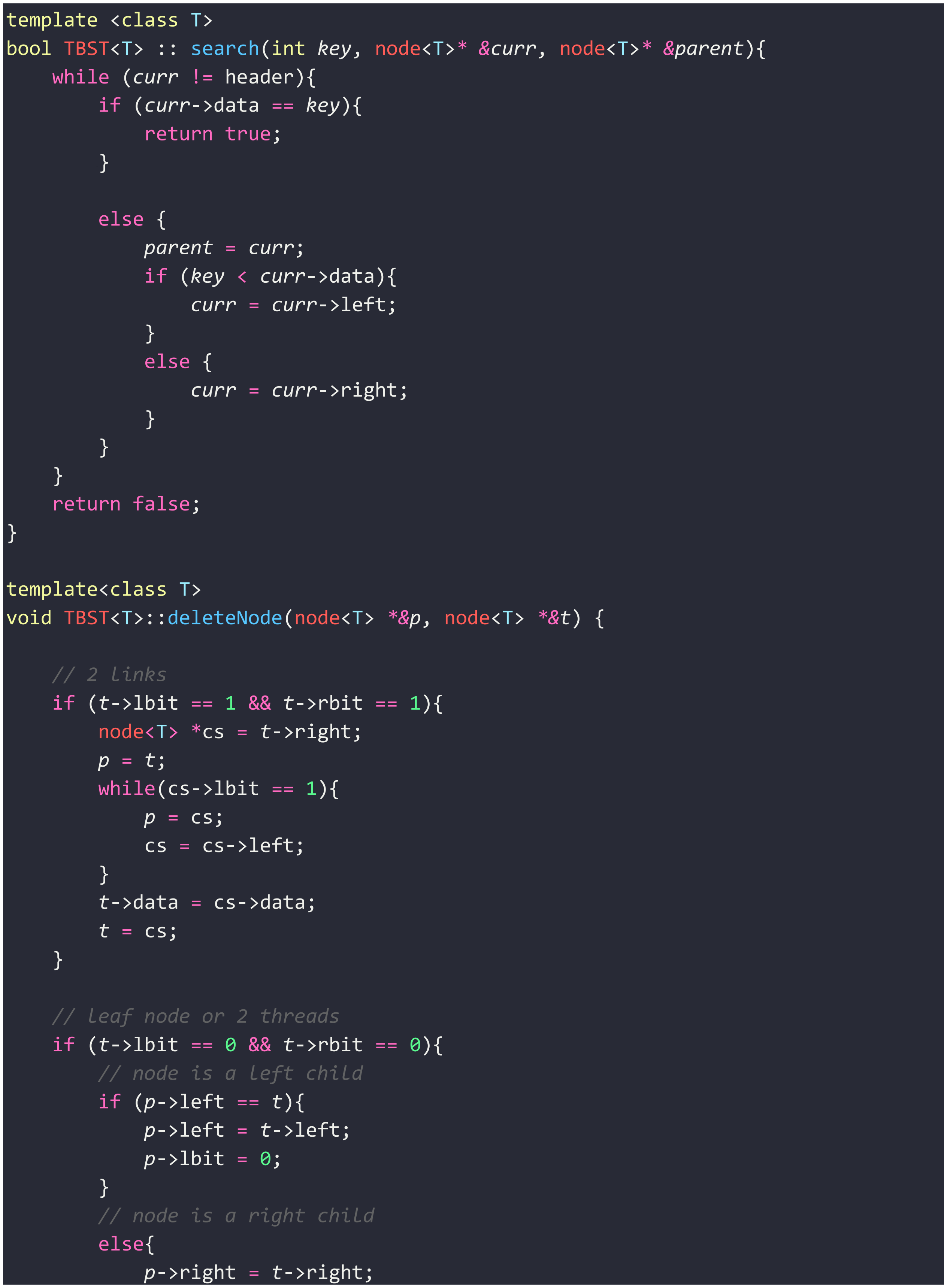
ASSIGNMENT NO 3

Create an inordered threaded binary search tree. Perform inorder, preorder traversals without recursion and deletion of a node. Analyze time and space complexity of the algorithm

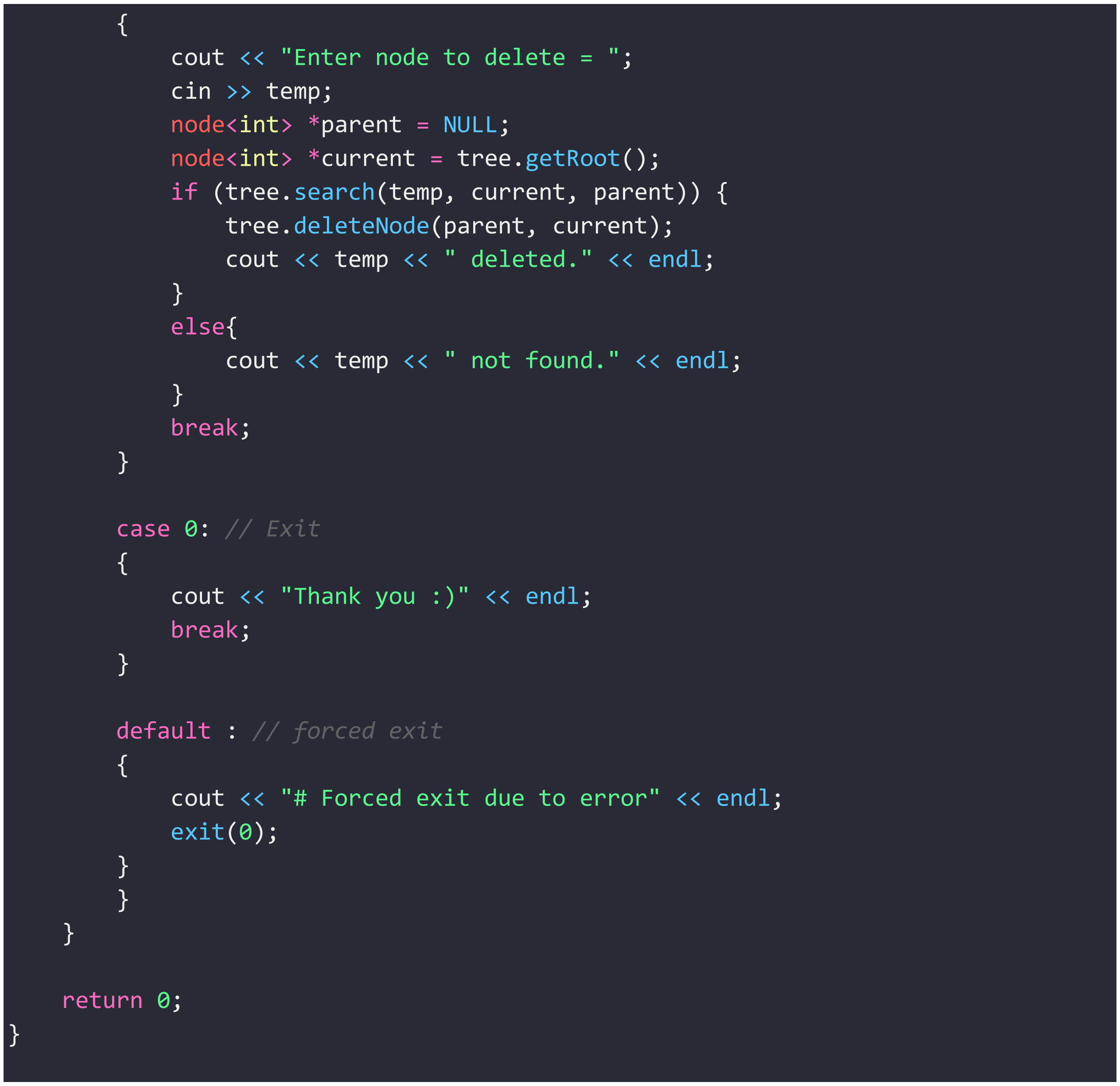
```
# include <iostream>
using namespace std;

// CLASS DEFINITON -----------------------------------------------------------------
-------

template <class T>
class node {
    T data;
    node<T> *left, *right;
    bool lbit, rbit;        // 0 - thread, 1 - link

public:
    // default constructor
    node(){
        data = 0;
        left = right = NULL;
        lbit = rbit = 0;        // thread
    }

    // parameterized constructor
    node(T val){
        data = val;
        left = right = NULL;
        lbit = rbit = 0;        // thread
    }

    void print(){
        cout << lbit << " |" << data << "| " << rbit << endl;
    }

    template <class F> friend class TBST;
};

template <class T>
class TBST {
    node<T> *header, *root;

public:
    // default constructor
    TBST(){
```

```cpp
        header = root = NULL;
    }

    // methods
    node<T>* getRoot(){
        return root;
    }

    void insert(T key);
    void inorder_traversal();
    void preorder_traversal();
    bool search(int key, node<T>* &curr, node<T>* &parent);
    void deleteNode(node<T> *&p, node<T> *&t);
};

// METHOD DEFINITON -----------------------------------------------------------------
-------

template<class T>
void TBST<T>::insert(T key) {
    // check if TBST is empty
    if (root == NULL){
        // create header
        header = new node<T>(-99);
        header->right = header;

        // create root
        root = new node<T>(key);

        header->left = root;
        header->lbit = 1;

        root->left = root->right = header;

        return;
    }

    // non-empty TBST
    else{
