Our Solution(s) Run Code

Solution 1

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

public class Program {
  // O(n) time | O(d) space - where n is the number of nodes in
  // the Binary Tree and d is the depth (height) of the Binary Tree
  public static BinaryTree RightSiblingTree(BinaryTree root) {
    mutate(root, null, false);
    return root;
  }

  public static void mutate(BinaryTree node, BinaryTree parent, bool
    if (node == null) return;

    var left = node.left;
    var right = node.right;
    mutate(left, node, true);
    if (parent == null) {
      node.right = null;
    } else if (isLeftChild) {
      node.right = parent.right;
    } else{
      if (parent.right == null) {
        node.right = null;
      } else {
        node.right = parent.right.left;
      }
    }
    mutate(right, node, false);
  }

  public class BinaryTree {
    public int value;
    public BinaryTree left = null;
```

Your Solutions Run Code

Solution 1 Solution 2 Solution 3

```
public class Program {
  public static BinaryTree RightSiblingTree(BinaryTree root) {
    // Write your code here.
    return root;
  }

  // This is the class of the input root. Do not edit it.
  public class BinaryTree {
    public int value;
    public BinaryTree left = null;
    public BinaryTree right = null;

    public BinaryTree(int value) {
      this.value = value;
    }
  }
}

```

Our Tests

Custom Output Submit Code

```
    Program.BinaryTree solutionRoot = Program.RightSideOrderTree(root);
    List<int> actual = getPreOrder(solutionRoot);
    var expected = new List<int>
    {
      1
    };
    this.AssertTrue(expected.SequenceEqual(actual));
  }

  [Test]
  public void Test2Case()
  {
    Program.BinaryTree root = new Program.BinaryTree(1);
    InsertInto(root, new int[] {2});
    Program.BinaryTree solutionRoot = Program.RightSideOrderTree(root);
    List<int> actual = getPreOrder(solutionRoot);
    var expected = new List<int>
```

Run or submit code when you're ready.