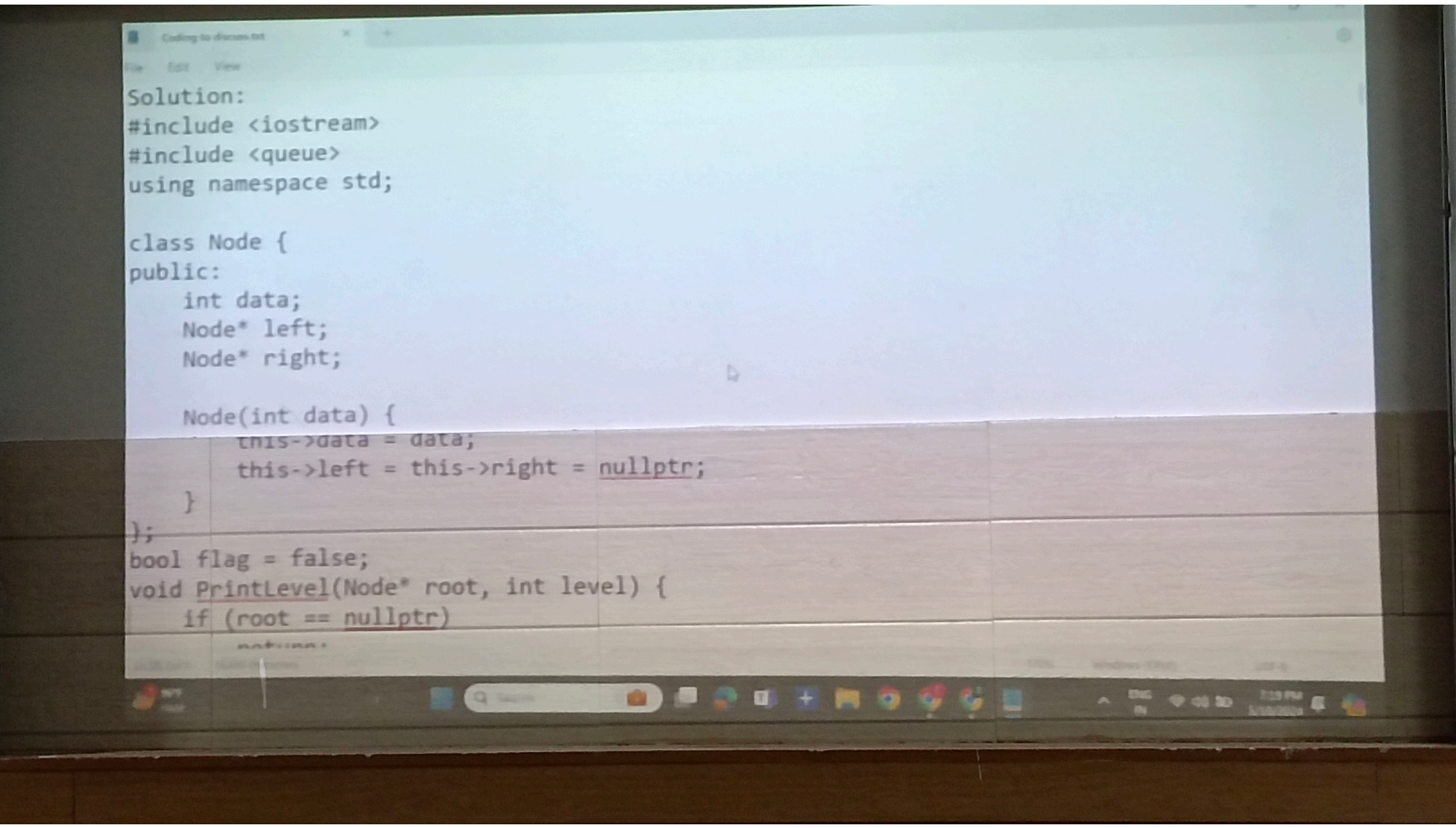
Coding to discuss.txt

File Edit View

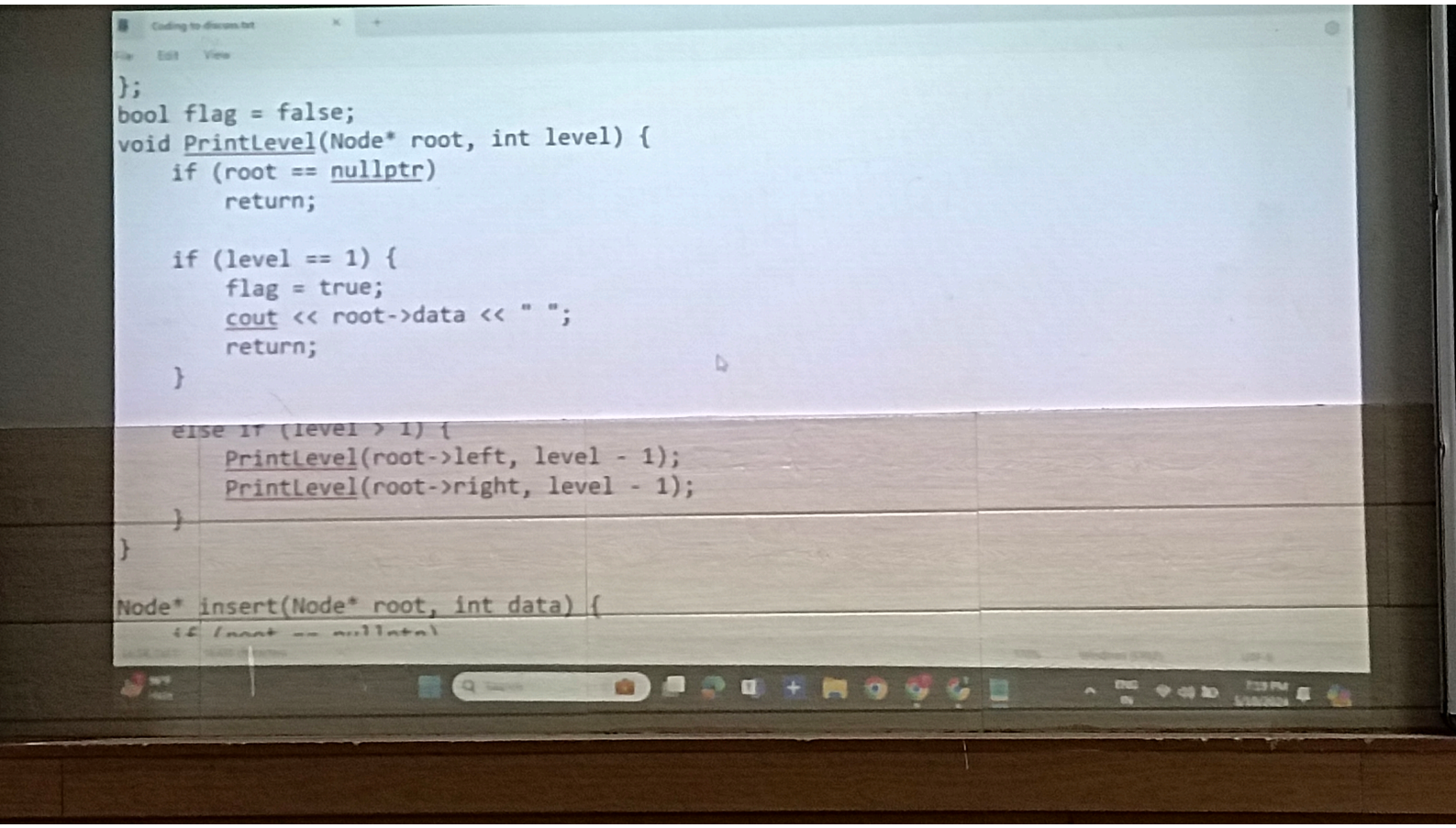
Q.1. Print Nodes of Specific Level
You are tasked with implementing a function to print the nodes of a specific level in a binary search tree. The tree nodes contain integer values, and the levels are 0-indexed.

Sample Testcase 1
Input:
5
4 3 5 1 6
2
Output:
1 6

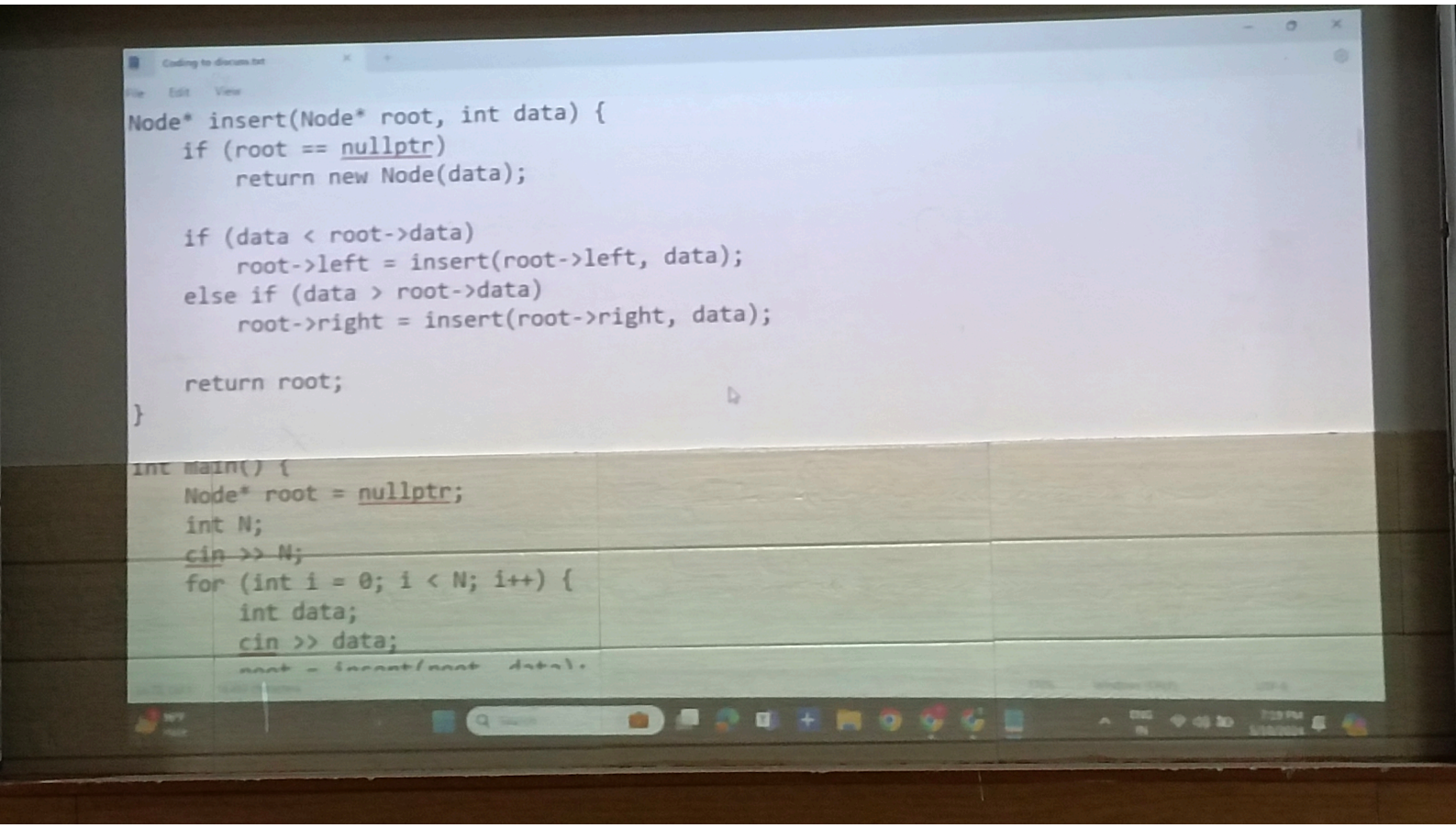
Sample Testcase 2:
Input:
7
10 5 15 3 7 12 18
2
Output:
3 7 12 18



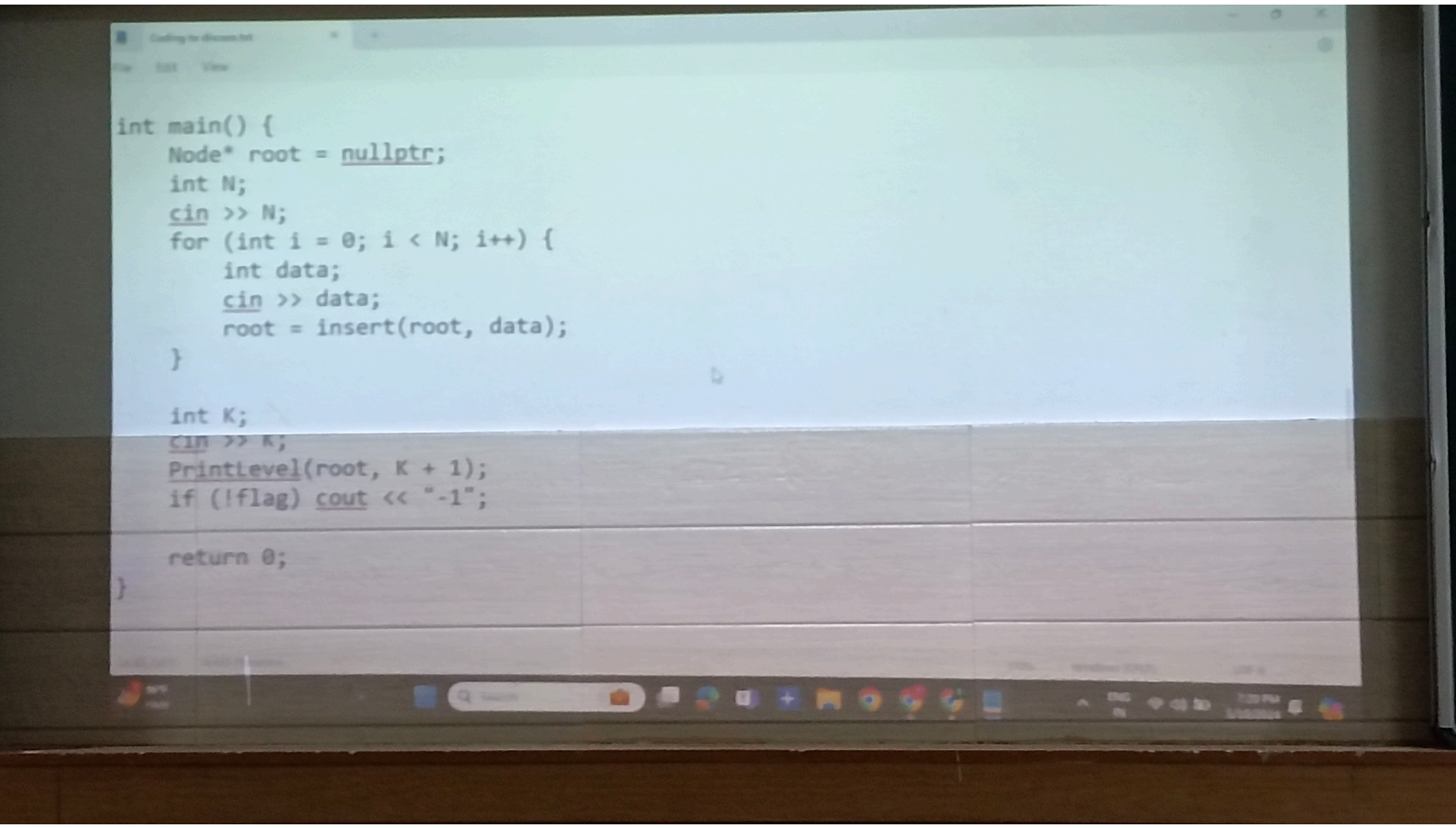
```
Solution:
#include <iostream>
#include <queue>
using namespace std;

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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};
bool flag = false;
void PrintLevel(Node* root, int level) {
    if (root == nullptr)
        return;
```



```
};
bool flag = false;
void PrintLevel(Node* root, int level) {
    if (root == nullptr)
        return;

    if (level == 1) {
        flag = true;
        cout << root->data << " ";
        return;
    }

    else if (level > 1) {
        PrintLevel(root->left, level - 1);
        PrintLevel(root->right, level - 1);
    }
}

Node* insert(Node* root, int data) {
    if (root == nullptr)
```



```
Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
```

