Experiment 3

Student Name: Nitil Jakhar
UID: 22BCS17300
Branch: CSE
Section/Group: NTPP_IOT-602/A
Semester: 6th
Date of Performance: 3/02/2
Subject: AP 2

1. **Aim: Divide and Conquer**
2. **Objective:**
   1. Longest Nice Substring
   2. Search 2d matrix 2
3. **Code:**

1. **Longest Nice Substring:**

```
from typing import Optional

# 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

class Solution:
   def maxDepth(self, root: Optional[TreeNode]) -> int:
      if not root:
         return 0
      left_depth = self.maxDepth(root.left)
      right_depth = self.maxDepth(root.right)
      return max(left_depth, right_depth) + 1
```

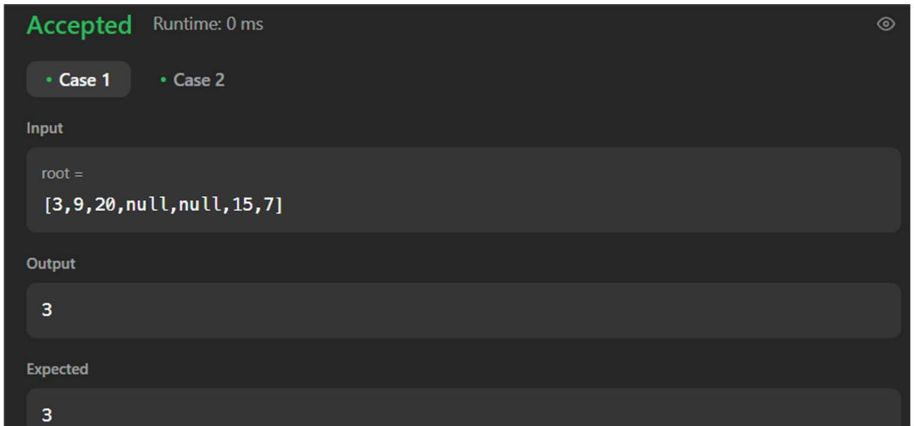
2. **Search 2d matrix 2:**

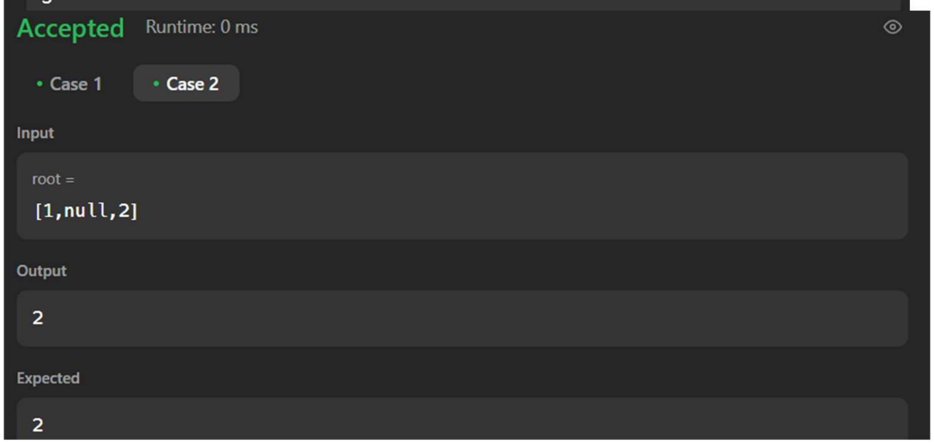
```
from typing import List, Optional

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

