```cpp
#include "tree.cpp"

bool BST::isEmpty()
{
    if (root == nullptr)
    {
        return true;
    }
    return false;
}

void BST::insert(int value)
{
    Node *newNode = new Node(value);
    if (isEmpty())
    {
        root = newNode;
    }
    else
    {
        Node *current = root;
        while (current)
        {
            if (value > current->data)
            {
                if (current->right == nullptr)
                {
                    current->right = newNode;
                    return;
                };
                current = current->right;
            }
            else if (value < current->data)
            {
                if (current->left == nullptr)
                {
                    current->left = newNode;
                    return;
                };
                current = current->left;
            }
```

```
        }
    }
}



bool BST::search(int value)
{

    Node *current = root;
    while (current)
    {
        if (value > current->data)
        {
            current = current->right;
        }
        else if (value < current->data)
        {
            current = current->left;
        }
        else if (value == current->data)
        {
            return true;
        }
    }
    return false;
};

Node *BST::findMin(Node *root)
{
    Node *current = root;
    while (current->left != nullptr)
    {
        current = current->left;
    }
    return current;
}
```

```cpp
int BST::findHight(Node *root)
{
    if (root == nullptr)
        return -1;

    return max(findHight(root->left), findHight(root->right)) + 1;
}

void BST::levelOrder(Node *root)
{
    queue<Node *> q;
    q.push(root);
    while (!q.empty())
    {
        Node *current = q.front();
        cout << current->data << " ";
        if (current->left)
            q.push(current->left);
        if (current->right)
            q.push(current->right);
        q.pop();
    }
}

void BST::preOrder(Node *root)
{
    if (root == nullptr)
        return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}
void BST::inOrder(Node *root)
{
    if (root == nullptr)
        return;
    inOrder(root->left);
    cout << root->data << ' ';
    inOrder(root->right);
}
```

```cpp
void BST::postOrder(Node *root)
{
    if (root == nullptr)
        return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << ' ';
}
```

```
Node *BST::deleteNode(Node *root, int data)
{
    if (root == nullptr)
        return root;
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else if (data < root->data)
        root->left = deleteNode(root->left, data);
    else
    {
        if (!root->left && !root->right)
        {

            delete root;
            root = nullptr;
        }
        else if (!root->left)
        {
            Node *temp = root;
            root = root->right;
            delete temp;
        }
        else if (!root->right)
        {
            Node *temp = root;
            root = root->left;
            delete temp;
        }
        else
        {
            Node *temp = findMin(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    };
    return root;
};

}
```