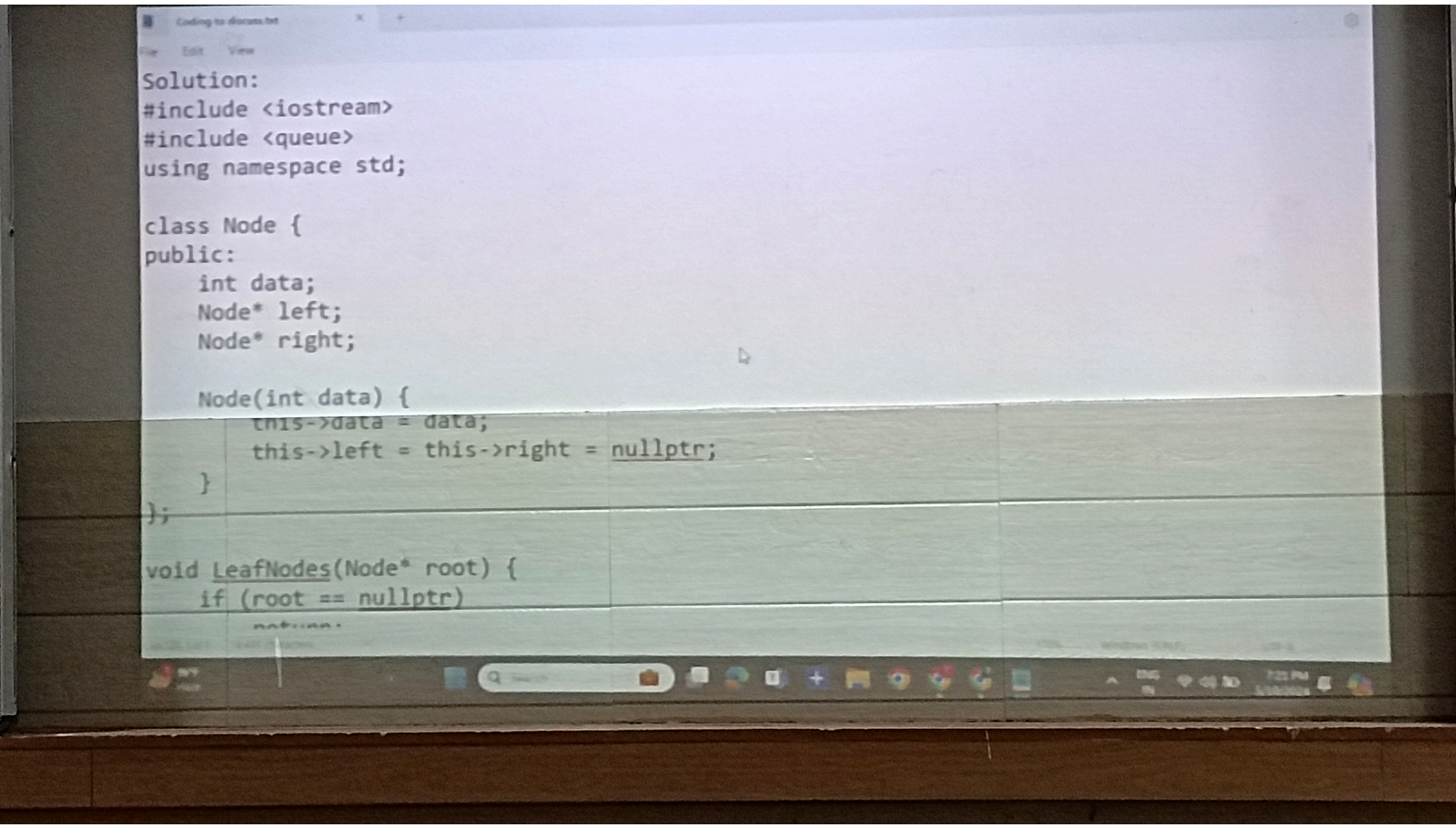


```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
    }

    int K;
    cin >> K;
    PrintLevel(root, K + 1);
    if (!flag) cout << "-1";

    return 0;
}
```

Q.2. Identifying Leaf Nodes in a Binary Search Tree
You are given a C++ program that constructs a binary search tree based on user input and identifies and prints the leaf nodes of the tree. The program defines a Node class representing individual nodes in the tree and includes a LeafNodes method that prints the values of leaf nodes.

Sample Testcase 1
Input:
9
8 3 10 1 6 4 7 13 14
Output:
1 4 7 14

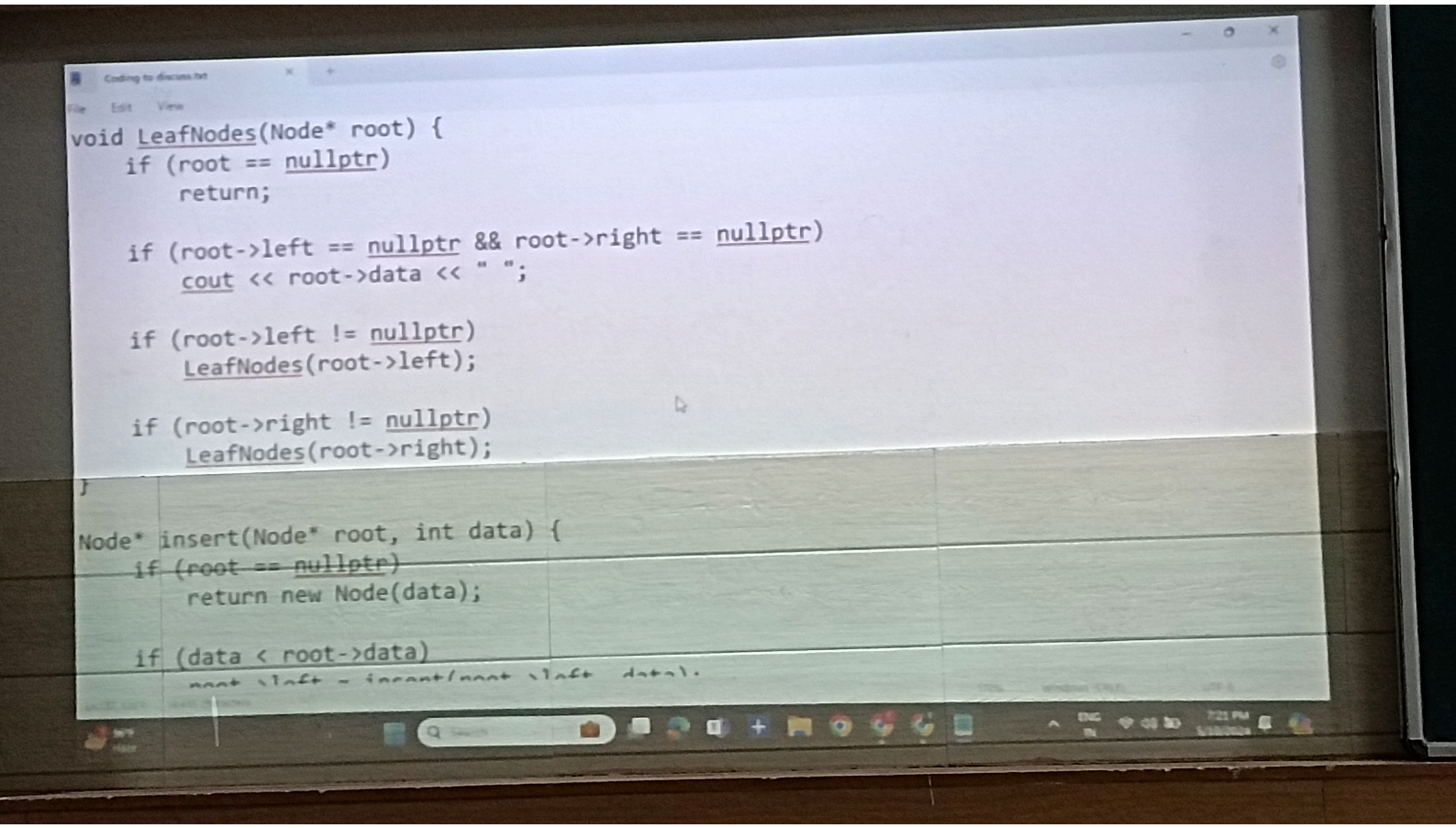
Sample Testcase 2:
Input:
7
10 5 15 3 7 12 18
Output:
3 7 12 18



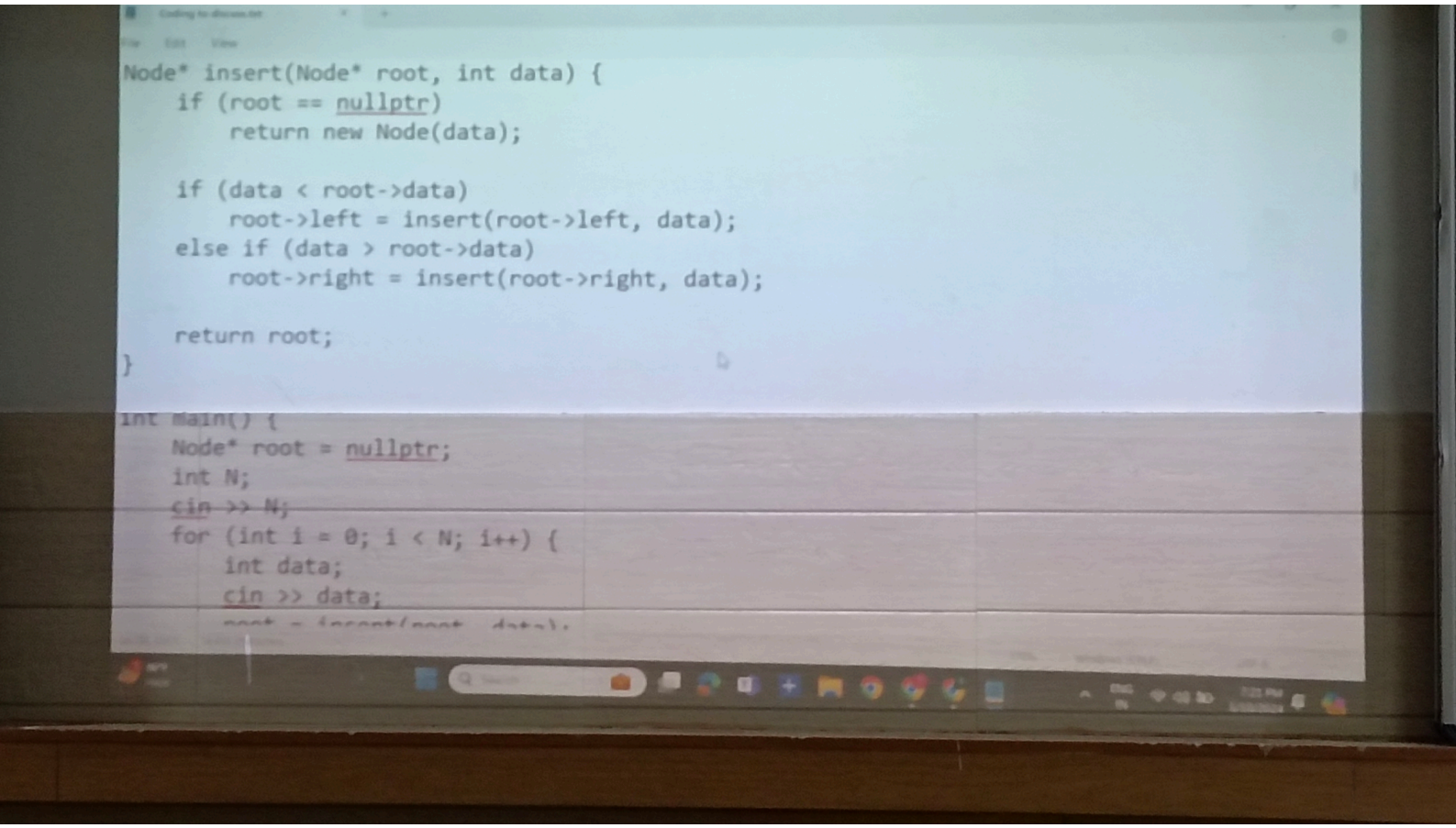
Coding to discuss.txt

File Edit View

```
Solution:
#include <iostream>
#include <queue>
using namespace std;

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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

void LeafNodes(Node* root) {
    if (root == nullptr)
        return;
```



```
void LeafNodes(Node* root) {
    if (root == nullptr)
        return;

    if (root->left == nullptr && root->right == nullptr)
        cout << root->data << " ";

    if (root->left != nullptr)
        LeafNodes(root->left);

    if (root->right != nullptr)
        LeafNodes(root->right);
}

Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);

    if (data < root->data)
        root->left = insert(root->left, data);
```



```
Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
```

```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
    }

    LeafNodes(root);

    return 0;
}
```

Q.3. Right View of a Binary Search Tree

You are provided with a C++ program that constructs a binary search tree based on user input and identifies and prints the right view of the tree. The program defines a Node class representing individual nodes in the tree and includes a RightView method that

Coding to discuss.txt

File Edit View

Q.3. Right View of a Binary Search Tree

You are provided with a C++ program that constructs a binary search tree based on user input and identifies and prints the right view of the tree. The program defines a Node class representing individual nodes in the tree and includes a RightView method that prints the values of nodes visible from the right side of the tree.

