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

class BST {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }

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

  // Average: O(log(n)) time | O(1) space
  // Worst: O(n) time | O(1) space
  contains(value) {
    let 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
  remove(value, parentNode = null) {
    let 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 : current
        } else if (parentNode.right === currentNode) {
          parentNode.right = currentNode.left !== null ? currentNode.left : curre
        }
        break;
      }
    }
    return this;
  }

  getMinValue() {
    let currentNode = this;
    while (currentNode.left !== null) {
```



```
// Do not edit the class below except for
// the insert, contains, and remove methods.
// Feel free to add new properties and methods
// to the class.
class BST {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }

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

  contains(value) {
    // Write your code here.
  }

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

// Do not edit the line below.
exports.BST = BST;

```

```
      currentNode = currentNode.left;
    }
    return currentNode.value;
  }
}

exports.BST = BST;

```

Run or submit code when you're ready.