        node<T> *parent = root;
        node<T> *temp = new node<T>(key);

        while(true){
            if (key == parent->data){
                cout << "Node " << key << " already exists" << endl;
                delete temp;
                temp = NULL;
                return;
            }
            else if (key < parent->data){
                // explore left subtree
                if (parent->lbit == 1){
                    // left link
                    parent = parent->left;
                }
                else{
                    // left thread - insert node as left child
                    temp->left = parent->left;
                    temp->right = parent;
                    parent->left = temp;
                    parent->lbit = 1;
                    return;
                }
            }
            else {  // key > parent->data
                // explore right subtree
                if (parent->rbit == 1){
                    // right link
                    parent = parent->right;
                }
                else{
                    // right thread - insert node as right child
                    temp->right = parent->right;
                    temp->left = parent;
                    parent->right = temp;
                    parent->rbit = 1;
                    return;
                }

            }
            // end-comparison
        }
        // end-while-loop
    }
    //end-else
}

template<class T>
void TBST<T>::inorder_traversal() {
    node<T> *temp = root;
    // find extreme left node from root
    while(temp->lbit == 1){
        temp = temp->left;
    }

    while(temp != header){
        // print node
        temp->print();

        // Right is a link
        if (temp->rbit == 1){
            temp = temp->right;
            // find leftmost node
            while(temp->lbit == 1){
                temp = temp->left;