Sample Testcase 1

Input:

9

8 3 10 1 6 4 7 13 14

Output:

8 10 13 14

Sample Testcase 2:

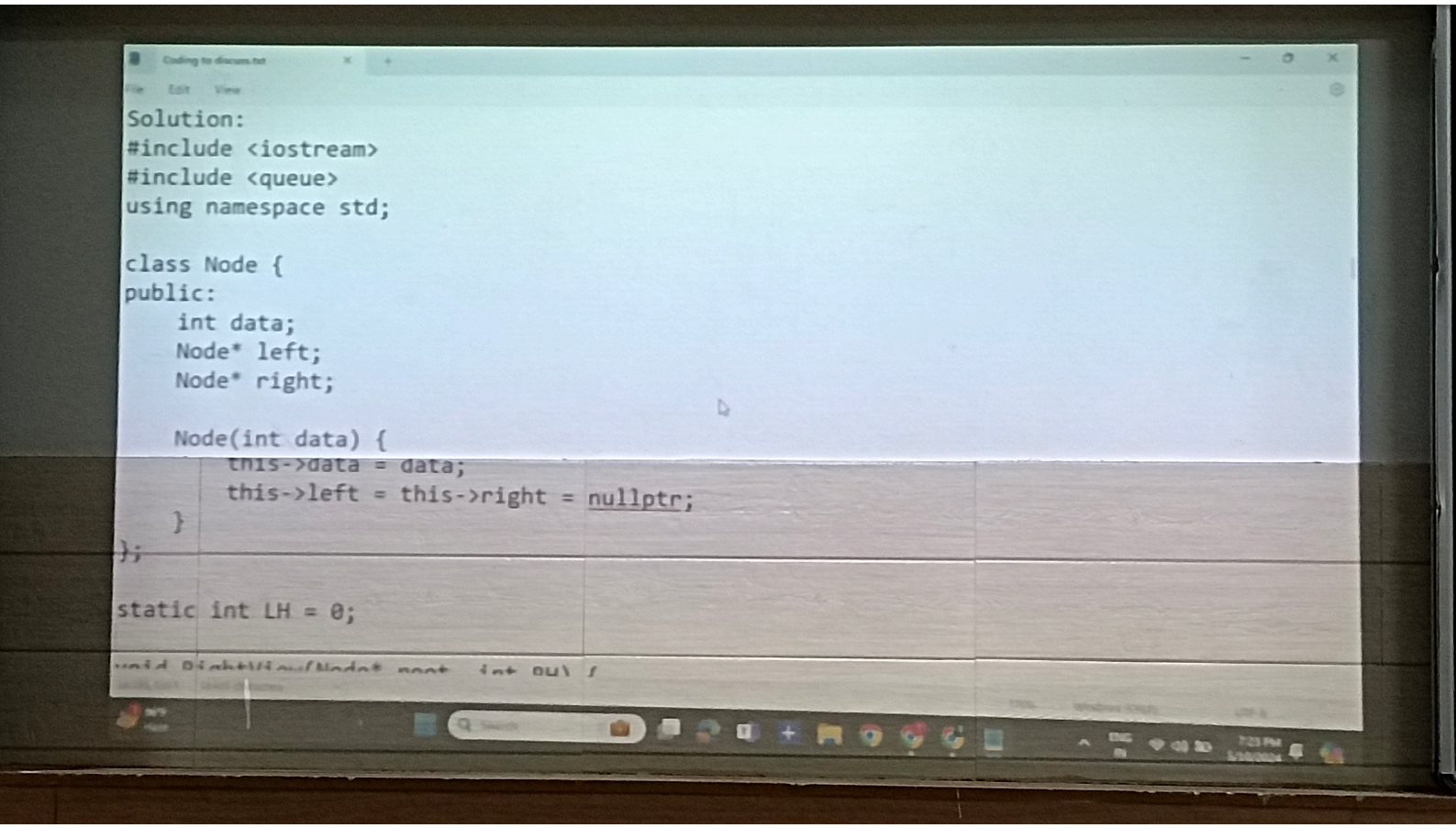
Input:

7

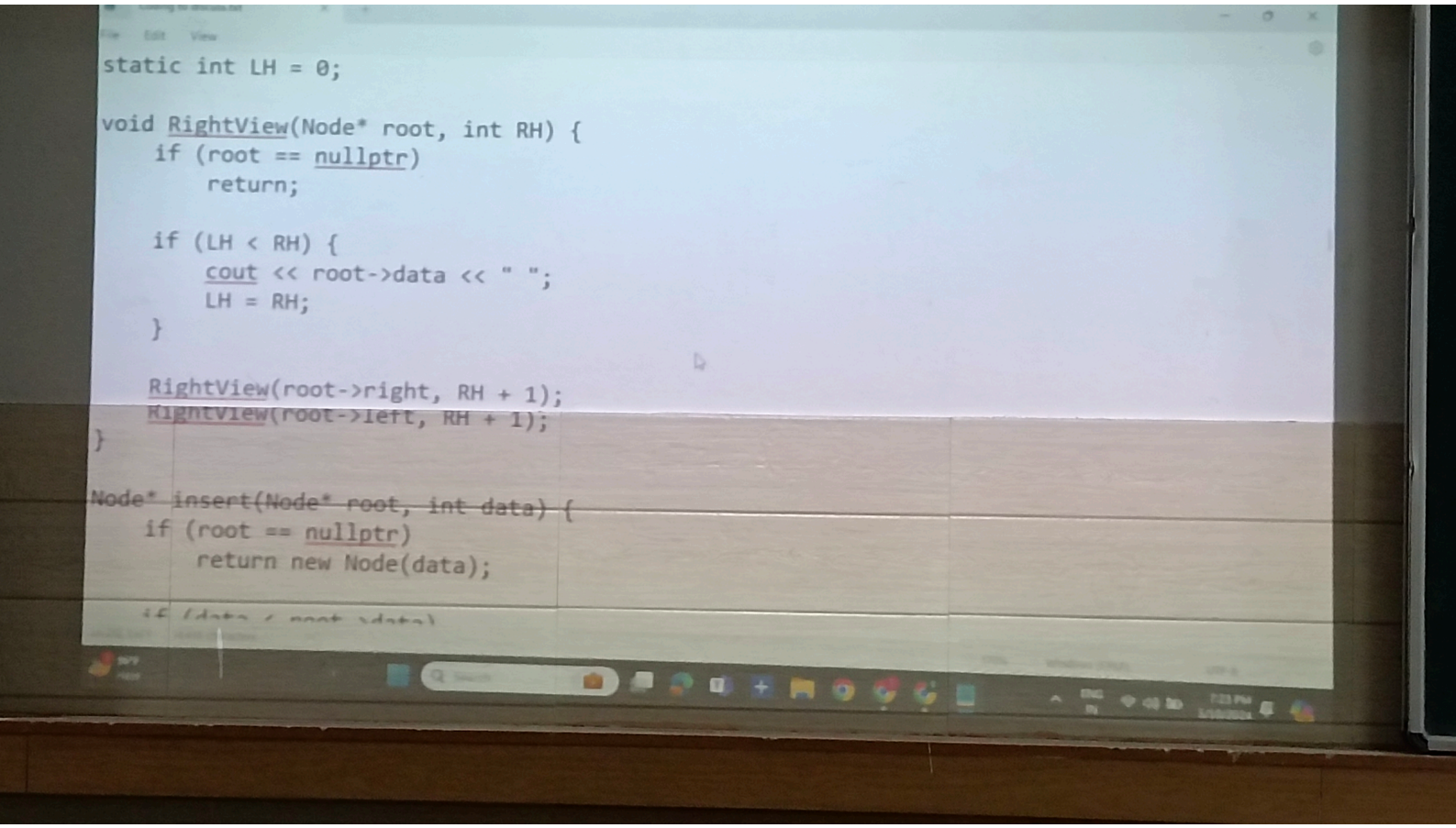
10 5 15 3 7 12 18

Output:

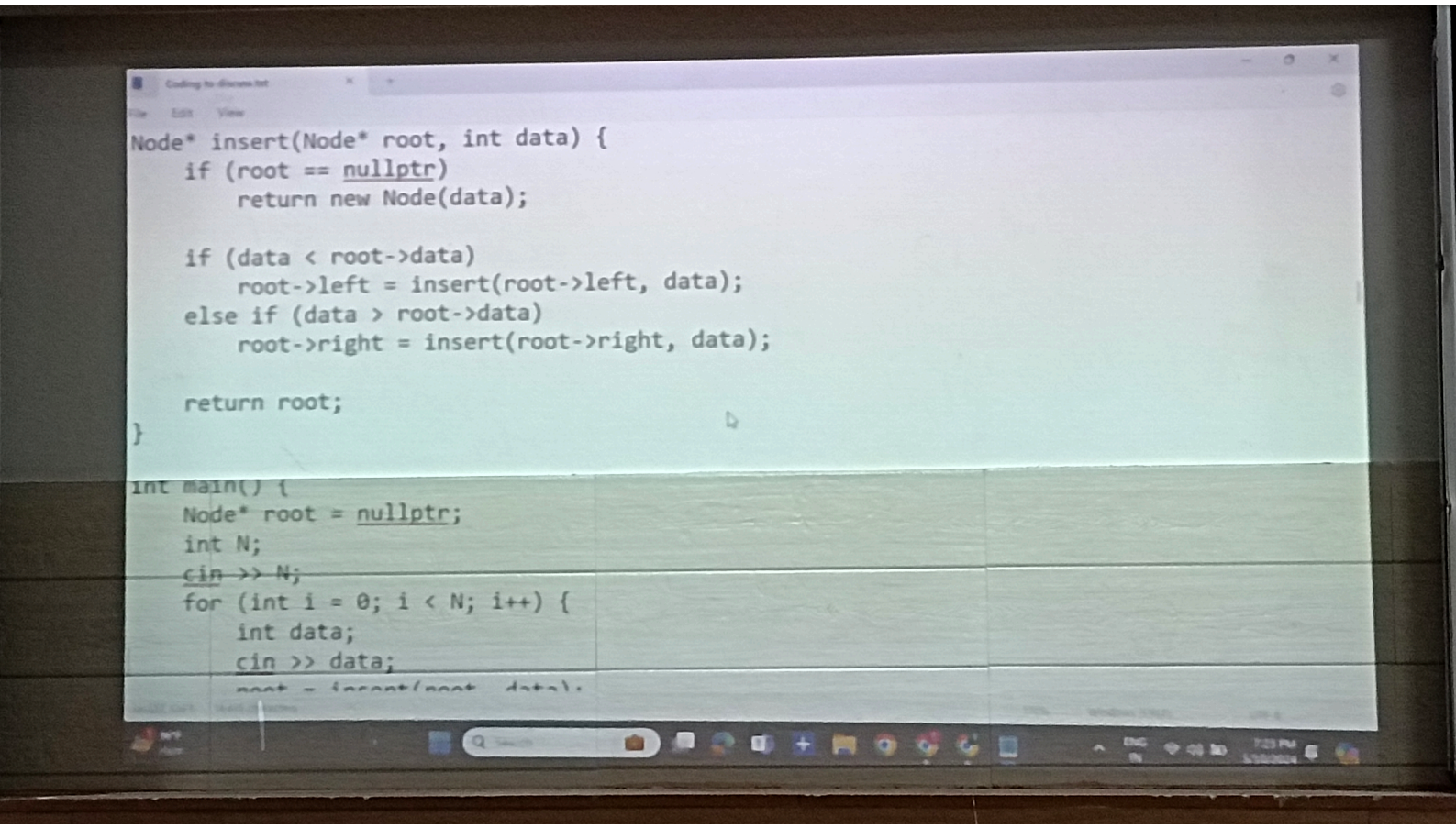
10 15 18



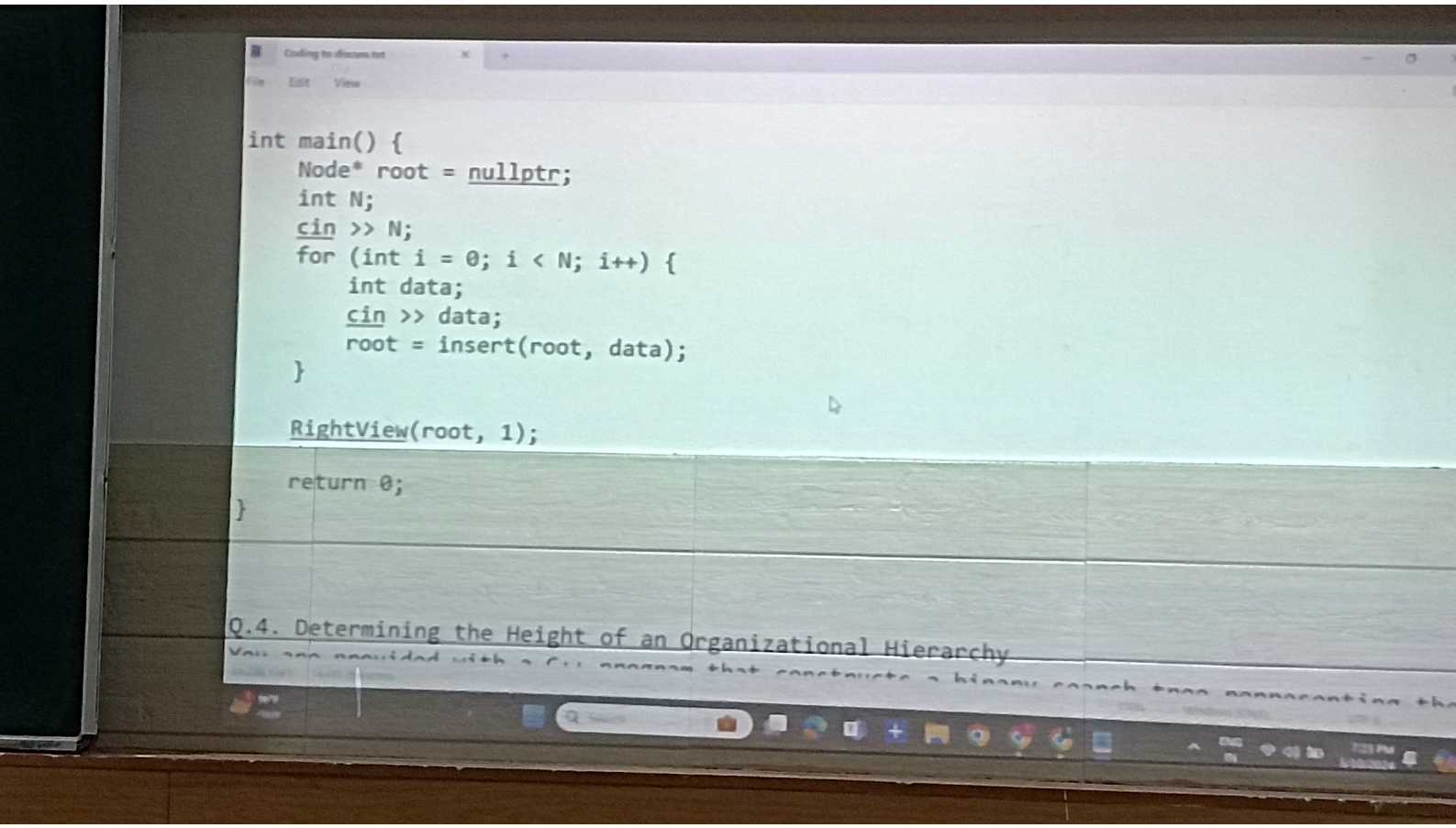
```
Solution:
#include <iostream>
#include <queue>
using namespace std;

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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

static int LH = 0;
```



```
static int LH = 0;

void RightView(Node* root, int RH) {
    if (root == nullptr)
        return;

    if (LH < RH) {
        cout << root->data << " ";
        LH = RH;
    }

    RightView(root->right, RH + 1);
    RightView(root->left, RH + 1);
}

Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);
```



```
Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
```


Coding to discuss.txt
File Edit View
int main() {
Node* root = nullptr;
int N;
cin >> N;
for (int i = 0; i < N; i++) {
int data;
cin >> data;
root = insert(root, data);
}
RightView(root, 1);
return 0;
}
Q.4. Determining the Height of an Organizational Hierarchy

Coding to discuss.txt

File Edit View

Q.4. Determining the Height of an Organizational Hierarchy

You are provided with a C++ program that constructs a binary search tree representing the organizational hierarchy of a company based on user input. The program defines a Node class representing individual employees in the hierarchy and includes a Height method that calculates and returns the height of the organizational tree.

