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

public class Program {
  public class BST {
    public int value;
    public BST left;
    public BST right;

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

    // Average: O(log(n)) time | O(1) space
    // Worst: O(n) time | O(1) space
    public BST Insert(int value) {
      BST currentNode = this;
      while (true) {
        if (value < currentNode.value) {
          if (currentNode.left == null) {
            BST newNode = new BST(value);
            currentNode.left = newNode;
            break;
          } else {
            currentNode = currentNode.left;
          }
        } else {
          if (currentNode.right == null) {
            BST newNode = new BST(value);
            currentNode.right = newNode;
            break;
          } else {
            currentNode = currentNode.right;
          }
        }
      }
      return this;
    }

    // Average: O(log(n)) time | O(1) space
    // Worst: O(n) time | O(1) space
    public bool Contains(int value) {
      BST currentNode = this;
      while (currentNode != null) {
        if (value < currentNode.value) {
          currentNode = currentNode.left;
        } else if (value > currentNode.value) {
          currentNode = currentNode.right;
        } else {
          return true;
        }
      }
      return false;
    }

    // Average: O(log(n)) time | O(1) space
    // Worst: O(n) time | O(1) space
    public BST Remove(int value) {
      Remove(value, null);
      return this;
    }

    public void Remove(int value, BST parentNode) {
      BST currentNode = this;
      while (currentNode != null) {
        if (value < currentNode.value) {
          parentNode = currentNode;
          currentNode = currentNode.left;
        } else if (value > currentNode.value) {
          parentNode = currentNode;
          currentNode = currentNode.right;
        } else {
          if (currentNode.left != null && currentNode.right != null) {
            currentNode.value = currentNode.right.getMinValue();
            currentNode.right.Remove(currentNode.value,
              currentNode);
          } else if (parentNode == null) {
            if (currentNode.left != null) {
              currentNode.value = currentNode.left.value;
              currentNode.right = currentNode.left.right;
              currentNode.left = currentNode.left.left;
            } else if (currentNode.right != null) {
              currentNode.value = currentNode.right.value;
              currentNode.left = currentNode.right.left;
              currentNode.right = currentNode.right.right;
            } else {
              // This is a single-node tree; do nothing.
            }
          } else if (parentNode.left == currentNode) {
            parentNode.left = currentNode.left !=
              null ? currentNode.left :
```



```
public class Program {
  public class BST {
    public int value;
    public BST left;
    public BST right;

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

    public BST Insert(int value) {
      // Write your code here.
      // Do not edit the return statement of this method.
      return this;
    }

    public bool Contains(int value) {
      // Write your code here.
      return false;
    }

    public BST Remove(int value) {
      // Write your code here.
      // Do not edit the return statement of this method.
      return this;
    }
  }
}

```

```
              currentNode.right;
          } else if (parentNode.right == currentNode) {
            parentNode.right = currentNode.left !=
              null ? currentNode.left :
              currentNode.right;
          }
          break;
        }
      }
    }

    public int getMinValue() {
      if (left == null) {
        return value;
      } else {
        return left.getMinValue();
      }
    }
  }
}

```

Run or submit code when you're ready.