```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
        if not nums:
            return None

        mid = len(nums) // 2
        root = TreeNode(nums[mid])

        root.left = self.sortedArrayToBST(nums[:mid])
        root.right = self.sortedArrayToBST(nums[mid+1:])

        return root

    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        def isMirror(t1: Optional[TreeNode], t2: Optional[TreeNode]) ->
bool:
            if not t1 and not t2:
                return True
            if not t1 or not t2:
                return False
            return (t1.val == t2.val and
                    isMirror(t1.right, t2.left) and
                    isMirror(t1.left, t2.right))

        return isMirror(root, root)
```

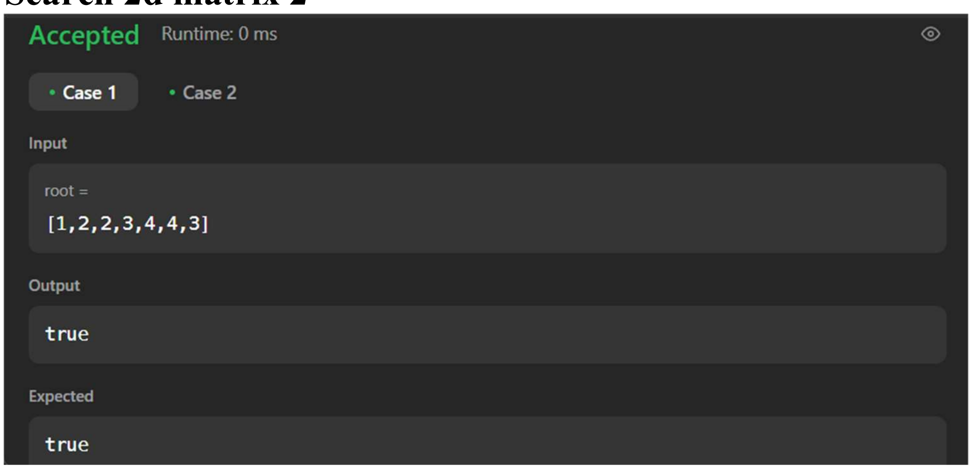
4. Output:

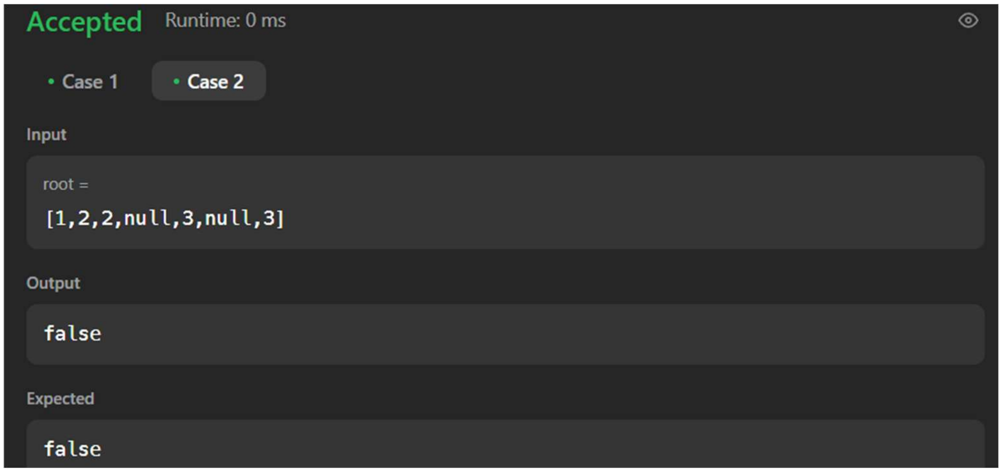
1. **Longest Nice Substring:**





2. **Search 2d matrix 2**





5. Learning Outcome

1) Learn how to traverse and manipulate binary trees using recursive methods, such as building a height-balanced BST and checking for symmetry.
2) Gain knowledge of how sorted arrays can be converted into balanced BSTs, ensuring efficient search operations (O(log n) complexity).
3) Develop an understanding of tree traversal techniques, including pre-order, in-order, and mirrored traversal for checking symmetry.
4) Learn to solve tree problems using both recursive and iterative methods, improving problem-solving skills and adaptability in coding interviews.
5) Explore how different implementations impact the efficiency of tree algorithms, particularly in recursive vs. iterative solutions.