```
// 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 FlattenBinaryTree(BinaryTree root) {
    flattenTree(root);
    return getLeftMost(root);
  }

  public static BinaryTree[] flattenTree(BinaryTree node) {
    BinaryTree leftMost;
    BinaryTree rightMost;

    if (node.left == null) {
      leftMost = node;
    } else {
      BinaryTree[] leftAndRightMostNodes = flattenTree(node.left);
      connectNodes(leftAndRightMostNodes[1], node);
      leftMost = leftAndRightMostNodes[0];
    }

    if (node.right == null) {
      rightMost = node;
    } else {
      BinaryTree[] leftAndRightMostNodes = flattenTree(node.right);
      connectNodes(node, leftAndRightMostNodes[0]);
      rightMost = leftAndRightMostNodes[1];
    }

    return new BinaryTree[] {leftMost, rightMost};
  }

  public static void connectNodes(BinaryTree left, BinaryTree right) {
    left.right = right;
    right.left = left;
  }

  public static BinaryTree getLeftMost(BinaryTree node) {
    while (node.left != null) {
      node = node.left;
    }
    return node;
  }

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

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

```