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

class Program {
    class BST {
        var value: Int
        var left: BST?
        var right: BST?

        init(value: Int) {
            self.value = value
            left = nil
            right = nil
        }

        // Average: O(log(n)) time | O(1) space
        // Worst: O(n) time | O(1) space
        func insert(value: Int) -> BST {
            var currentNode: BST? = self

            while true {
                if let node = currentNode, value < node.value {
                    if node.left === nil {
                        node.left = BST(value: value)
                        break
                    } else {
                        currentNode = node.left
                    }
                } else if let node = currentNode {
                    if node.right === nil {
                        node.right = BST(value: value)
                        break
                    } else {
                        currentNode = node.right
                    }
                }
            }

            return self
        }

        // Average: O(log(n)) time | O(1) space
        // Worst: O(n) time | O(1) space
        func contains(value: Int) -> Bool {
            var currentNode: BST? = self

            while currentNode !== nil {
                if let node = currentNode, value < node.value {
                    currentNode = node.left
                } else if let node = currentNode, value > node.value {
                    currentNode = node.right
                } else {
                    return true
                }
            }

            return false
        }

        // Average: O(log(n)) time | O(1) space
        // Worst: O(n) time | O(1) space
        func remove(value: Int, parentNode: BST?) -> BST {
            var currentNode: BST? = self
            var parentNode: BST? = parentNode
            while let node = currentNode {
                if value < node.value {
                    parentNode = node
                    currentNode = node.left
                } else if value > node.value {
                    parentNode = node
                    currentNode = node.right
                } else {
                    if let left = node.left, let right = node.right {
                        node.value = right.getMinValue()
                        right.remove(value: node.value, parentNode: node)
                    } else if parentNode === nil {
                        if let left = node.left {
                            node.value = left.value
                            node.right = left.right
                            node.left = left.left
                        } else if let right = node.right {
                            node.value = right.value
                            node.left = right.left
                            node.right = right.right
                        } else {
                            // This is a single-node tree; do nothing.
```