            }
        }
        // Right is a thread
        else{
            temp = temp->right;
        }
    }
}

template<class T>
void TBST<T>::preorder_traversal() {
    bool flag = 0;
    node<T> *temp = root;

    while(temp != header){
        if (flag == 0){
            // print node
            temp->print();
        }
        if (flag == 0 && temp->lbit == 1){
            temp = temp->left;
        }
        else{
            flag = (temp->rbit == 1) ? (0) : (1);
            temp = temp->right;
        }
    }
}
```

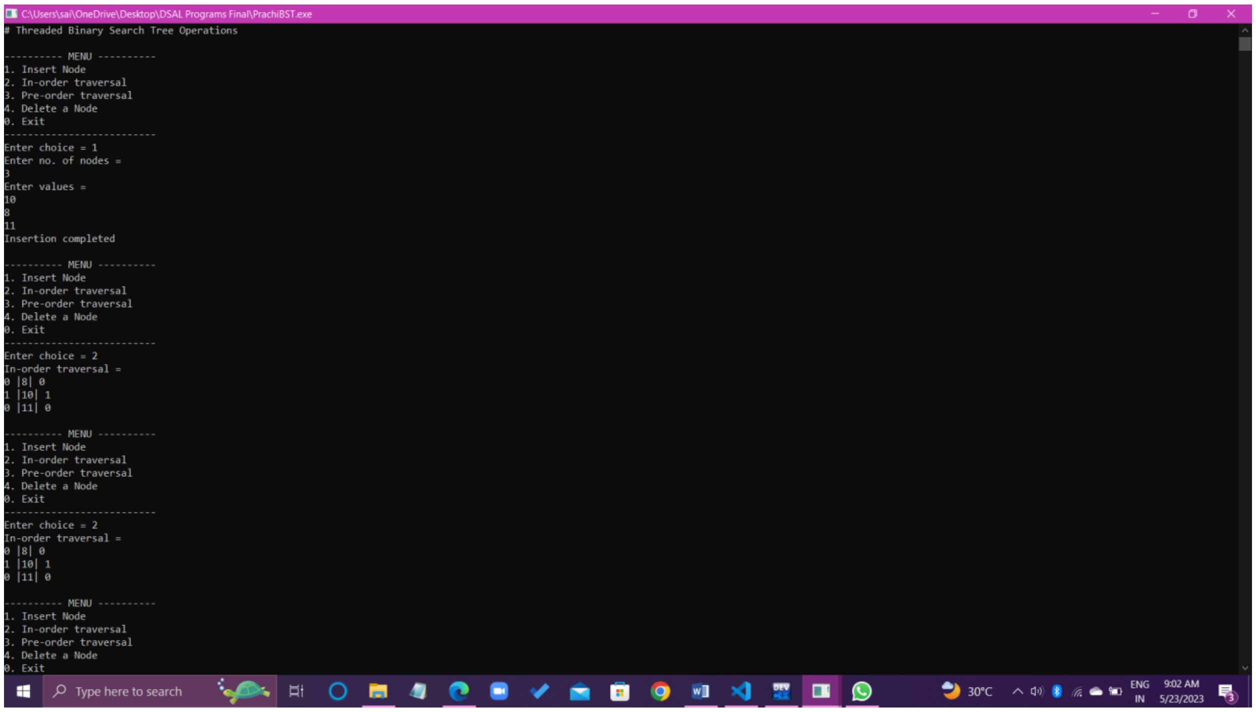


```cpp
template <class T>
bool TBST<T> :: search(int key, node<T>* &curr, node<T>* &parent){
    while (curr != header){
        if (curr->data == key){
            return true;
        }

        else {
            parent = curr;
            if (key < curr->data){
                curr = curr->left;
            }
            else {
                curr = curr->right;
            }
        }
    }
    return false;
}

template<class T>
void TBST<T>::deleteNode(node<T> *&p, node<T> *&t) {

    // 2 links
    if (t->lbit == 1 && t->rbit == 1){
        node<T> *cs = t->right;
        p = t;
        while(cs->lbit == 1){
            p = cs;
            cs = cs->left;
        }
        t->data = cs->data;
        t = cs;
    }

    // leaf node or 2 threads
    if (t->lbit == 0 && t->rbit == 0){
        // node is a left child
        if (p->left == t){
            p->left = t->left;
            p->lbit = 0;
        }
        // node is a right child
        else{
            p->right = t->right;
            p->rbit = 0;
        }
        delete t;
    }

    // left link, right thread
    if (t->lbit == 1 && t->rbit == 0){
        node<T> *temp = t->left;
        // node is a left child
        if (p->left == t){
            p->left = temp;
        }
        // node is a right child
        else{
