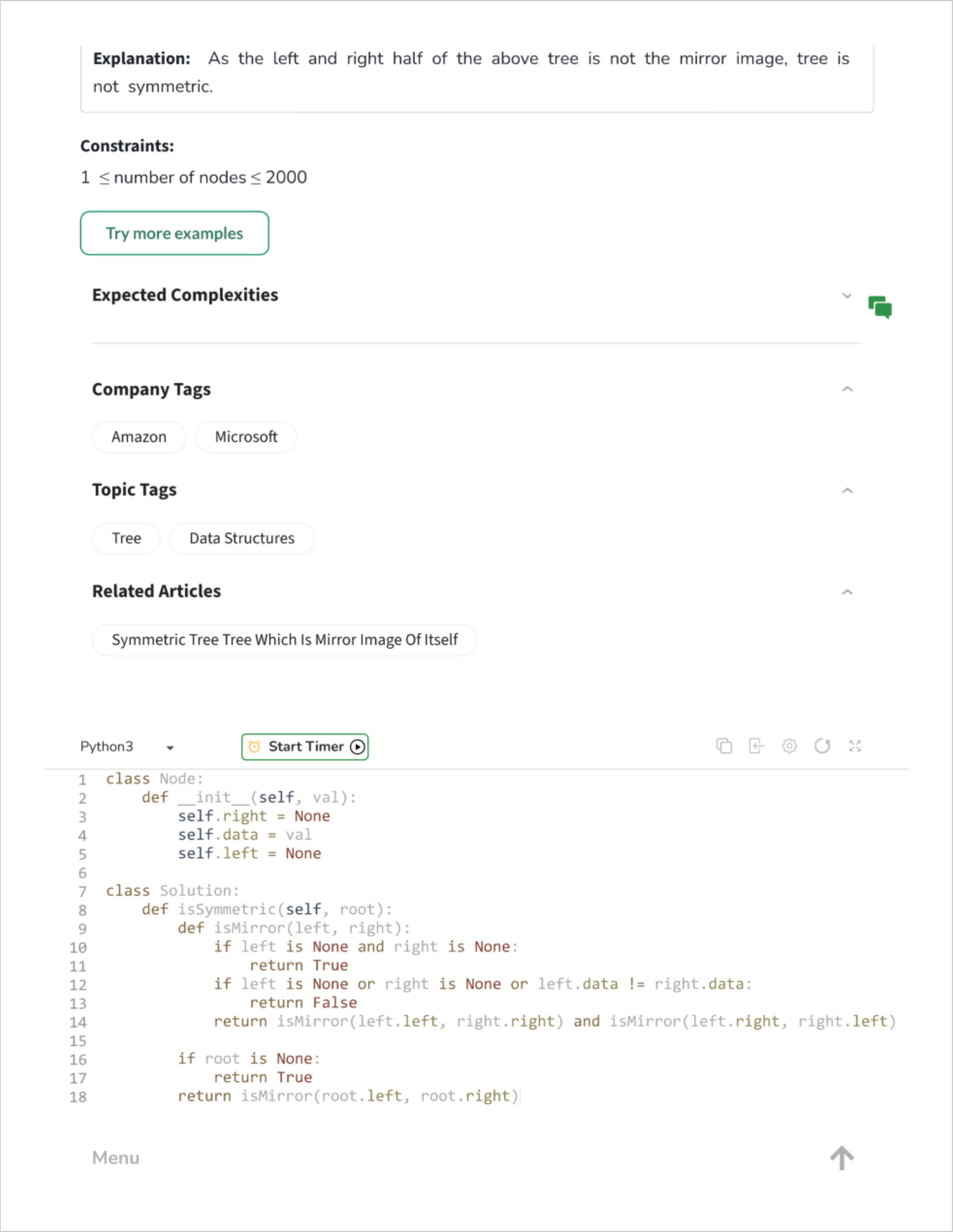
A binary tree is symmetric if the left subtree is a mirror reflection of the right subtree.

Examples:

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

Output: True
Explanation: As the left and right half of the above tree is mirror image, tree is symmetric.

Input: root[] = [1, 2, 2, N, 3, N, 3]



Explanation: As the left and right half of the above tree is not the mirror image, tree is not symmetric.

