Our Solution(s) Run Code

Solution 1

```python
# Copyright © 2020 AlgoExpert, LLC. All rights reserved.

# O(wh) time | O(wh) space
def riverSizes(matrix):
    sizes = []
    visited = [[False for value in row] for row in matrix]
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if visited[i][j]:
                continue
            traverseNode(i, j, matrix, visited, sizes)
    return sizes


def traverseNode(i, j, matrix, visited, sizes):
    currentRiverSize = 0
    nodesToExplore = [[i, j]]
    while len(nodesToExplore):
        currentNode = nodesToExplore.pop()
        i = currentNode[0]
        j = currentNode[1]
        if visited[i][j]:
            continue
        visited[i][j] = True
        if matrix[i][j] == 0:
            continue
        currentRiverSize += 1
        unvisitedNeighbors = getUnvisitedNeighbors(i, j, matrix,
        for neighbor in unvisitedNeighbors:
            nodesToExplore.append(neighbor)
    if currentRiverSize > 0:
        sizes.append(currentRiverSize)

```

Your Solutions Run Code

Solution 1 Solution 2 Solution 3

```python
def riverSizes(matrix):
    # Write your code here.
    pass

```

Our Tests

Custom Output Submit Code

```
import unittest


class TestProgram(unittest.TestCase):
    def test_case_0(self):
        testInput = [80]
        expected = 2
        self.assertEqual(canTest(program.flowerGame(testInput)),
                         expected)

    def test_case_1(self):
        testInput = [30]
        expected = 0
        self.assertEqual(canTest(program.flowerGame(testInput)),
                         expected)

    def test_case_2(self):
        testInput = [1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0]
        expected = [1, 2, 3]
        self.assertEqual(canTest(program.flowerGame(testInput)),
                         expected)

    def test_case_3(self):
        testInput = [1, 0, 0, 1], [1, 0, 1, 0], [0, 0, 1, 0], [
```

Run or submit code when you're ready.