Sample Testcase 1

Input:

10

10 5 15 3 7 12 18 20 2 9

Output:

4

Sample Testcase 2:

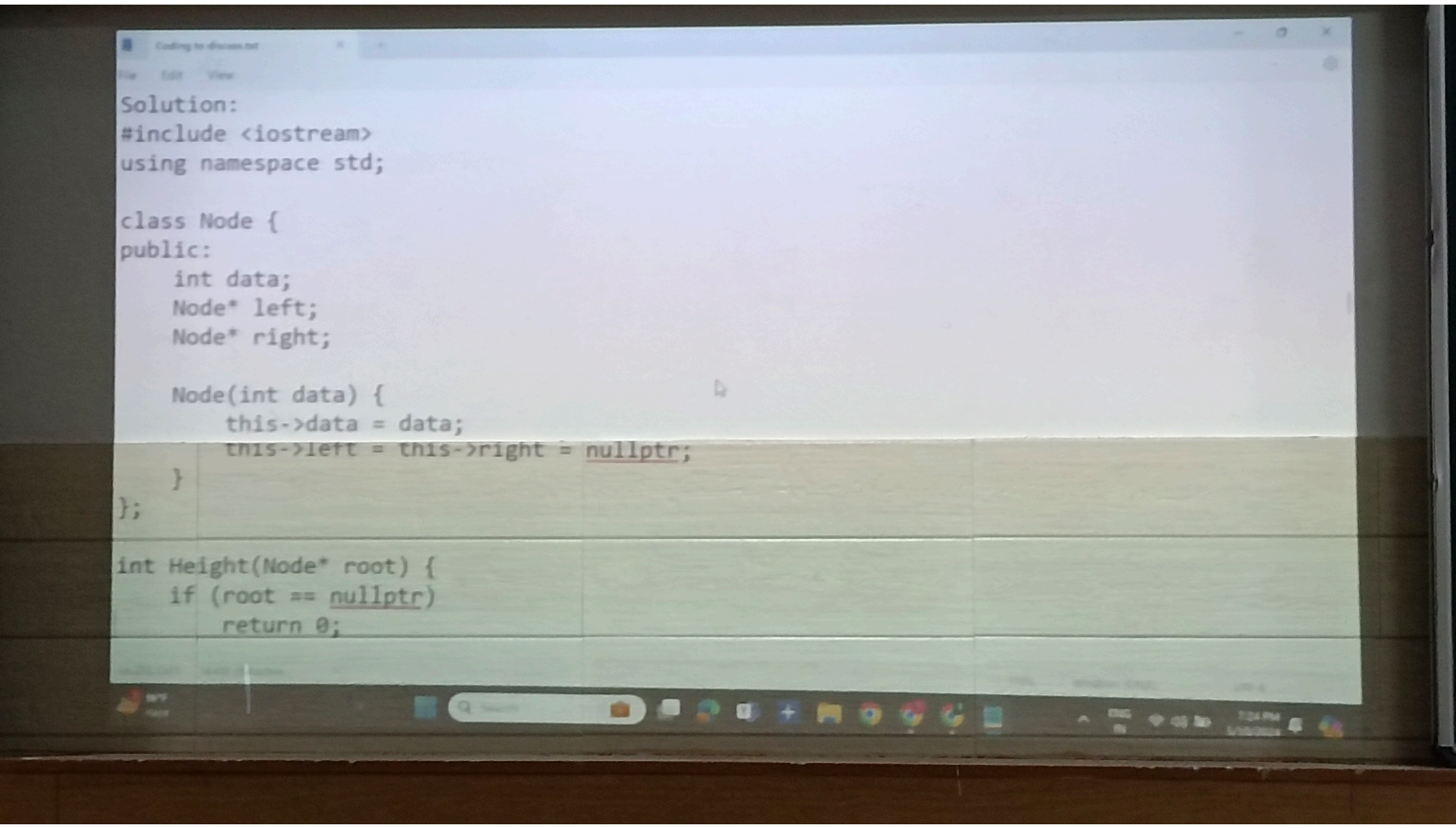
Input:

6

5 3 8 2 7 9

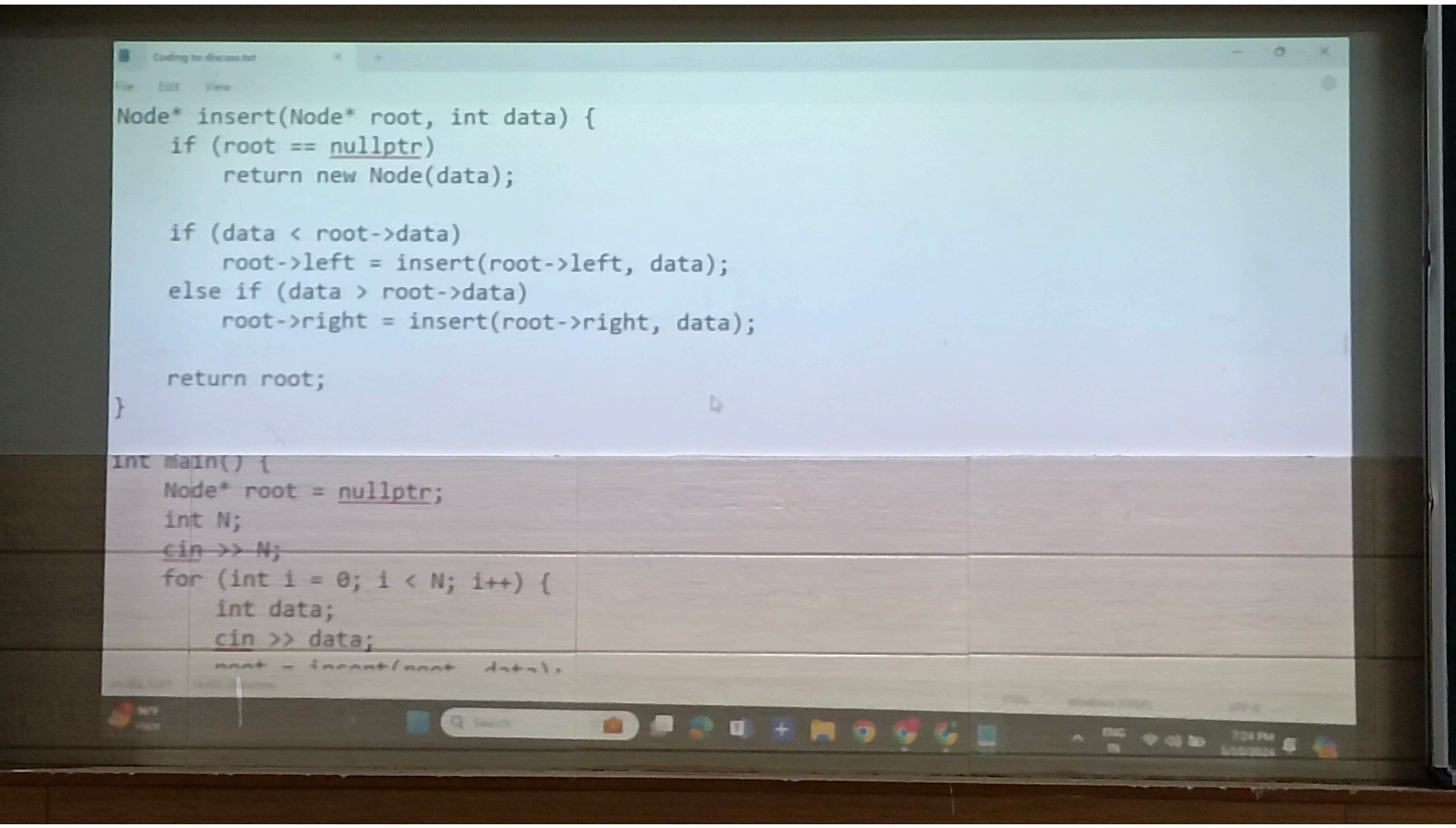
Output:

3



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

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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

int Height(Node* root) {
    if (root == nullptr)
        return 0;
```

```
int Height(Node* root) {
    if (root == nullptr)
        return 0;

    int leftHeight = Height(root->left);
    int rightHeight = Height(root->right);

    if (leftHeight > rightHeight)
        return (leftHeight + 1);
    else
        return (rightHeight + 1);
}

Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);

    if (data < root->data)
        root->left = insert(root->left, data);
```


Coding to discuss.txt
File Edit View
Node* insert(Node* root, int data) {
if (root == nullptr)
return new Node(data);
if (data < root->data)
root->left = insert(root->left, data);
else if (data > root->data)
root->right = insert(root->right, data);
return root;
}
int main() {
Node* root = nullptr;
int N;
cin >> N;
for (int i = 0; i < N; i++) {
int data;
cin >> data;

```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
    }
    cout << Height(root) << endl;

    return 0;
}
```

Q.5. Left View of a Binary Search Tree

You are provided with a C++ program that constructs a binary search tree based on user input and identifies and prints the left view of the tree. The program defines a Node class representing individual nodes in the tree and includes a LeftView method that prints the values of nodes visible from the right side of the tree.

Q.5. Left View of a Binary Search Tree

You are provided with a C++ program that constructs a binary search tree based on user input and identifies and prints the left view of the tree. The program defines a Node class representing individual nodes in the tree and includes a LeftView method that prints the values of nodes visible from the right side of the tree.

Sample Testcase 1

Input:

9

8 3 10 1 6 4 7 13 14

Output:

8 3 1 4

Sample Testcase 2:

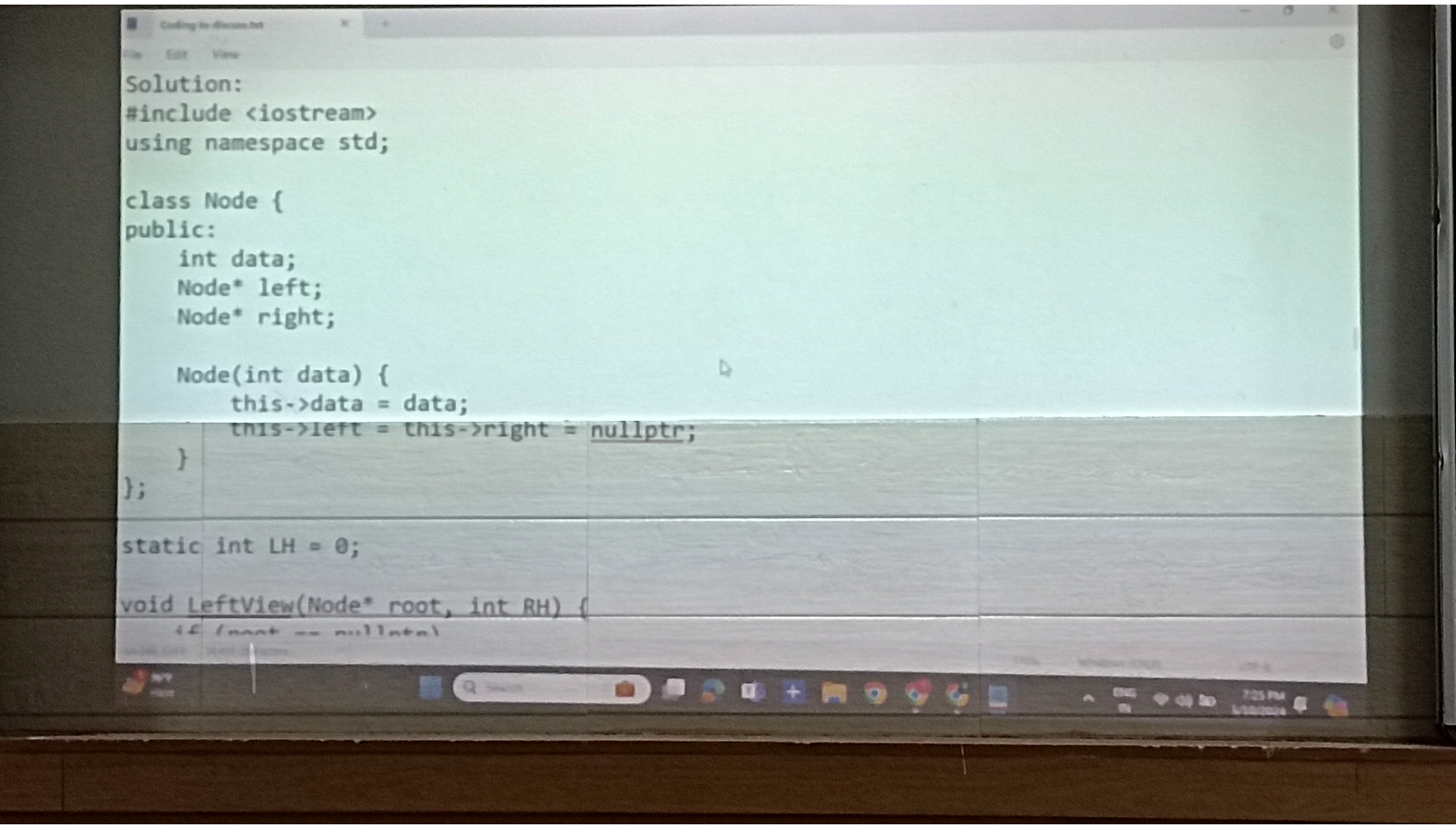
Input:

7

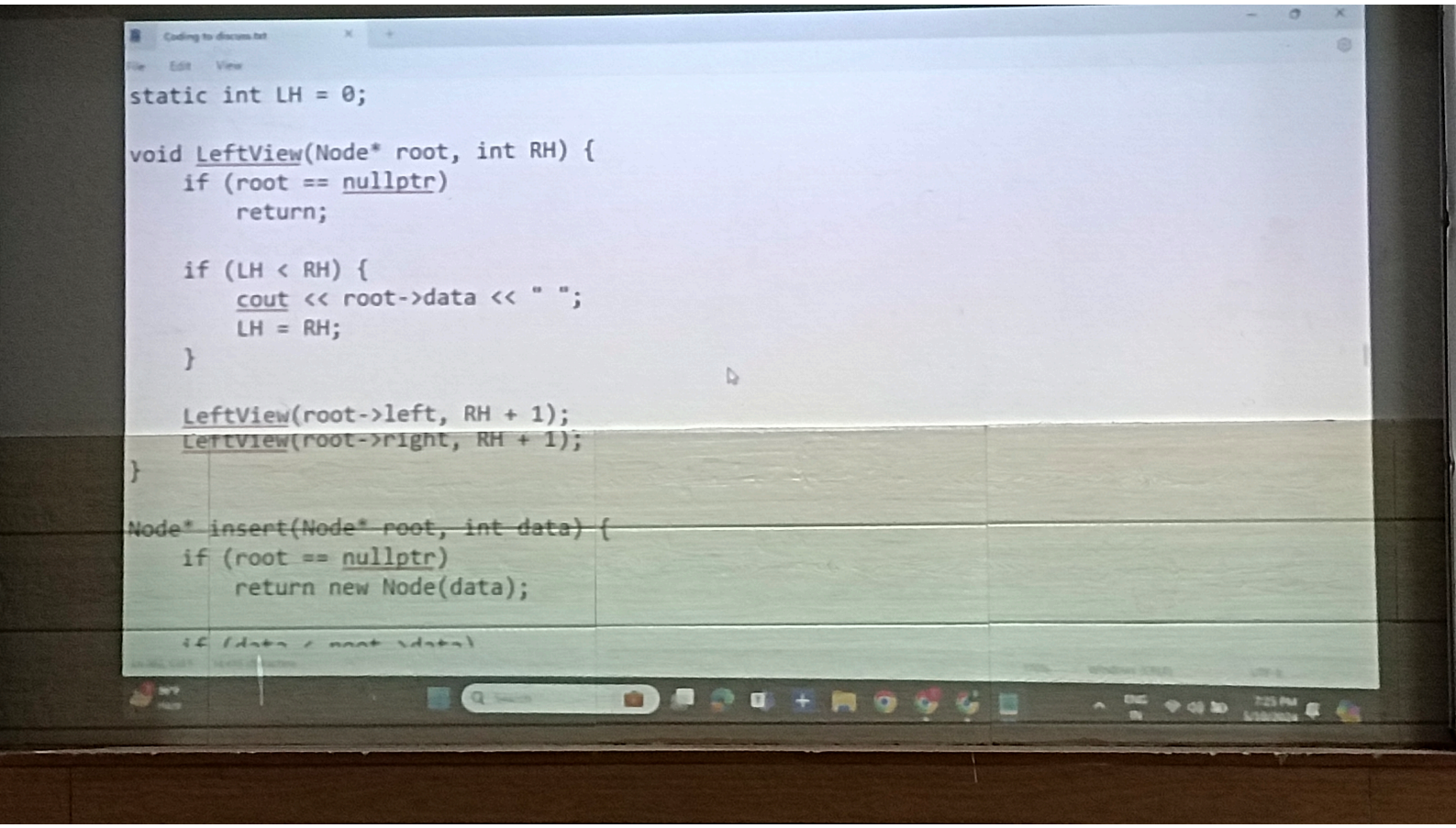
10 5 15 3 7 12 18

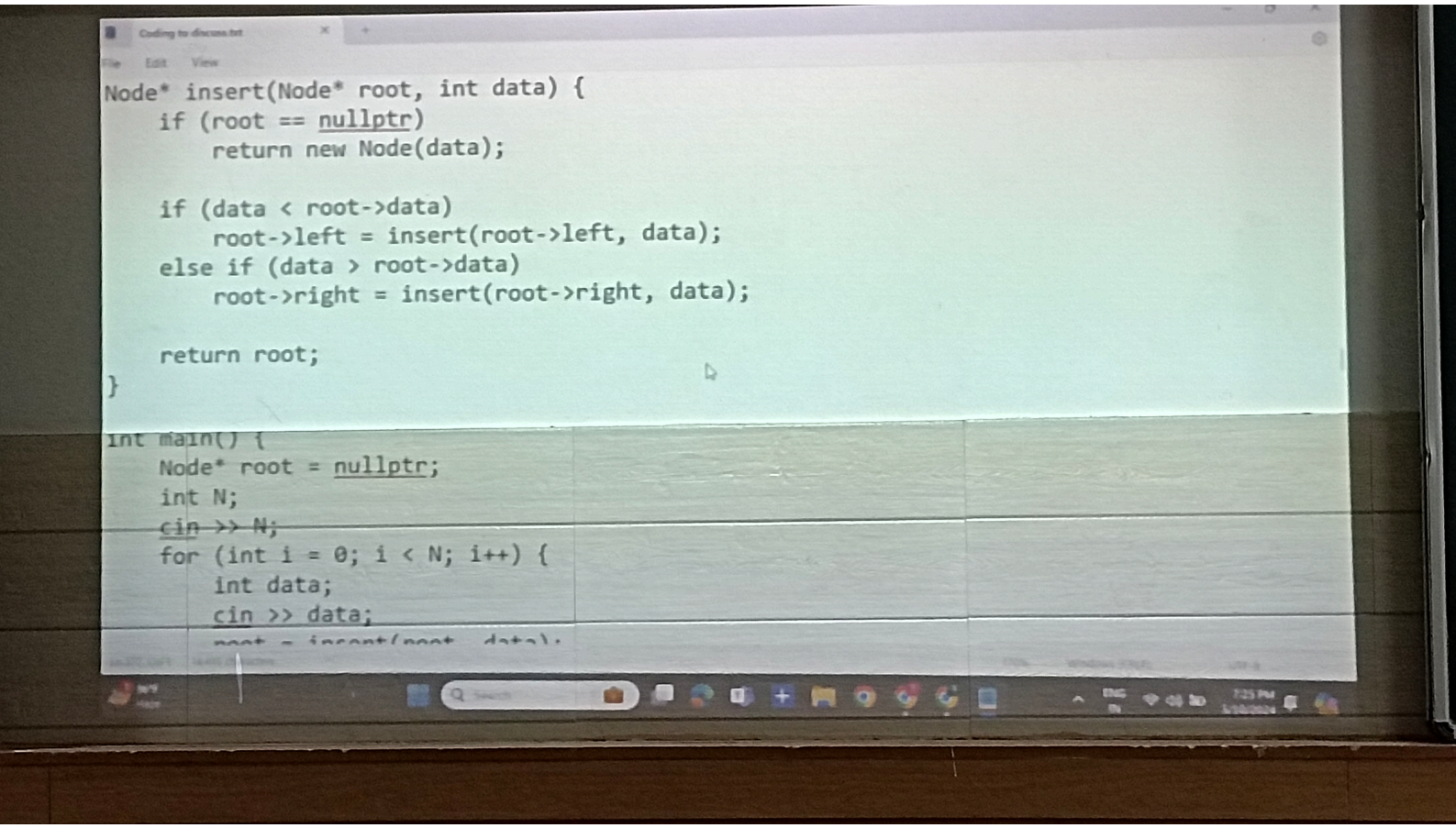
Output:

10 5 3



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

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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

static int LH = 0;

void LeftView(Node* root, int RH) {
```


