**ADVANCE CODING - 2**

**K.Revanth**

**VU21CSEN0300254**

**CODE:**

from typing import List

class Solution:

def trap(self, height: List[int]) -> int: if not height:

return 0

left, right = 0, len(height) - 1 left\_max, right\_max = 0, 0

water = 0

while left < right:

if height[left] < height[right]: if height[left] >= left\_max:

left\_max = height[left] else:

water += left\_max - height[left] left += 1

else:

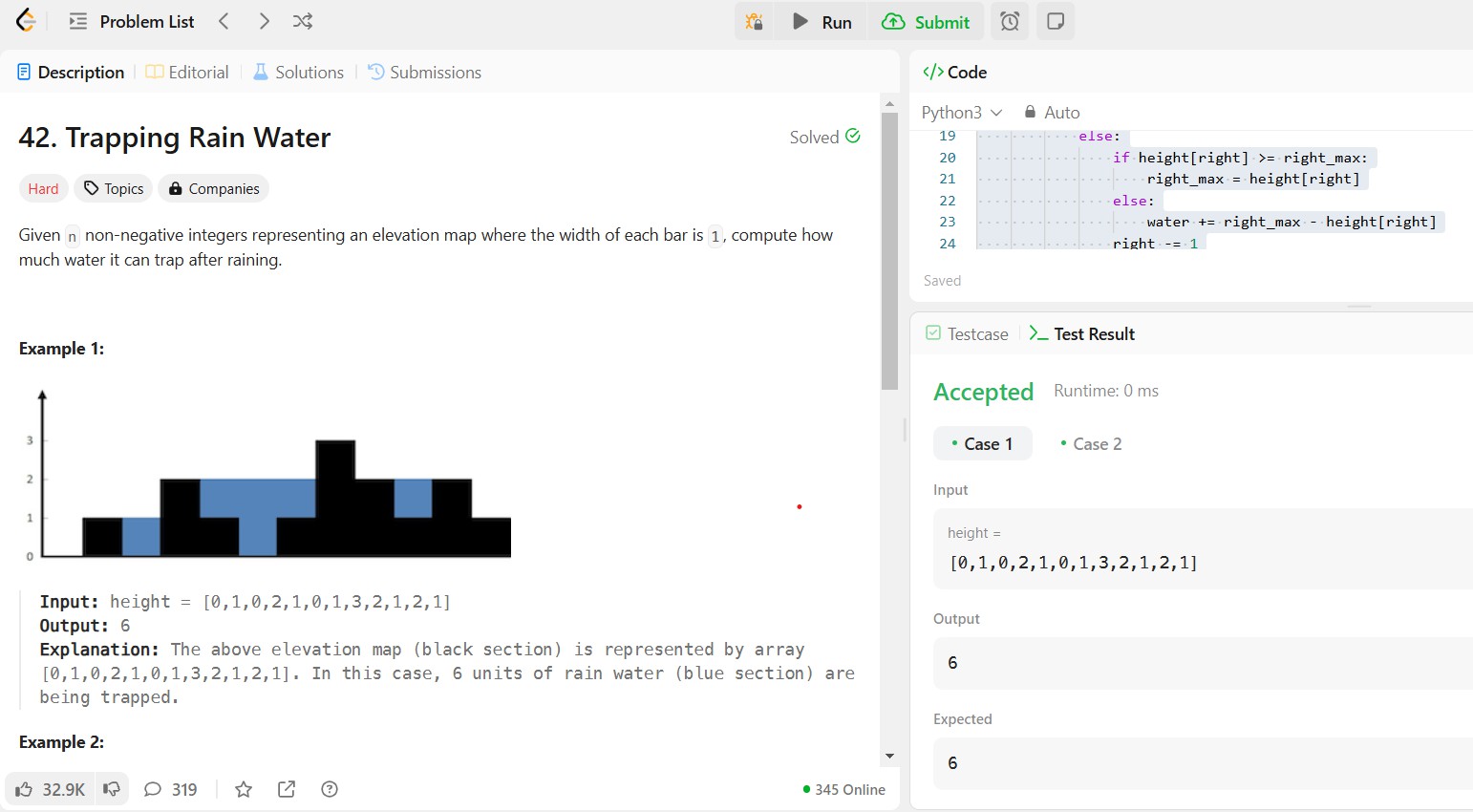
if height[right] >= right\_max: right\_max = height[right]

else:

water += right\_max - height[right] right -= 1

return water

**OUTPUT:**



**CODE:**

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 flatten(self, root: Optional[TreeNode]) -> None: """

Do not return anything, modify root in-place instead. """

if not root:

return

# Helper function to recursively flatten the tree def flatten\_tree(node):

if not node:

return None

# Flatten the left and right subtrees left\_tail = flatten\_tree(node.left) right\_tail = flatten\_tree(node.right)

# If there is a left subtree, attach it to the right of the current node

if node.left:

if left\_tail:

left\_tail.right = node.right # Connect the tail of the left subtree to the start of the right subtree

node.right = node.left # Move the left subtree to the right node.left = None # Set the left child to None

node

# Return the tail of the flattened tree

return right\_tail if right\_tail else left\_tail if left\_tail else

flatten\_tree(root)

**OUTPUT:**

