/ COSC 2007 Section O & LAB O * Author: Trevor Maliro * Student ID: 239498690 * Date: 2023/12/04 * * Lab 4: Binary Search Tree * Pseudocode/Algorithm: * - A class Node to represent each node of the binary tree. * - A class BinarySearchTree to represent the binary tree. * - A method insert to construct the binary tree from the array. * - A method inorder to perform the inorder traversal of the binary tree. * - A method search to search for a key in the binary tree. * - A method delete to delete a key from the binary tree. * * * Code: * No input necessary * Output: The algorithm works as expected. */

/** * This class represents a Binary Search Tree data structure. */ public class BinarySearchTree { private Node root;

```
/**
 * Constructs an empty Binary Search Tree.
 */
public BinarySearchTree() {
    root = null;
}

/**
 * Inserts a new key into the Binary Search Tree.
 *
 * @param key the key to be inserted
 */
public void insert(int key) {
    root = insertRec(root, key);
}

/**
 * Recursive helper method to insert a new key into the Binary Search Tree.
 *
 * @param root the root of the current subtree
 * @param key  the key to be inserted
 * @return the updated root of the subtree
 */
public Node insertRec(Node root, int key) {
    if (root == null) {
        root = new Node(key);
        return root;
    }

    if (key < root.key) {
        root.left = insertRec(root.left, key);
    } else if (key > root.key) {
        root.right = insertRec(root.right, key);
    }
    return root;
}

/**
 * Searches for a key in the Binary Search Tree.
 *
 * @param key the key to be searched
 * @return true if the key is found, false otherwise
 */
public boolean search(int key) {
    return searchRec(root, key);
}

/**
 * Recursive helper method to search for a key in the Binary Search Tree.
 *
 * @param root the root of the current subtree
 * @param key  the key to be searched
 * @return true if the key is found, false otherwise
 */
public boolean searchRec(Node root, int key) {
    if (root == null) {
        return false;
    }
    if (key == root.key) {
        return true;
    }
    return key < root.key ? searchRec(root.left, key) : searchRec(root.right, key);
}

/**
 * Deletes a key from the Binary Search Tree.
 *
 * @param key the key to be deleted
 */
public void deleteKey(int key) {
    root = deleteRec(root, key);
}

/**
 * Recursive helper method to delete a key from the Binary Search Tree.
 *
 * @param root the root of the current subtree
 * @param key  the key to be deleted
 * @return the updated root of the subtree
 */
```

```
public Node deleteRec(Node root, int key) {
    if (root == null) { // Base case: empty tree
        return root;
    }

    if (key < root.key) { // Recursively search for the key to be deleted
        root.left = deleteRec(root.left, key); // in the left subtree
    } else if (key > root.key) { // Recursively search for the key to be deleted
        root.right = deleteRec(root.right, key); // in the right subtree
    } else {
        if (root.left == null) // Node with only one child or no child
            return root.right; // Copy the contents of the non-empty child
        else if (root.right == null) // Node with only one child or no child
            return root.left; // Copy the contents of the non-empty child

        root.key = minValue(root.right); // Node with two children: Get the inorder
        root.right = deleteRec(root.right, root.key); // successor (smallest in the right su
    }

    return root;
}

/**
 * Finds the minimum value in the Binary Search Tree.
 *
 * @param root the root of the current subtree
 * @return the minimum value in the subtree
 */
public int minValue(Node root) {
    int minValue = root.key;
    while (root.left != null) {
        minValue = root.left.key;
        root = root.left;
    }
    return minValue;
}

/**
 * Performs an inorder traversal of the Binary Search Tree.
 */
public void inorder() {
    inorderRec(root);
}

/**
 * Recursive helper method to perform an inorder traversal of the Binary Search Tree.
 *
 * @param root the root of the current subtree
 */
public void inorderRec(Node root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}

/**
 * The main method to test the Binary Search Tree implementation.
 *
 * @param args the command-line arguments
 */
public static void main(String[] args) {
    BinarySearchTree bst = new BinarySearchTree();

    // Insert elements into the BST
    int[] elements = {45, 10, 7, 90, 12, 50, 13, 39, 57};
    for (int element : elements) {
        bst.insert(element);
    }

    // Display the tree elements in increasing order
    System.out.println("Inorder traversal of the Tree:");
    bst.inorder();
    System.out.println();

    // // Check whether a node with value 4 exists
    // boolean found = bst.search(4);
    // System.out.println("Search for value 4 in Tree? " + found);

    // // Delete Node (2) with no children
    // bst.deleteKey(2);
    // System.out.println("Inorder traversal after deleting node 2:");
    // bst.inorder();
    // System.out.println();

    // // Delete Node with one child (4)
    // bst.deleteKey(4);
    // System.out.println("Inorder traversal after deleting node 4:");
    // bst.inorder();
    // System.out.println();
    // // Delete Node with two children (10)
    // bst.deleteKey(10);
    // System.out.println("Inorder traversal after deleting node 10:");
    // bst.inorder();
}

}
```