Coding to discuss.txt
File Edit View
static int LH = 0;
void LeftView(Node* root, int RH) {
if (root == nullptr)
return;
if (LH < RH) {
cout << root->data << " ";
LH = RH;
}
LeftView(root->left, RH + 1);
LeftView(root->right, RH + 1);
}
Node* insert(Node* root, int data) {
if (root == nullptr)
return new Node(data);


Coding to discuss.txt
File Edit View
Node* insert(Node* root, int data) {
if (root == nullptr)
return new Node(data);
if (data < root->data)
root->left = insert(root->left, data);
else if (data > root->data)
root->right = insert(root->right, data);
return root;
}
int main() {
Node* root = nullptr;
int N;
cin >> N;
for (int i = 0; i < N; i++) {
int data;
cin >> data;

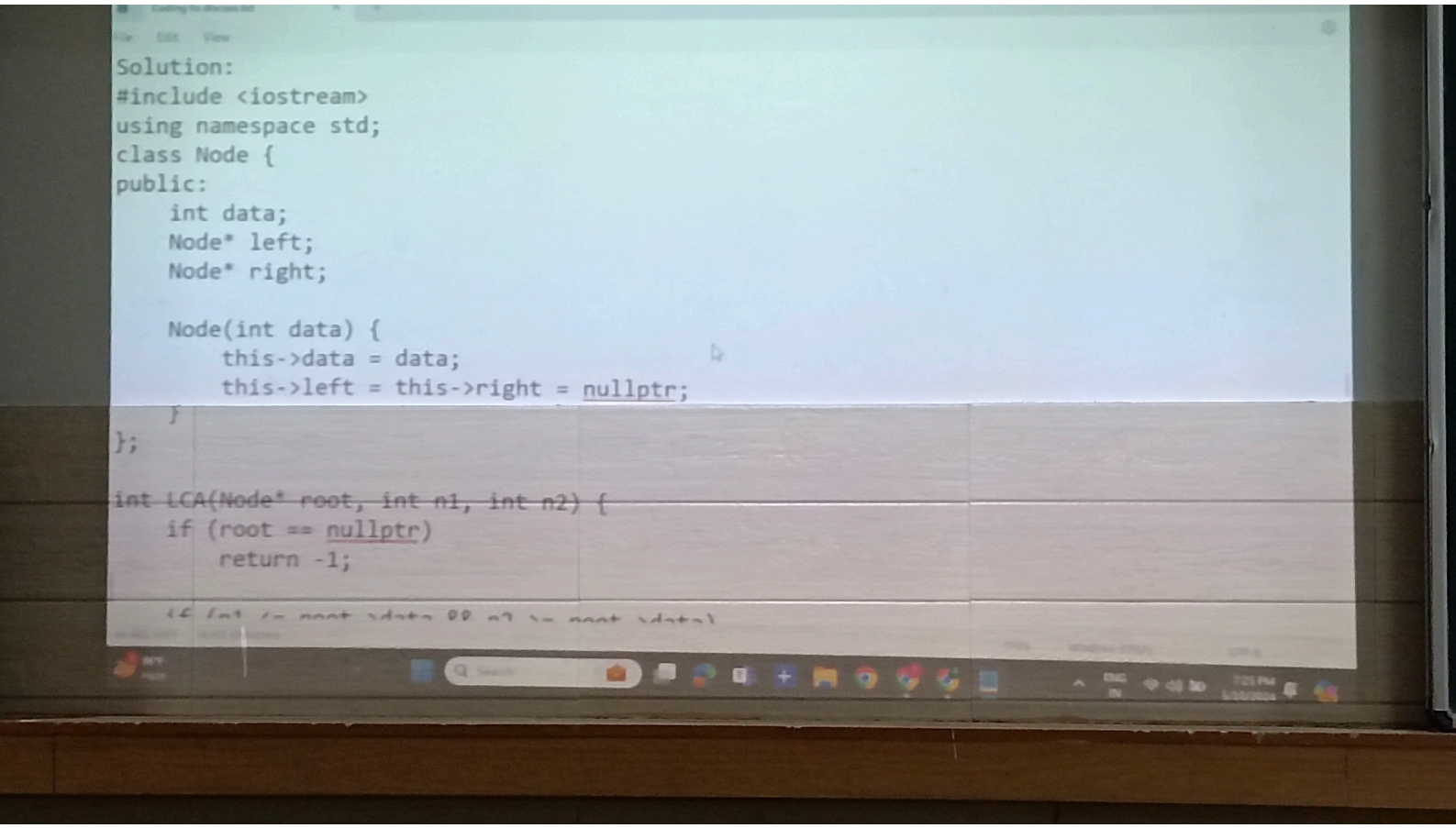
```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
    }

    LeftView(root, 1);

    return 0;
}
```

Q.6. Finding the Lowest Common Ancestor (LCA) in a Binary Search Tree (BST)
Consider a community where individuals are represented in a binary search tree based on their familial relationships. Each individual is identified by a unique numerical value. The community wants to determine the familial relationship between two individuals by finding their Lowest Common Ancestor (LCA) in the community's family tree.

Q.6. Finding the Lowest Common Ancestor (LCA) in a Binary Search Tree (BST)
Consider a community where individuals are represented in a binary search tree based on their familial relationships. Each individual is identified by a unique numerical value. The community wants to determine the familial relationship between two individuals by finding their Lowest Common Ancestor (LCA) in the community's family tree.

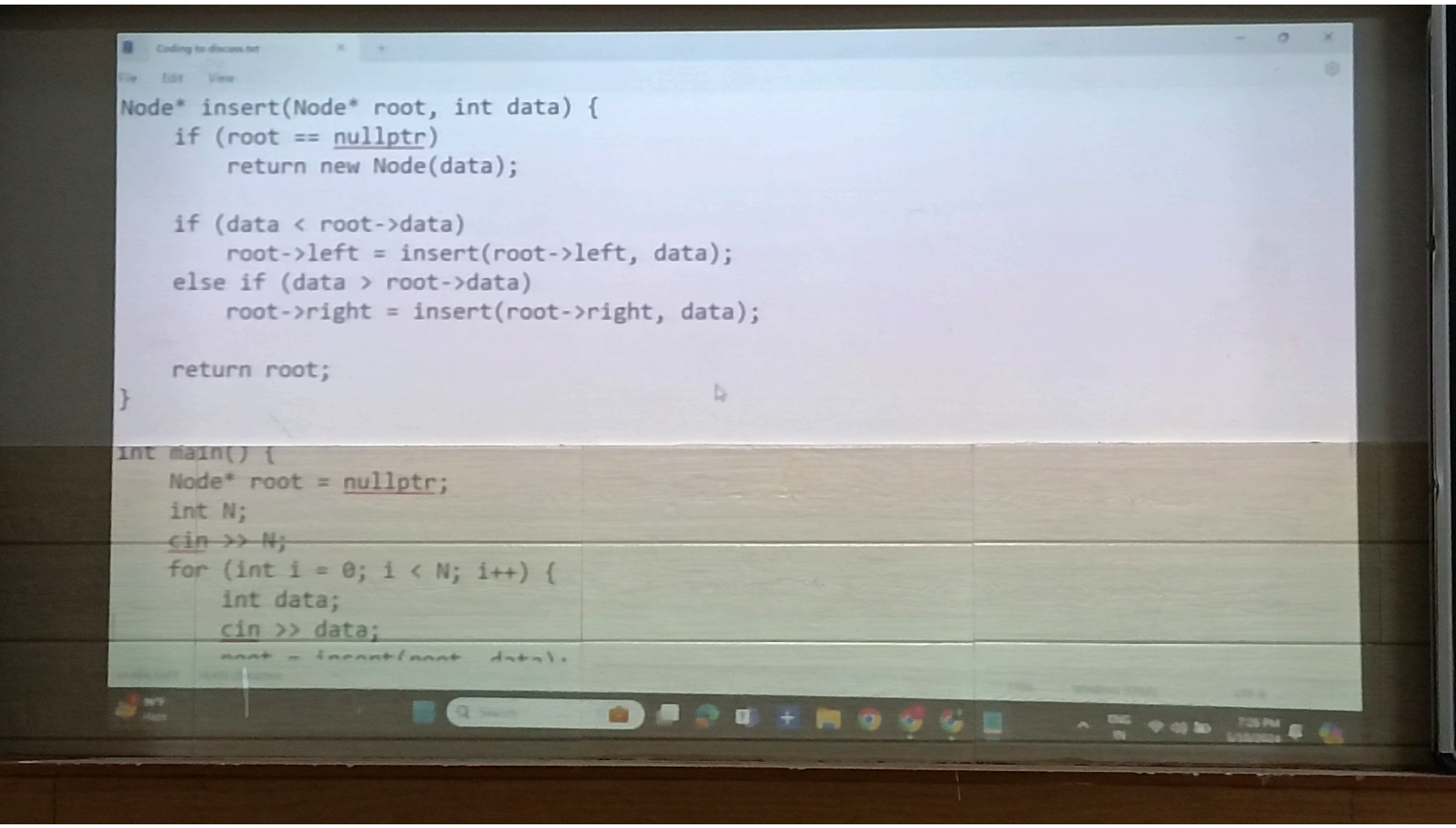
Sample Testcase 1
Input:
9
8 3 10 1 6 4 7 13 14
3 14
Output:
8

Sample Testcase 2:
Input:
8
8 3 10 1 6 4 7 13
14 12



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

    Node(int data) {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

int LCA(Node* root, int n1, int n2) {
    if (root == nullptr)
        return -1;
```

```
int LCA(Node* root, int n1, int n2) {
    if (root == nullptr)
        return -1;

    if (n1 <= root->data && n2 >= root->data)
        return root->data;

    else if (n1 < root->data && n2 < root->data)
        return LCA(root->left, n1, n2);

    else if (n1 > root->data && n2 > root->data)
        return LCA(root->right, n1, n2);

    return -1;
}

Node* insert(Node* root, int data) {
    if (root == nullptr)
        return new Node(data);
```


Coding to discuss.txt
File Edit View
Node* insert(Node* root, int data) {
if (root == nullptr)
return new Node(data);
if (data < root->data)
root->left = insert(root->left, data);
else if (data > root->data)
root->right = insert(root->right, data);
return root;
}
int main() {
Node* root = nullptr;
int N;
cin >> N;
for (int i = 0; i < N; i++) {
int data;
cin >> data;

```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int data;
        cin >> data;
        root = insert(root, data);
    }
    int N1, N2;
    cin >> N1 >> N2;
    cout << LCA(root, N1, N2);
    return 0;
}
```

Q.7. Directory Structure Management

Suppose you are tasked with implementing a program that manages a directory structure where each node represents a folder containing a numerical ID. The program allows users to insert new foldons into the structure and perform various traversals on the tree. You

Coding to discuss.txt

File Edit View

Q.7. Directory Structure Management

Suppose you are tasked with implementing a program that manages a directory structure where each node represents a folder containing a numerical ID. The program allows users to insert new folders into the structure and perform various traversals on the tree. You decide to use a binary search tree to efficiently manage the folders.

A user wants to organize folders in the directory structure, after inserting these folders using the provided program, the user wants to know the contents of the directory structure in different orders. (Inorder, Preorder, Postorder, Levelorder)

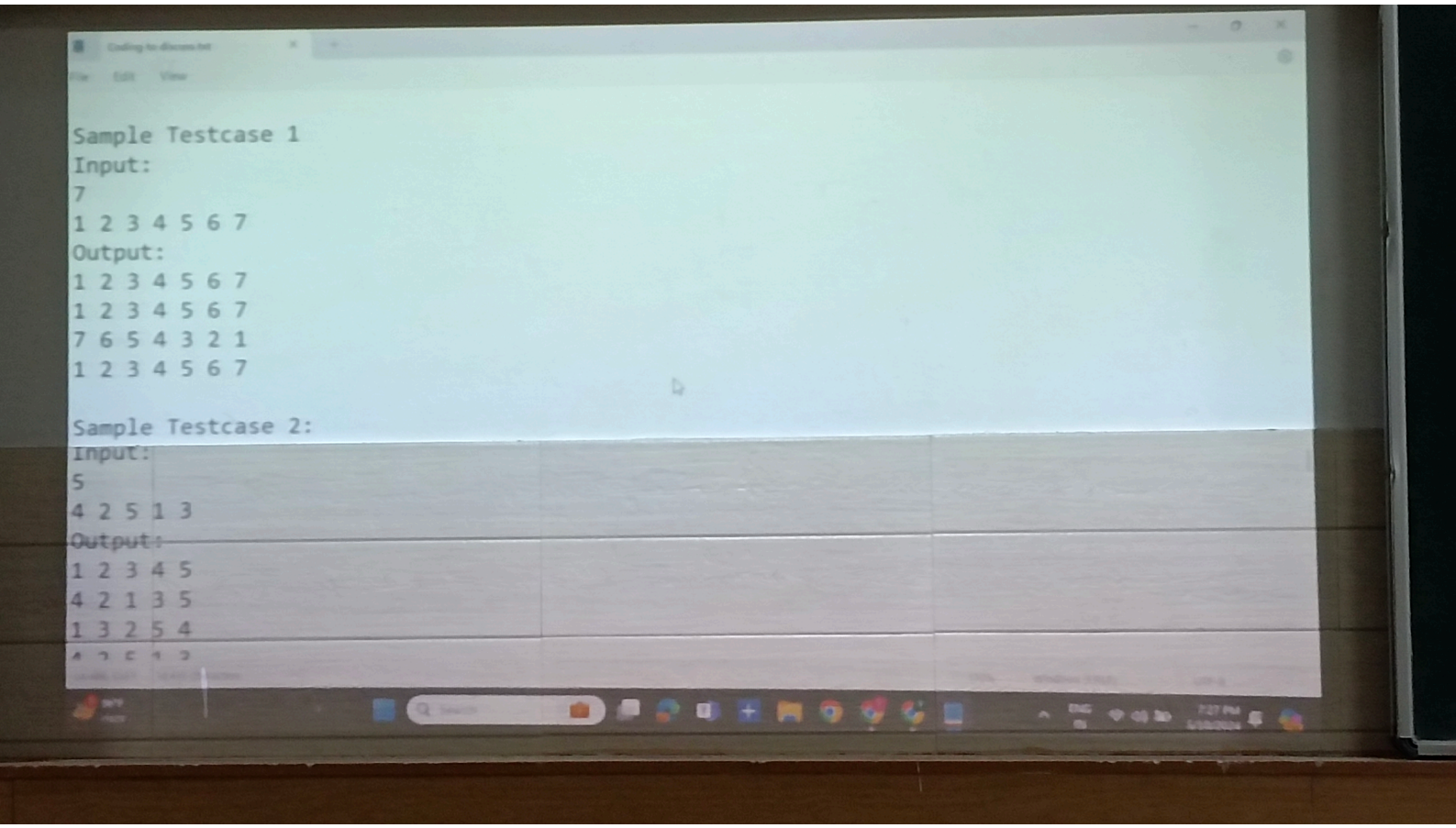
Sample Testcase 1

Input:

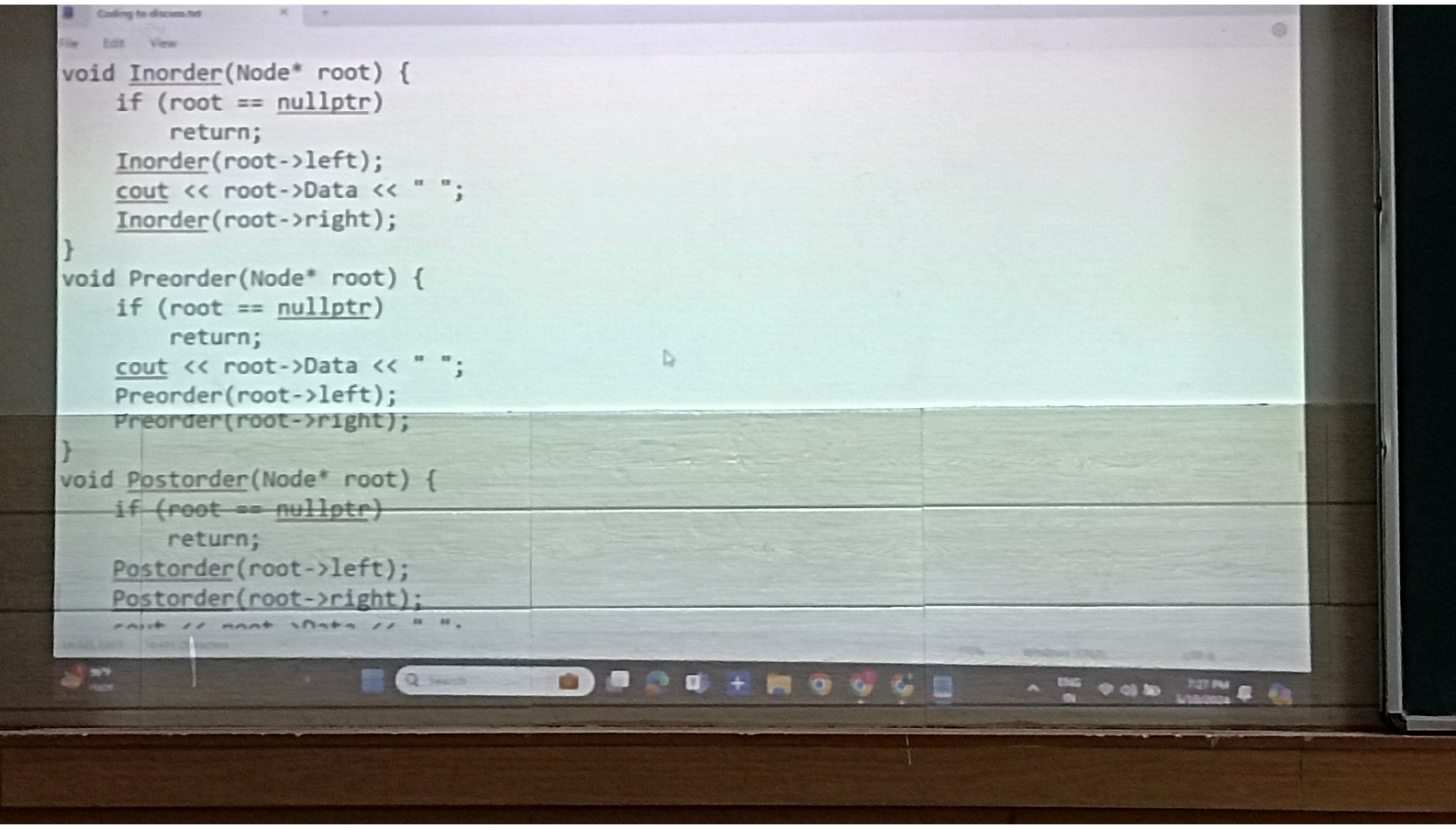
```
7
1 2 3 4 5 6 7
```

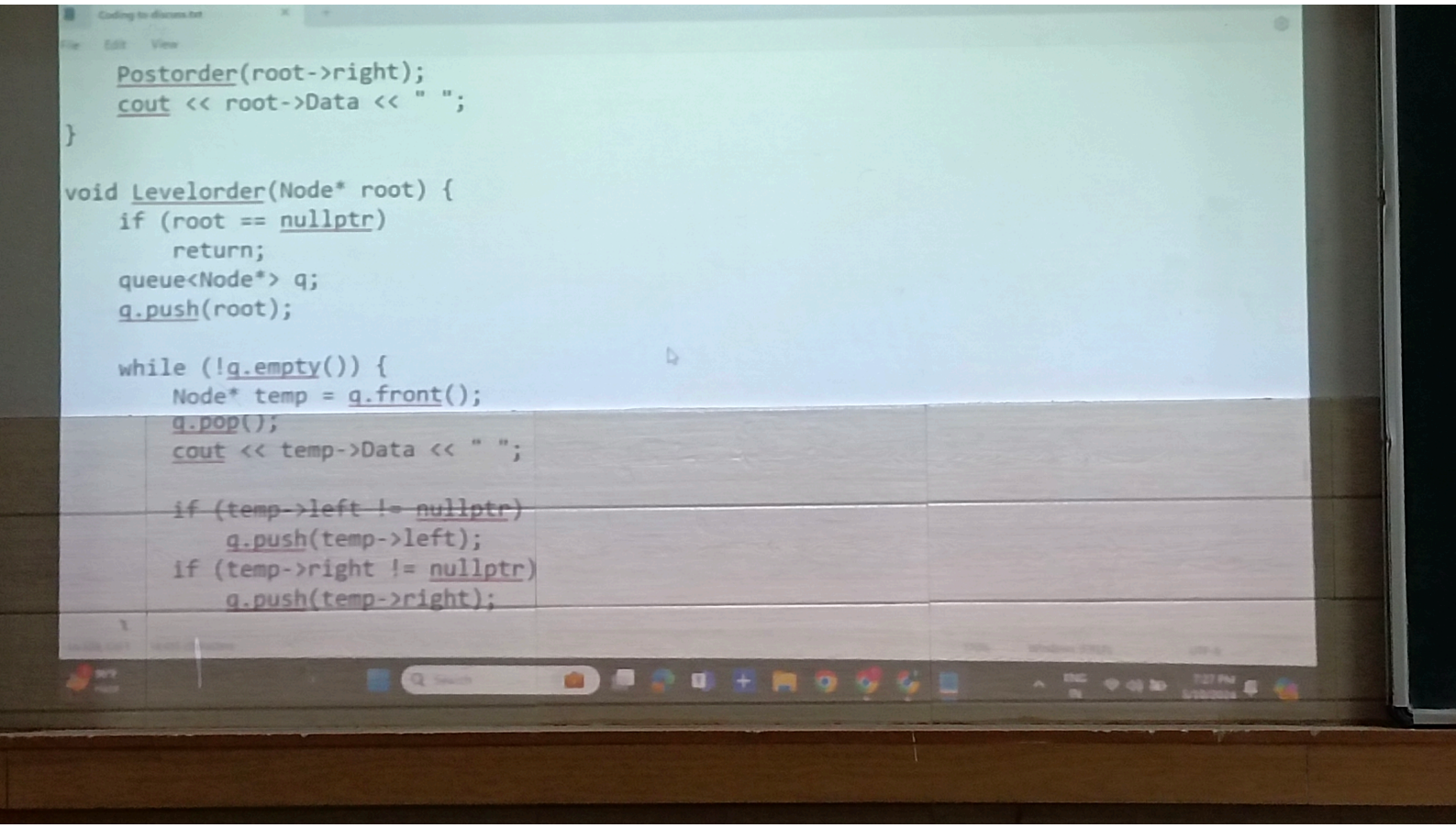
Output:

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

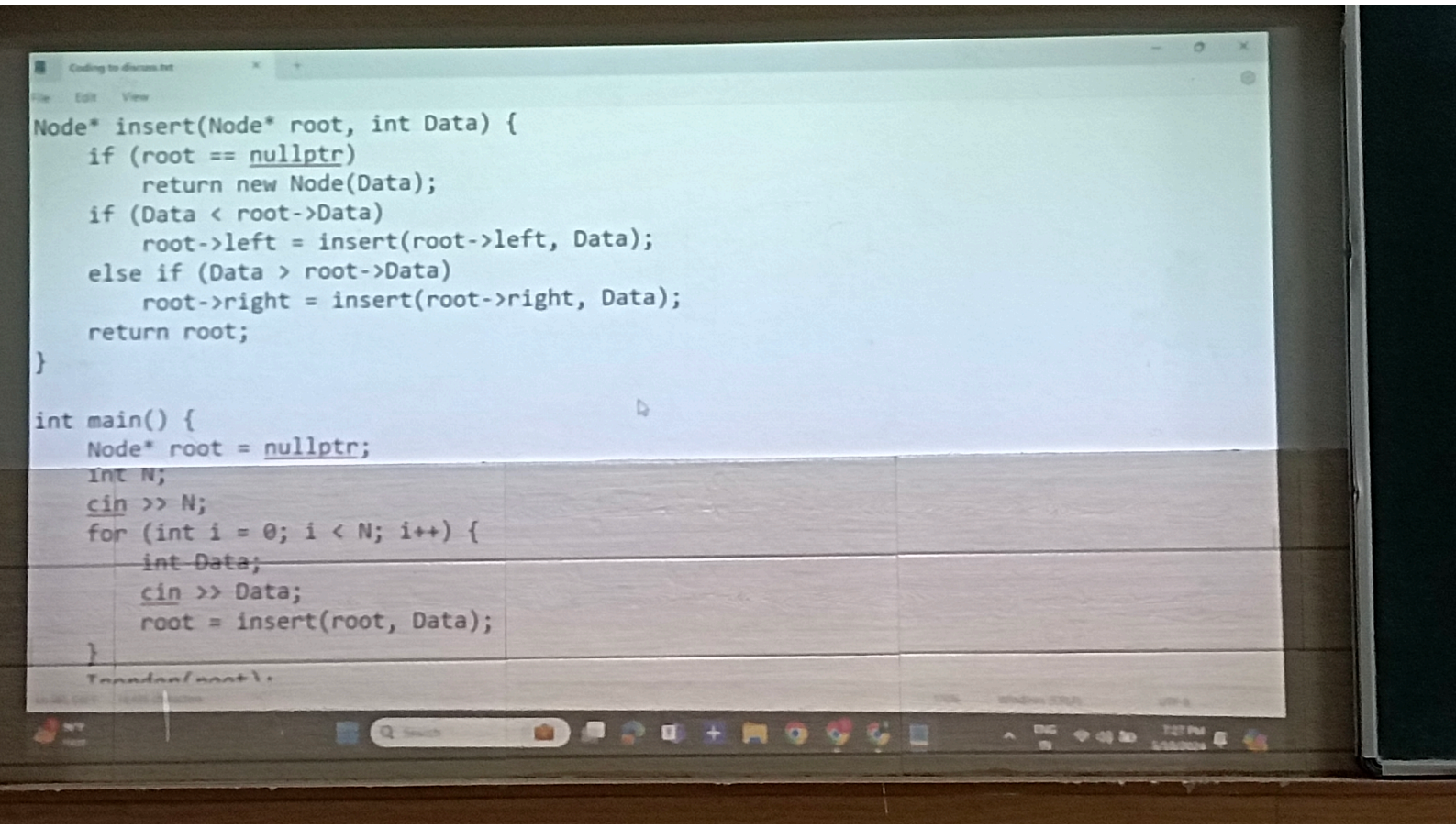

Sample Testcase 1
Input:
7
1 2 3 4 5 6 7
Output:
1 2 3 4 5 6 7
1 2 3 4 5 6 7
7 6 5 4 3 2 1
1 2 3 4 5 6 7
Sample Testcase 2:
Input:
5
4 2 5 1 3
Output:
1 2 3 4 5
4 2 1 3 5
1 3 2 5 4