            p->right = temp;
        }
        while (temp->rbit == 1){
            temp = temp->right;
        }
        temp->right = t->right;
        delete t;
    }

    // right link, left thread
    if (t->lbit == 0 && t->rbit == 1){
        node<T> *temp = t->right;
        if (p->left == t){
            p->left = temp;
        }
        else{
            p->right = temp;
        }
        while (temp->lbit == 1){
            temp = temp->left;
        }
        temp->left = t->left;
        delete t;
    }
}

// DRIVER CODE -----------------------------------------------------------------
-------

int main(){
    TBST<int> tree;
```

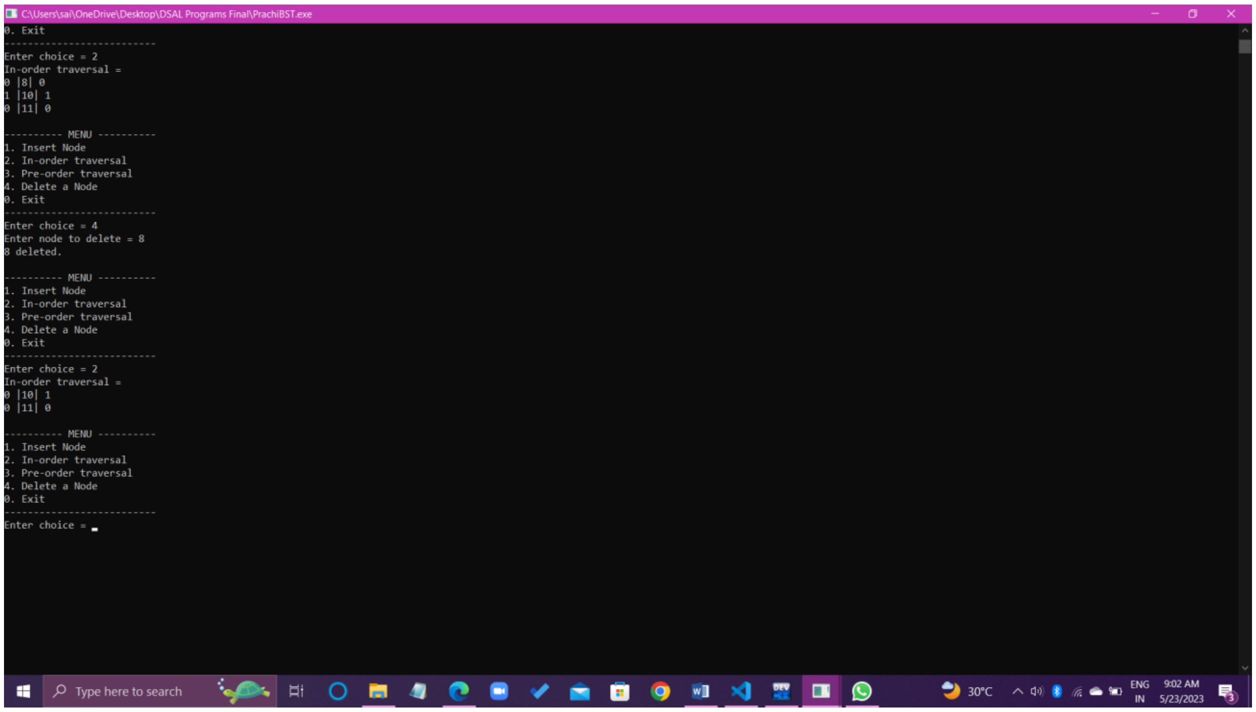
```cpp
    int choice = -1;
    int temp = 0;
    int value = 0;

    cout << "# Threaded Binary Search Tree Operations\n";
    while(choice){
        cout << "\n---------- MENU ----------\n"
             << "1. Insert Node\n"
             << "2. In-order traversal\n"
             << "3. Pre-order traversal\n"
             << "4. Delete a Node\n"
             << "0. Exit"
             << "\n-----------------------------" << endl;
        cout << "Enter choice = ";
        cin >> choice;

        switch(choice){
        case 1: // Insert Node
        {
            cout << "Enter no. of nodes = " << endl;
            cin >> temp;
            cout << "Enter values = " << endl;
            while(temp--){
                cin >> value;
                tree.insert(value);
            }
            cout << "Insertion completed" << endl;
            break;
        }

        case 2: // In-order traversal
        {
            cout << "In-order traversal =\n";
            tree.inorder_traversal();
            break;
        }

        case 3: // Pre-order traversal
        {
            cout << "Pre-order traversal =\n";
            tree.preorder_traversal();
            break;
        }

        case 4: // Delete node
```



```cpp
        {
            cout << "Enter node to delete = ";
            cin >> temp;
            node<int> *parent = NULL;
            node<int> *current = tree.getRoot();
            if (tree.search(temp, current, parent)) {
                tree.deleteNode(parent, current);
                cout << temp << " deleted." << endl;
            }
            else{
                cout << temp << " not found." << endl;
            }
            break;
        }

        case 0: // Exit
        {
            cout << "Thank you :)" << endl;
            break;
        }

        default : // forced exit
        {
            cout << "# Forced exit due to error" << endl;
            exit(0);
        }
        }
    }

    return 0;
}
```


C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\PrachiBST.exe
Type here to search
30°C
ENG IN
9:02 AM
5/23/2023

```
# Threaded Binary Search Tree Operations

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = 1
Enter no. of nodes =
3
Enter values =
10
8
11
Insertion completed

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = 2
In-order traversal =
0 |8| 0
1 |10| 1
0 |11| 0

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = 2
In-order traversal =
0 |8| 0
1 |10| 1
0 |11| 0

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
```


C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\PrachiBST.exe
Type here to search
30°C
ENG IN
9:02 AM
5/23/2023

```
0. Exit
--------------------------
Enter choice = 2
In-order traversal =
0 |8| 0
1 |10| 1
0 |11| 0

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = 4
Enter node to delete = 8
8 deleted.

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = 2
In-order traversal =
0 |10| 1
0 |11| 0

---------- MENU ----------
1. Insert Node
2. In-order traversal
3. Pre-order traversal
4. Delete a Node
0. Exit
--------------------------
Enter choice = _
```