Our Solution(s) Run Code

Solution 1 Solution 2

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

type BinaryTree struct {
  Value int

  Left  *BinaryTree
  Right *BinaryTree
}

// O(n) time | O(d) space - where n is the number of nodes in the Bin
// and d is the depth (height) of the Binary Tree
func FlattenBinaryTree(root *BinaryTree) *BinaryTree {
  leftMost, _ := flattenTree(root)
  return leftMost
}

func flattenTree(node *BinaryTree) (leftMost, rightMost *BinaryTree) {
  leftMost = node
  if node.Left != nil {
    leftSubtreeLeftMost, leftSubtreeRightMost := flattenTree(node.Left
    connectNodes(leftSubtreeRightMost, node)
    leftMost = leftSubtreeLeftMost
  }

  rightMost = node
  if node.Right != nil {
    rightSubtreeLeftMost, rightSubtreeRightMost := flattenTree(node.Ri
    connectNodes(node, rightSubtreeLeftMost)
    rightMost = rightSubtreeRightMost
  }
  return leftMost, rightMost
```

Your Solutions Run Code

Solution 1 Solution 2 Solution 3

```go
package main

// This is the class of the input root. Do not edit it.
type BinaryTree struct {
  Value int

  Left  *BinaryTree
  Right *BinaryTree
}

func FlattenBinaryTree(root *BinaryTree) *BinaryTree {
  // Write your code here.
  return nil
}

```

Our Tests

Custom Output Submit Code

```
    return &BinaryTree{Value: value}
}

func (tree *BinaryTree) insertLevelOrder(arr []int) *BinaryTree {
    queue := []*BinaryTree{tree}
    for len(queue) > 0 {
        var current *BinaryTree
        queue, current = queue[1:], queue[0]
        if current.Left == nil {
            current.Left = NewBinaryTree(value)
            return tree
        }
        queue = append(queue, current.Left)

        if current.Right == nil {
```

Run or submit code when you're ready.