Constraints:

1 $\leq$ number of nodes $\leq$ 2000

Expected Complexities

Company Tags

Amazon

Microsoft

Topic Tags

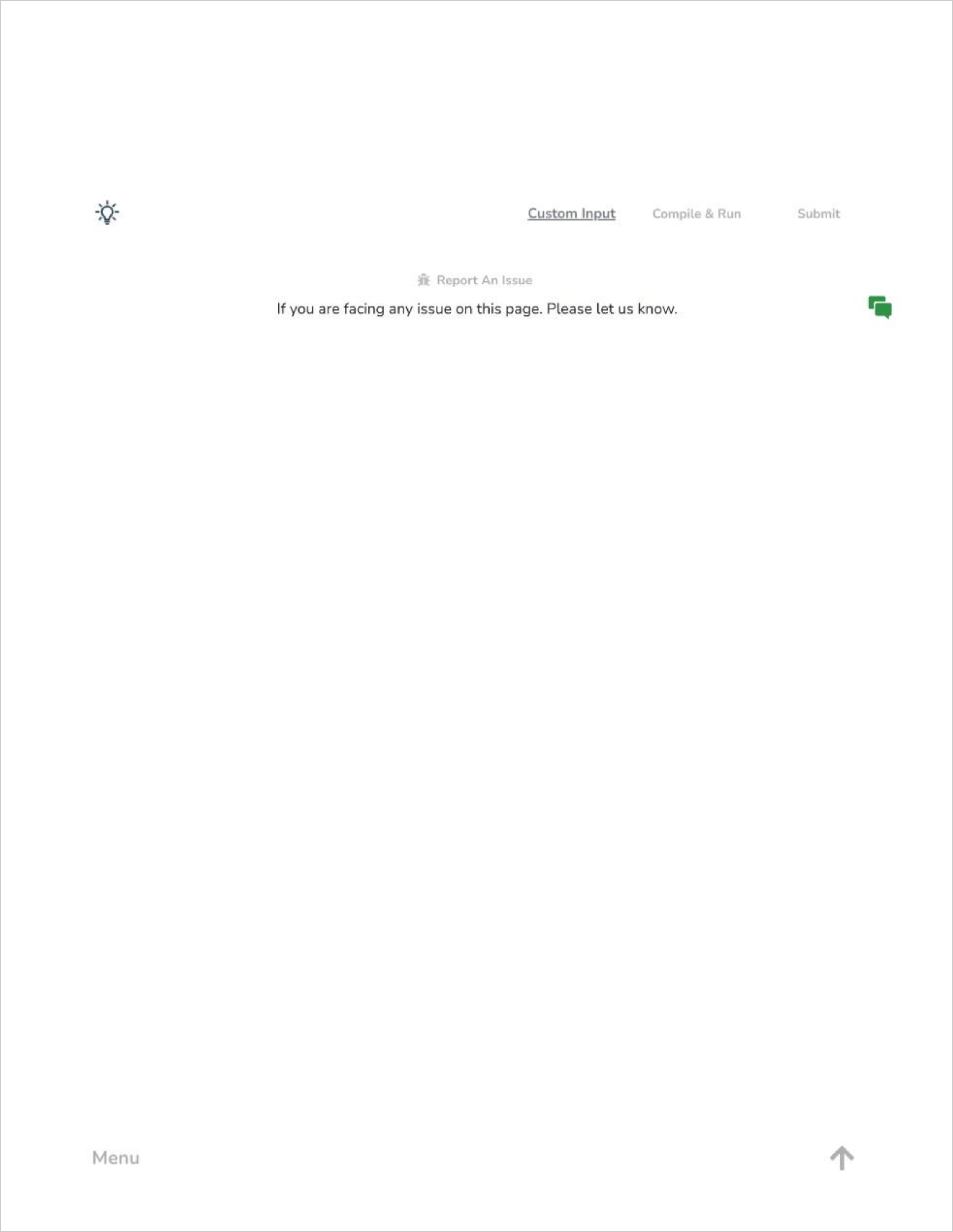
Tree

Data Structures

Related Articles

Symmetric Tree Tree Which Is Mirror Image Of Itself

```python
class Node:
    def __init__(self, val):
        self.right = None
        self.data = val
        self.left = None

class Solution:
    def isSymmetric(self, root):
        def isMirror(left, right):
            if left is None and right is None:
                return True
            if left is None or right is None or left.data != right.data:
                return False
            return isMirror(left.left, right.right) and isMirror(left.right, right.left)

        if root is None:
            return True
        return isMirror(root.left, root.right)
```



If you are facing any issue on this page. Please let us know.