```
Solution:
#include <iostream>
#include <queue>
using namespace std;
class Node {
public:
    int Data;
    Node *left, *right;
    Node(int Data) {
        this->Data = Data;
        left = right = nullptr;
    }
};
void Inorder(Node* root) {
    if (root == nullptr)
        return;
    Inorder(root->left);
    cout << root->Data << " ";
    Inorder(root->right);
}
```


Coding to discuss.txt
File Edit View
void Inorder(Node* root) {
if (root == nullptr)
return;
Inorder(root->left);
cout << root->Data << " ";
Inorder(root->right);
}
void Preorder(Node* root) {
if (root == nullptr)
return;
cout << root->Data << " ";
Preorder(root->left);
Preorder(root->right);
}
void Postorder(Node* root) {
if (root == nullptr)
return;
Postorder(root->left);
Postorder(root->right);



```
    Postorder(root->right);
    cout << root->Data << " ";
}

void Levelorder(Node* root) {
    if (root == nullptr)
        return;
    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        Node* temp = q.front();
        q.pop();
        cout << temp->Data << " ";

        if (temp->left != nullptr)
            q.push(temp->left);
        if (temp->right != nullptr)
            q.push(temp->right);
```



```
Node* insert(Node* root, int Data) {
    if (root == nullptr)
        return new Node(Data);
    if (Data < root->Data)
        root->left = insert(root->left, Data);
    else if (Data > root->Data)
        root->right = insert(root->right, Data);
    return root;
}

int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int Data;
        cin >> Data;
        root = insert(root, Data);
    }
```

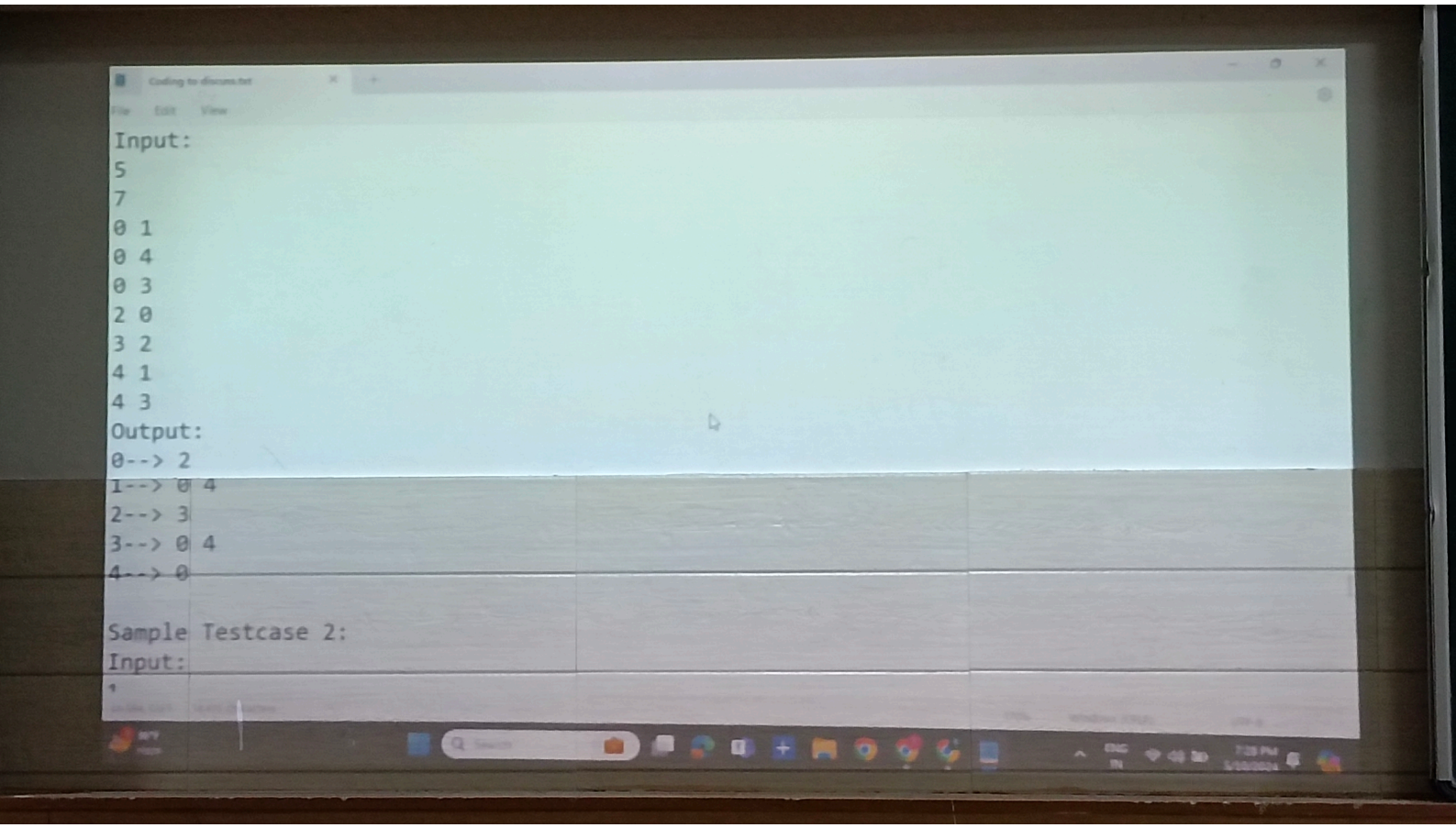
```
int main() {
    Node* root = nullptr;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int Data;
        cin >> Data;
        root = insert(root, Data);
    }
    Inorder(root);
    cout << endl;
    Preorder(root);
    cout << endl;
    Postorder(root);
    cout << endl;
    Levelorder(root);
    return 0;
}
```

Coding to discuss.txt

File Edit View

Q.8. Utilizing Graph Transpose for Friend Recommendations in Social Networks
Suppose you are working on a social network platform where users are represented as vertices in a graph, and friendships between users are represented as edges. You are tasked with implementing a feature that allows users to see their friend recommendations based on mutual friends. To achieve this, you decide to use the transpose of the friendship graph.

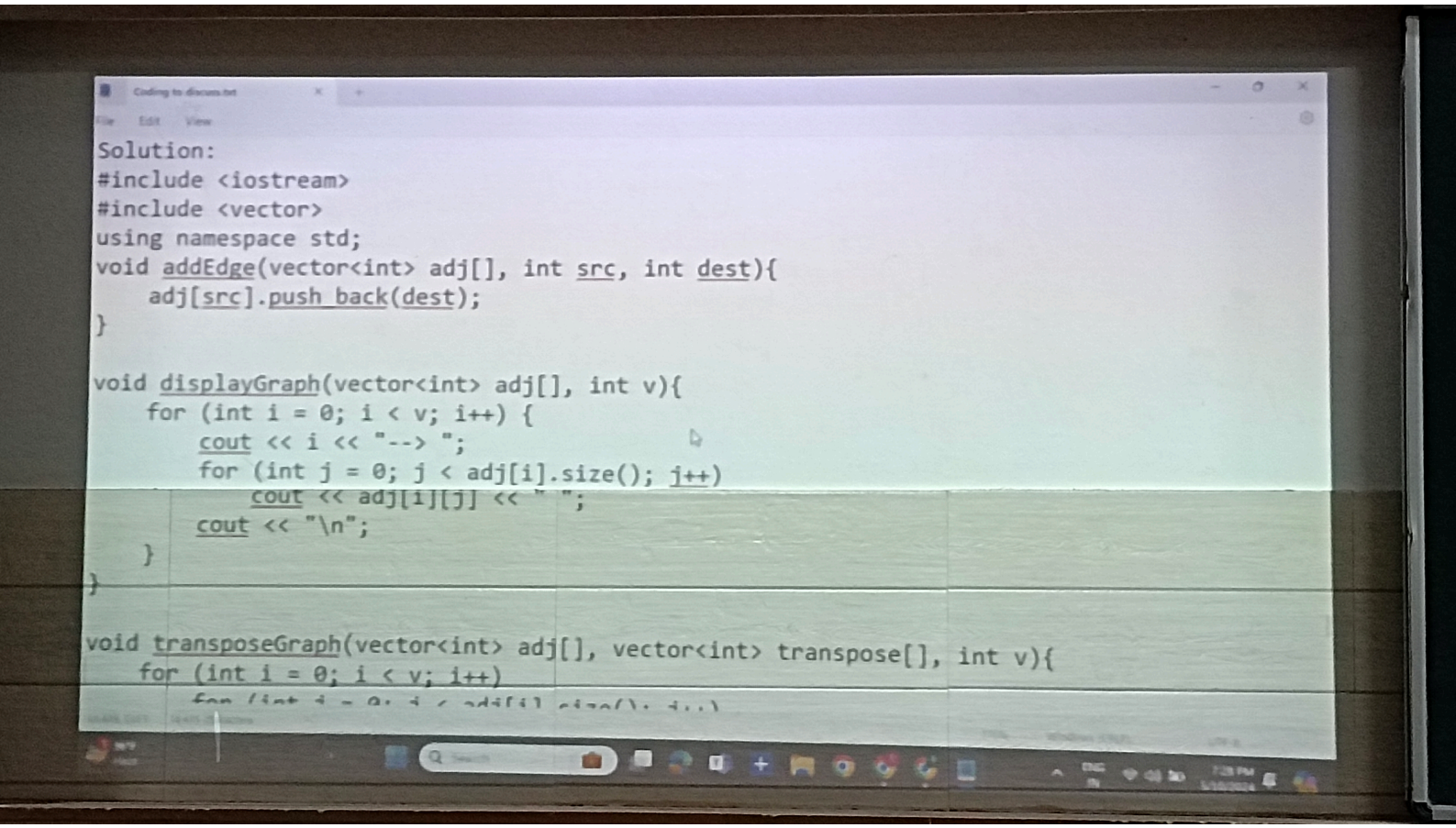
Sample Testcase 1
Input:
5
7
0 1
0 4
0 3
2 0
3 2
4 1
4 3
Output:



Coding to discuss.txt

File Edit View

```
Input:
5
7
0 1
0 4
0 3
2 0
3 2
4 1
4 3
Output:
0--> 2
1--> 0 4
2--> 3
3--> 0 4
4--> 0

Sample Testcase 2:
Input:
```

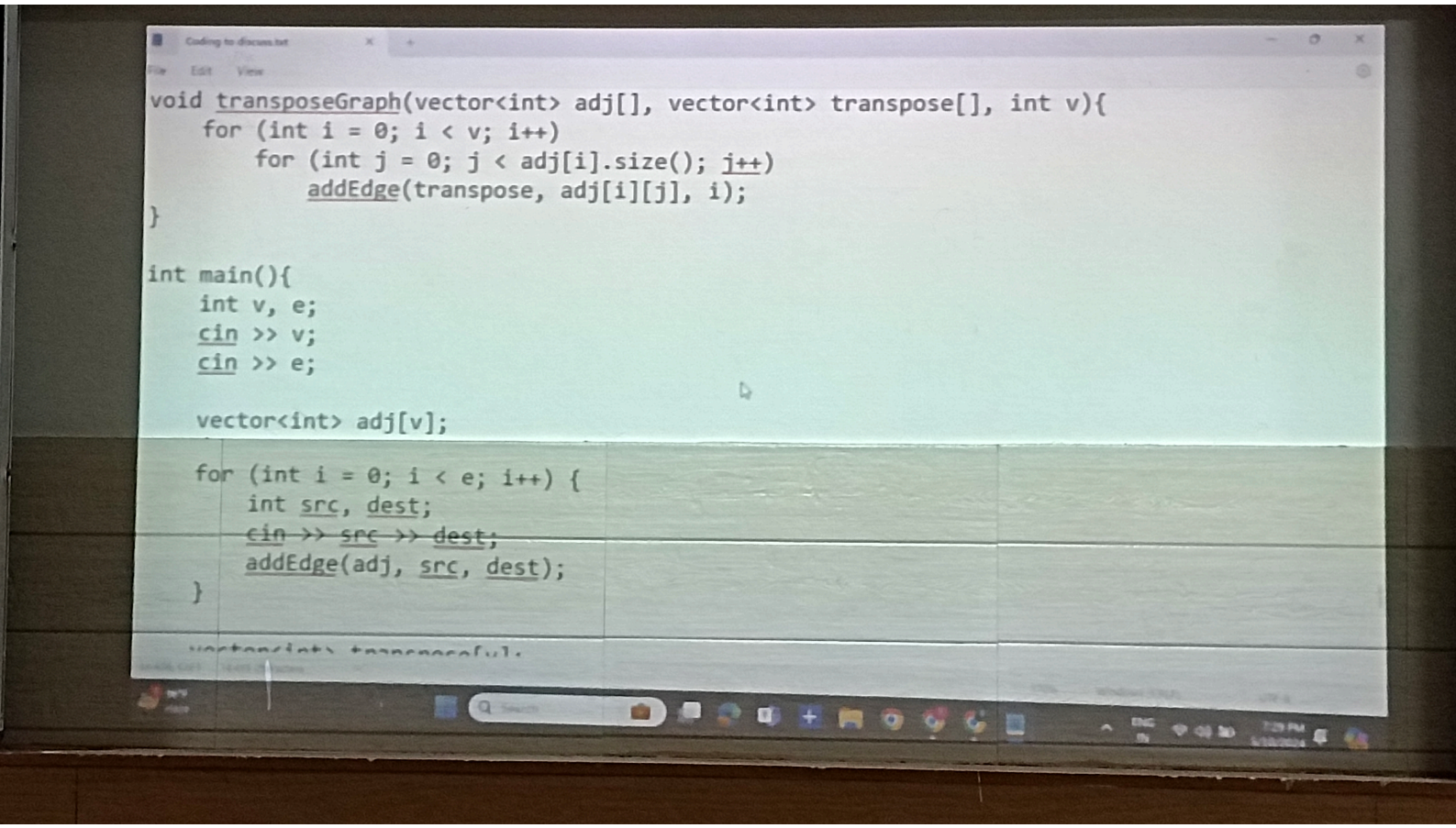


Coding to discuss.txt

File Edit View

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
void addEdge(vector<int> adj[], int src, int dest){
    adj[src].push_back(dest);
}

void displayGraph(vector<int> adj[], int v){
    for (int i = 0; i < v; i++) {
        cout << i << "--> ";
        for (int j = 0; j < adj[i].size(); j++)
            cout << adj[i][j] << " ";
        cout << "\n";
    }
}

void transposeGraph(vector<int> adj[], vector<int> transpose[], int v){
    for (int i = 0; i < v; i++)
        for (int i = 0; i < adj[i].size(); i++)
```


