BINARY SEARCH TREE

CODE:

```
#include <iostream>
using namespace std;

// Define the structure for a node in the BST
struct Node {
    int data;
    Node* left;
    Node* right;
};

class BST {
private:
    Node* root;

    Node* insertRec(Node* root, int data) {
        if (root == NULL) {
            Node* newNode = new Node();
            newNode->data = data;
            newNode->left = newNode->right = NULL;
            return newNode;
        }
        if (data < root->data) {
            root->left = insertRec(root->left, data);
        }
        else {
            root->right = insertRec(root->right, data);
        }
        return root;
    }

    Node* minValueNode(Node* node) {
        Node* current = node;
        while (current->left != NULL) {
            current = current->left;
```

```
    }
    return current;
}

Node* deleteRec(Node* root, int data) {
    if (root == NULL) {
        return root;
    }
    if (data < root->data) {
        root->left = deleteRec(root->left, data);
    }
    else if (data > root->data) {
        root->right = deleteRec(root->right, data);
    }
    else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteRec(root->right, temp->data);
    }
    return root;
}

bool searchRec(Node* root, int data) {
    if (root == NULL) {
        return false;
    }
    if (root->data == data) {
        return true;
    }
    if (data < root->data) {
```

```
            return searchRec(root->left, data);
        }
        return searchRec(root->right, data);
    }

    void preOrderRec(Node* root) {
        if (root != NULL) {
            cout << root->data << " ";
            preOrderRec(root->left);
            preOrderRec(root->right);
        }
    }

    void inOrderRec(Node* root) {
        if (root != NULL) {
            inOrderRec(root->left);
            cout << root->data << " ";
            inOrderRec(root->right);
        }
    }

    void postOrderRec(Node* root) {
        if (root != NULL) {
            postOrderRec(root->left);
            postOrderRec(root->right);
            cout << root->data << " ";
        }
    }

public:
    BST() {
        root = NULL;
    }

    void insert(int data) {
        root = insertRec(root, data);
    }

    void remove(int data) {
        root = deleteRec(root, data);
```

```
    }

    bool search(int data) {
        return searchRec(root, data);
    }

    void preOrder() {
        preOrderRec(root);
    }

    void inOrder() {
        inOrderRec(root);
    }

    void postOrder() {
        postOrderRec(root);
    }
};

int main() {
    BST bst;
    bst.insert(50);
    bst.insert(30);
    bst.insert(20);
    bst.insert(40);
    bst.insert(70);
    bst.insert(60);
    bst.insert(80);

    cout << "Inorder traversal: ";
    bst.inOrder();
    cout << endl;

    cout << "Preorder traversal: ";
    bst.preOrder();
    cout << endl;

    cout << "Postorder traversal: ";
    bst.postOrder();
    cout << endl;
```

```
    cout << "Delete 20" << endl;
    bst.remove(20);
    cout << "Inorder traversal: ";
    bst.inOrder();
    cout << endl;

    cout << "Search 40: " << (bst.search(40) ? "Found" : "Not Found") << endl;
    cout << "Search 100: " << (bst.search(100) ? "Found" : "Not Found") <<
endl;

    return 0;
}
```

OUTPUT: