DSA Assignment 8

Name: Jashnoor Singh
Roll: 1024030106

Class BST and Class Node (making a separate file to include in later codes)

```
#include <iostream>
using namespace std;
class node{
    public:
    int data;
    node* left;
    node* right;

    node(int d){
        this->data = d;
        this->left = NULL;
        this->right = NULL;
    }
};
node* buildTree(node* root, int data){
    if(root == NULL){
        return new node(data);
    }
    if(data>root->data){
        root->right = buildTree(root->right, data);
    }
    else{
        root->left = buildTree(root->left, data);
    }
    return root;
}
```

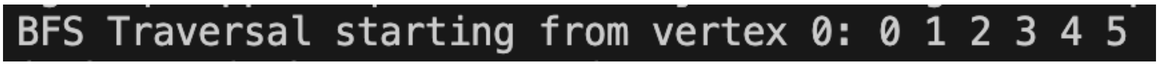
```
// making here tree class to include in other programs
```

```
#include <iostream>
using namespace std;
class node{
    public:
    int data;
    node* left;
    node* right;

    node(int d){
        this->data = d;
        this->left = NULL;
        this->right = NULL;
    }
};

node* buildTree(node* root){
    int data;
    cout<<"Enter data\n";
    cin>>data;
    root = new node(data);
    if(data == -1){
        return NULL;
    }
    cout<<"Enter to the left of "<<data<<endl;
    root->left = buildTree(root->left);
    cout<<"Enter to the right of "<<data<<endl;
    root->right = buildTree(root->right);
    return root;
}
```

Q1 a) Input is 1 2 3 4 5 6 7 in all the cases

```
// preorder traversal

#include "class_node.cpp" // i can use node class directly now

void preorder(node* root){
    if(root == NULL){
        return;
    }
    cout<<root->data<<" ";
    preorder(root->left);
    preorder(root->right);
}
int main(){
    node* root;
    root = buildTree(root);
    preorder(root);
}
```

Output:



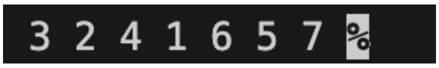
```
BFS Traversal starting from vertex 0: 0 1 2 3 4 5
```

Q1 b)

```
// in order traversal

#include "class_node.cpp"
```

```
void inorder(node* root){
    if(root == NULL){
        return;
    }
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);

}

int main(){
    node* root;
    root = buildTree(root);
    inorder(root);
}
```

Output:

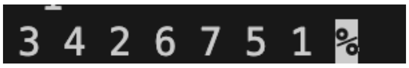


```
 3 2 4 1 6 5 7 %
```

Q1 c)

```
// post order traversal

#include "class_node.cpp"

void postorder(node* root){
    if(root == NULL){
        return;
    }
    postorder(root->left);
    postorder(root->right);
    cout<<root->data<<" ";
}

int main(){
    node* root;
    root = buildTree(root);
    postorder(root);
}
```

Output:



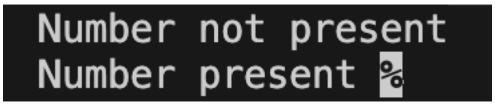
```
3 4 2 6 7 5 1 %
```

Q2 a)

```cpp
// search a given element in bst

#include "class_bst.cpp" // i can use binary tree directly now

bool searching(node* root, int key){ // recursive method
    if(root == NULL) return false;
    if(key == root->data) return true;
    else if(key < root->data)return searching(root->left,key);
    else return searching(root->right,key);
}
bool search_iterative(node* root, int key){
    while(root!=NULL){
        if(key == root->data){
            return true;
        }
        if(key > root->data){
            root = root->right;
        }
        else{
            root = root->left;
        }
    }
    return false;
}
int main(){
    node* root = NULL;
    root = buildTree(root,8);
    root = buildTree(root,6);
    root = buildTree(root,5);
    root = buildTree(root,7);
    root = buildTree(root,10);
    root = buildTree(root,9);
    root = buildTree(root,12);
    if(searching(root,100)){
        cout<<"Number present \n";
    }
    else cout<<"Number not present \n";
    if(search_iterative(root,10)){
        cout<<"Number present ";
    }
    else cout<<"Number not present ";
}
```

Output:

Searching 100 and 10

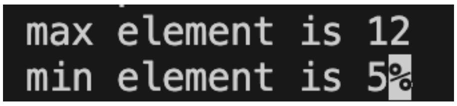


```
Number not present
Number present %
```

Q2 b)

```
#include "class_bst.cpp" // I can use binary tree class directly 
now

int max_element(node* root){
    if(root->left == NULL && root->right == NULL){
        return root->data;
    }
    return max_element(root->right);
}
int min_element(node* root){
    if(root->left == NULL && root->right == NULL){
        return root->data;
    }
    return min_element(root->left);
}
int main(){
    node* root = NULL;
    root = buildTree(root,8);
    root = buildTree(root,6);
    root = buildTree(root,5);
    root = buildTree(root,7);
    root = buildTree(root,10);
    root = buildTree(root,9);
    root = buildTree(root,12);
    cout<<"max element is "<<max_element(root)<<endl;
    cout<<"min element is "<<min_element(root);
}
```

Output:

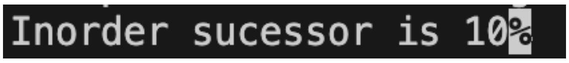


```
max element is 12
min element is 5%
```

Q2 c)

```
// finding inorder sucessor

#include "class_bst.cpp"

node* suc_rec(node* root, int value){
    if(root == NULL){
        return root;
    }
    if(value>=root->data){
        return suc_rec(root->right,value);
    }
    else{
        node* successor = suc_rec(root->left,value);
        if(successor!=NULL){
            return successor;
        }
        else return root;
    }
}
int main(){
    node* root = NULL;
    root = buildTree(root,8);
    root = buildTree(root,6);
    root = buildTree(root,5);
    root = buildTree(root,7);
    root = buildTree(root,10);
    root = buildTree(root,9);
    root = buildTree(root,12);
    node* succ = suc_rec(root,9);
    cout<<"Inorder sucessor is "<<succ->data;
}
```

Output:

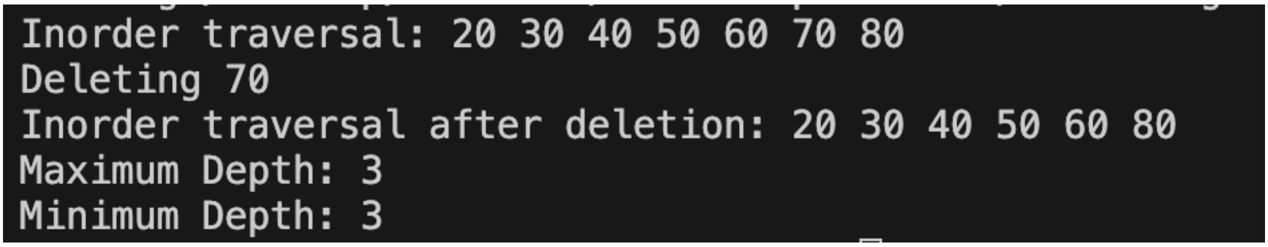


```
Inorder sucessor is 10%
```

Q3:

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

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = right = NULL;
    }
};

class BST {
public:
    Node* root;

    BST() {
        root = NULL;
    }

    // Insert function (no duplicates)
    Node* insert(Node* node, int key) {
        if (node == NULL)
            return new Node(key);

        if (key < node->data)
            node->left = insert(node->left, key);
        else if (key > node->data)
            node->right = insert(node->right, key);
        else
            cout << "Duplicate value not allowed!\n";

        return node;
    }

    void insert(int key) {
        root = insert(root, key);
    }

    // Find minimum node (for deletion)
    Node* findMin(Node* node) {
        while (node && node->left != NULL)
            node = node->left;
        return node;
    }

    // Delete a node
    Node* deleteNode(Node* node, int key) {
        if (node == NULL)
            return node;

        if (key < node->data)
            node->left = deleteNode(node->left, key);

        else if (key > node->data)
            node->right = deleteNode(node->right, key);

        else {
            // Case 1: leaf or single child
            if (node->left == NULL) {
                Node* temp = node->right;
                delete node;
                return temp;
            }
            else if (node->right == NULL) {
                Node* temp = node->left;
                delete node;
                return temp;
            }
            // Case 2: node with two children
            Node* temp = findMin(node->right);
            node->data = temp->data;
            node->right = deleteNode(node->right, temp->data);
        }
        return node;
    }

    void deleteValue(int key) {
        root = deleteNode(root, key);
    }

    // Maximum depth
    int maxDepth(Node* node) {
        if (node == NULL) return 0;
        return 1 + max(maxDepth(node->left), maxDepth(node->right));
    }

    // Minimum depth
    int minDepth(Node* node) {
        if (node == NULL) return 0;
        if (node->left == NULL && node->right == NULL) return 1;

        int leftDepth = (node->left) ? minDepth(node->left) : INT_MAX;
        int rightDepth = (node->right) ? minDepth(node->right) : INT_MAX;

        return 1 + min(leftDepth, rightDepth);
    }

    // Inorder traversal
    void inorder(Node* node) {
        if (node == NULL) return;
        inorder(node->left);
        cout << node->data << " ";
        inorder(node->right);
    }

    void printInorder() {
        inorder(root);
        cout << endl;
    }
};

int main() {
    BST tree;

    // Insert values
    tree.insert(50);
    tree.insert(30);
    tree.insert(70);
    tree.insert(20);
    tree.insert(40);
    tree.insert(60);
    tree.insert(80);

    cout << "Inorder traversal: ";
    tree.printInorder();

    // Delete a value
    cout << "Deleting 70\n";
    tree.deleteValue(70);
    cout << "Inorder traversal after deletion: ";
    tree.printInorder();

    // Maximum depth
    cout << "Maximum Depth: " << tree.maxDepth(tree.root) << endl;

    // Minimum depth
    cout << "Minimum Depth: " << tree.minDepth(tree.root) << endl;

    return 0;
}
```

Output:



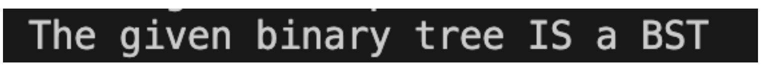
```
Inorder traversal: 20 30 40 50 60 70 80
Deleting 70
Inorder traversal after deletion: 20 30 40 50 60 80
Maximum Depth: 3
Minimum Depth: 3
```

Q4:

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

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int val){
        data = val;
        left = right = nullptr;
    }
```

```
};

class BinaryTree {
    Node* root;
    int prev;       // used for BST check
    bool firstNode; // track first visited inorder node
    bool isBSTUtil(Node* node) {
        if (node == nullptr) return true;
        // Left subtree
        if (!isBSTUtil(node->left)) return false;
        // Check current node with previous inorder node
        if (!firstNode && node->data <= prev)
            return false;
        prev = node->data;
        firstNode = false;
        // Right subtree
        return isBSTUtil(node->right);
    }
    public:
    BinaryTree() {
        root = nullptr;
        prev = -999999;
        firstNode = true;
    }
    Node* insert(Node* node, int val) {
        if (node == nullptr) {
            return new Node(val);
        }
        if (val < node->data)
            node->left = insert(node->left, val);
        else
            node->right = insert(node->right, val);

        return node;
    }

    void insertValue(int val) {
        root = insert(root, val);
    }

    bool isBST() {
        prev = -999999;
        firstNode = true;
        return isBSTUtil(root);
    }
};

int main() {
    BinaryTree bt;

    // Build a BST
    bt.insertValue(50);
    bt.insertValue(30);
    bt.insertValue(70);
    bt.insertValue(20);
    bt.insertValue(40);
    bt.insertValue(60);
    bt.insertValue(80);

    if (bt.isBST())
        cout << "The given binary tree IS a BST\n";
    else
        cout << "The given binary tree is NOT a BST\n";

    return 0;
}
```

Output:



```
 The given binary tree IS a BST
```