Coding to discuss.txt
File Edit View
void transposeGraph(vector<int> adj[], vector<int> transpose[], int v){
for (int i = 0; i < v; i++)
for (int j = 0; j < adj[i].size(); j++)
addEdge(transpose, adj[i][j], i);
}
int main(){
int v, e;
cin >> v;
cin >> e;
vector<int> adj[v];
for (int i = 0; i < e; i++) {
int src, dest;
cin >> src >> dest;
addEdge(adj, src, dest);
}

```
    cin >> e;

    vector<int> adj[v];

    for (int i = 0; i < e; i++) {
        int src, dest;
        cin >> src >> dest;
        addEdge(adj, src, dest);
    }

    vector<int> transpose[v];
    transposeGraph(adj, transpose, v);
    displayGraph(transpose, v);
    return 0;
}
```

Q.9. Utilizing Graph Analysis for Traffic Flow Management

Suppose you are designing a system to analyze traffic flow in a city using a directed graph where each vertex represents an intersection and each edge represents a one-way

Coding to discuss.txt

File Edit View

Q.9. Utilizing Graph Analysis for Traffic Flow Management

Suppose you are designing a system to analyze traffic flow in a city using a directed graph where each vertex represents an intersection and each edge represents a one-way street connecting two intersections. You have implemented a C++ program to calculate the in-degree and out-degree of a specific intersection based on user input of the city's road network.

Sample Testcase 1

Input:

6

7

0 1

0 2

1 2

1 3

2 3

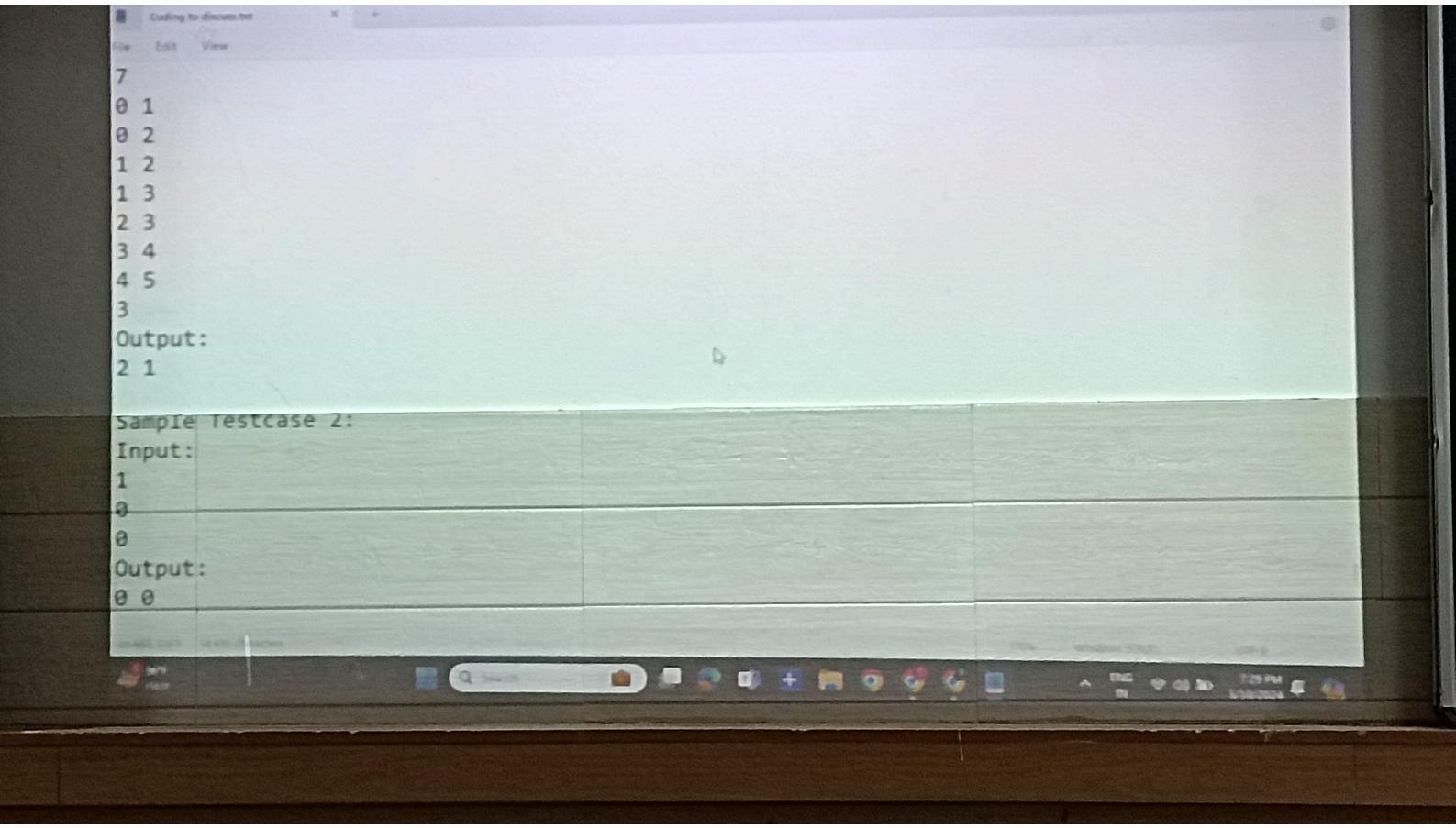
3 4

4 5

3

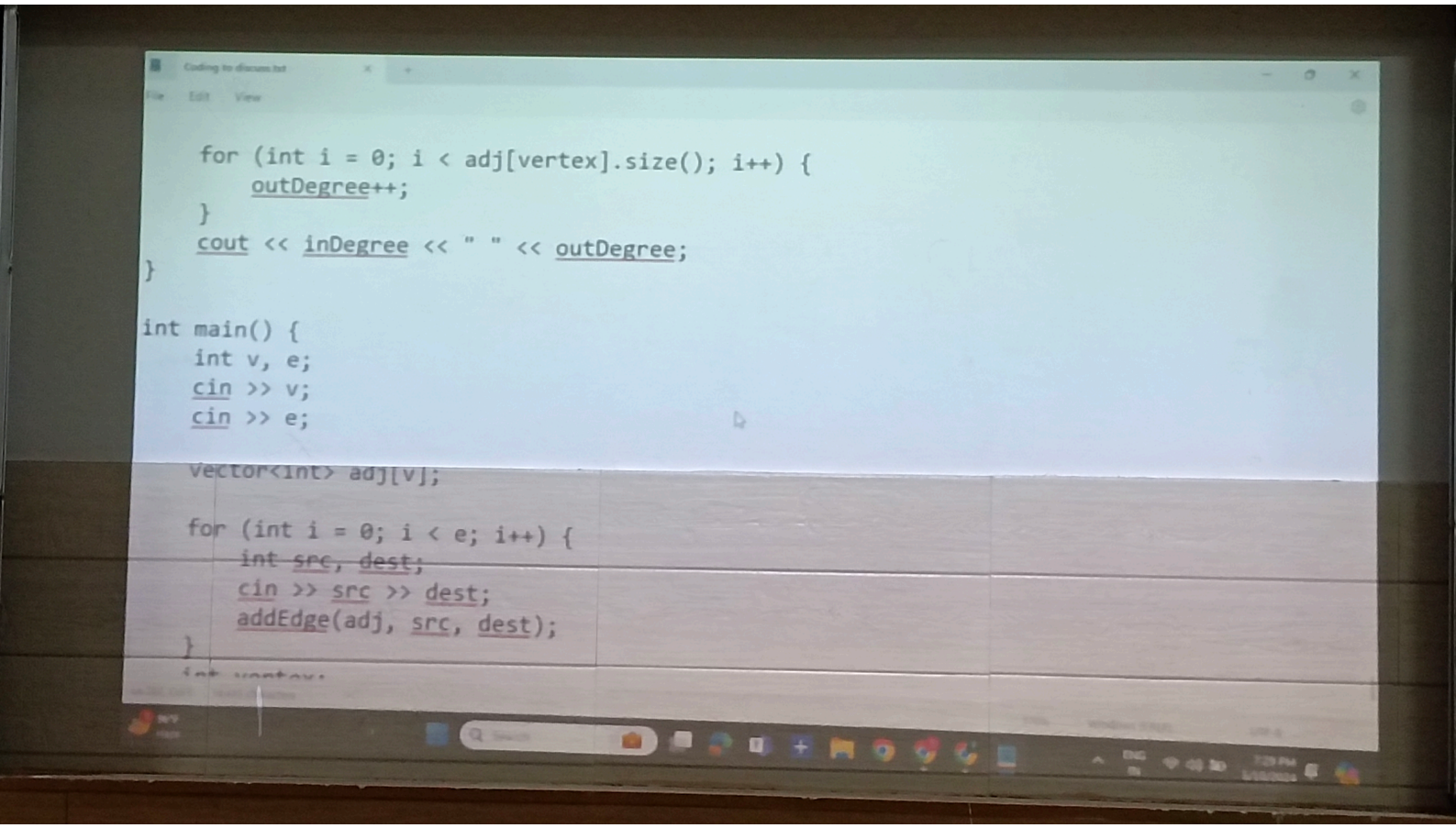
Output:

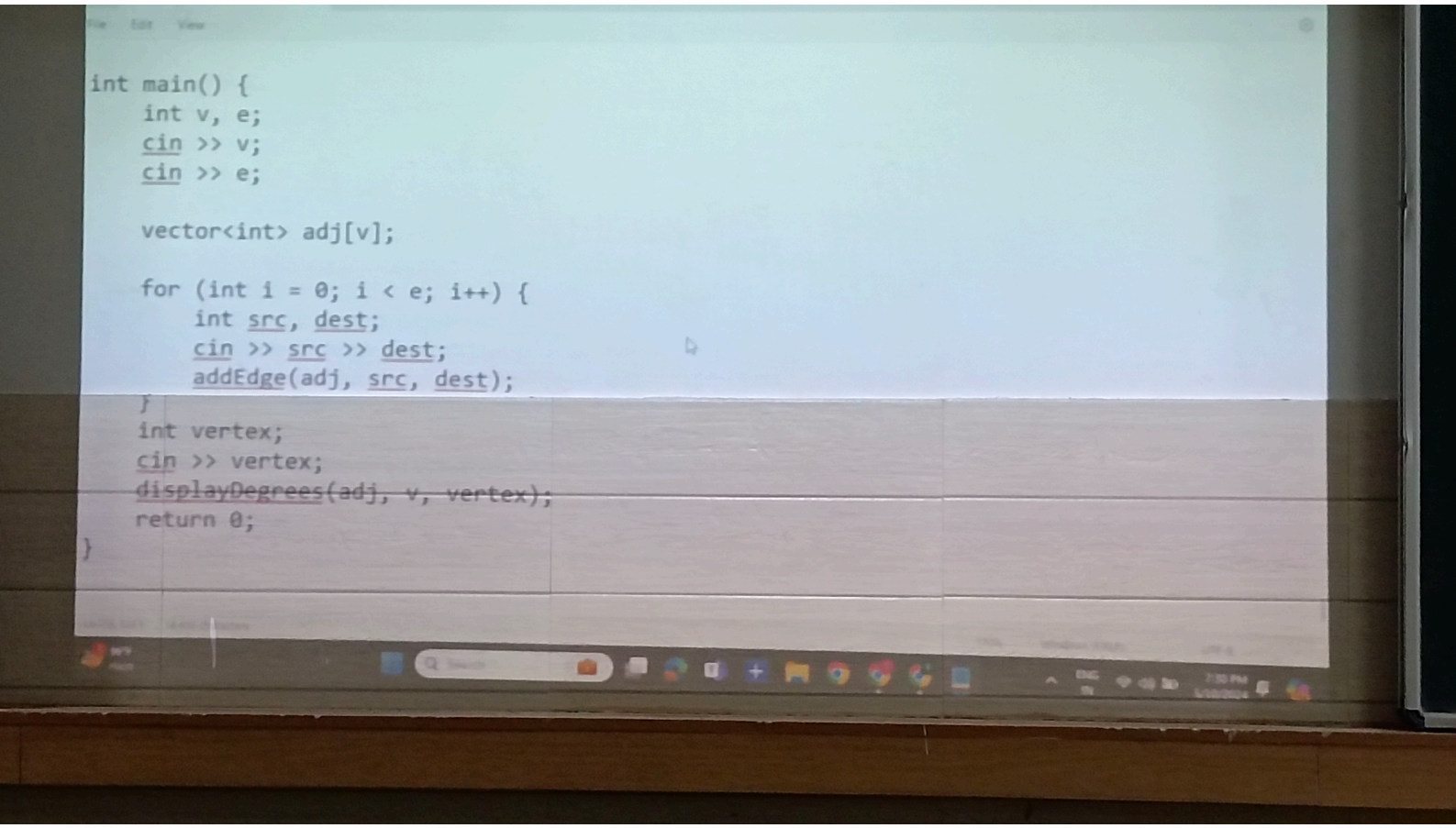
2 1


Coding to discuss.txt
File Edit View
7
0 1
0 2
1 2
1 3
2 3
3 4
4 5
3
Output:
2 1
Sample Testcase 2:
Input:
1
0
0
Output:
0 0

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
void addEdge(vector<int> adj[], int src, int dest) {
    adj[src].push_back(dest);
}

void displayDegrees(vector<int> adj[], int v, int vertex) {
    int inDegree = 0;
    int outDegree = 0;

    for (int i = 0; i < v; i++) {
        for (int j = 0; j < adj[i].size(); j++) {
            int dest = adj[i][j];
            if (dest == vertex) {
                inDegree++;
            }
        }
    }
```


Coding to discuss.txt
File Edit View
for (int i = 0; i < adj[vertex].size(); i++) {
outDegree++;
}
cout << inDegree << " " << outDegree;
}
int main() {
int v, e;
cin >> v;
cin >> e;
vector<int> adj[v];
for (int i = 0; i < e; i++) {
int src, dest;
cin >> src >> dest;
addEdge(adj, src, dest);
}



```
int main() {
    int v, e;
    cin >> v;
    cin >> e;

    vector<int> adj[v];

    for (int i = 0; i < e; i++) {
        int src, dest;
        cin >> src >> dest;
        addEdge(adj, src, dest);
    }
    int vertex;
    cin >> vertex;
    displayDegrees(adj, v, vertex);
    return 0;
}
```