**LAB -04**

**Task -01:**

**Q1-Implement AVL tree-**

#include <iostream>

using namespace std;

class Node{

    public:

    int data;

    int height;

    Node\* left;

    Node\* right;

    Node(int d){

        data = d;

        left = NULL;

        right = NULL;

        height = 1;

    }

};

int getheight(Node\* root){

    if(root==NULL)return 0;

    return root->height;

}

int balancefactor(Node\* root){

    if(root==NULL)return 0;

    getheight(root->right) -getheight(root->left);

}

void updateHeight(Node\* node){

    node->height = 1 + max(getheight(node->left) , getheight(node->right));

}

Node\* rightRotate(Node\* y) {

    Node\* x = y->left;

    Node\* T2 = x->right;

    x->right = y;

    y->left = T2;

    updateHeight(y);

    updateHeight(x);

    return x;

}

Node\* leftRotate(Node\* x) {

    Node\* y = x->right;

    Node\* T2 = y->left;

    y->left = x;

    x->right = T2;

    updateHeight(x);

    updateHeight(y);

    return y;

}

Node\* insert(Node\* root, int data) {

    if (!root) 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);

    else

        return root;

    updateHeight(root);

    int balance = balancefactor(root);

    if (balance > 1 && data < root->left->data)

        return rightRotate(root);

    if (balance < -1 && data > root->right->data)

        return leftRotate(root);

    if (balance > 1 && data > root->left->data) {

        root->left = leftRotate(root->left);

        return rightRotate(root);

    }

    if (balance < -1 && data < root->right->data) {

        root->right = rightRotate(root->right);

        return leftRotate(root);

    }

    return root;

}

void inorder(Node\* root) {

    if (root) {

        inorder(root->left);

        cout << root->data<< " ";

        inorder(root->right);

    }

}

int main(){

    Node\* root = new Node(5);

    root = insert(root, 10);

    root = insert(root, 20);

    root = insert(root, 30);

    cout << "Inorder traversal after half insertion: ";

    inorder(root);

    cout << endl;

    root = insert(root, 40);

    root = insert(root, 50);

    root = insert(root, 25);

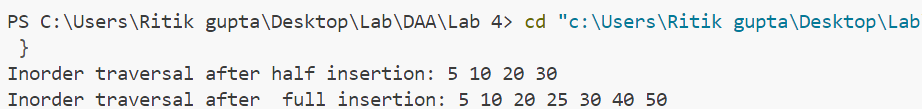
    cout << "Inorder traversal after  full insertion: ";

    inorder(root);

    cout << endl;

}

Output:



**Task -02:**

Q1-Target Pair Sum in array-

#include <bits/stdc++.h>

using namespace std;

int main(){

    int size = 6;

    vector<int>arr = {2,6,4,8,9,4};

    int target;

    cout<<"Enter target sum you want to find: ";

    cin>>target;

    sort(arr.begin() , arr.end());

    int s=0;

    int e=size-1;

    bool found = false;

    while(s<=e){

        if(arr[s] + arr[e] == target){

            found = true;

            break;

        }

        else if(arr[s] + arr[e] > target)e--;

        else s++;

    }

    if(found){

        cout<<"Target sum is present: ";

        cout<<arr[s]<<" "<<arr[e];

    }

    else{

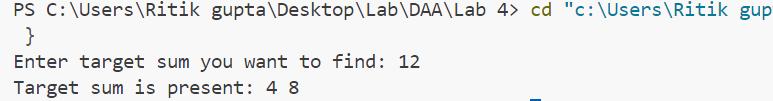
        cout<<"Target sum is not present";

    }

    return 0;

}

**Output:**

****

**Q2-Digital Root**

#include <bits/stdc++.h>

using namespace std;

    int sumDigit(int n){

        if(n==0)return 0;

        return n%10 + sumDigit(n/10);

    }

    int findRoot(int n){

        if(n<10)return n;

        return findRoot(sumDigit(n));

    }

int main(){

    int n;

    cout<<"Enter the number for digital root: ";

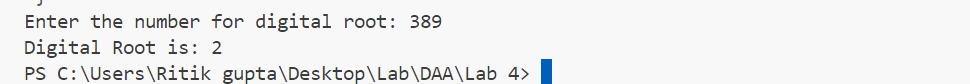
    cin>>n;

    int ans = findRoot(n);

    cout<<"Digital Root is: "<<ans;

}

**Output:**

****

**Q3. Delete Middle Element of Stack:**

#include <bits/stdc++.h>

using namespace std;

int main(){

    stack<int>st;

    stack<int> temp;

    int x;

    cout<<"Push value in stack and press -1 for exit: ";

    while(true){

        cin>>x;

        if(x==-1)break;

        st.push(x);

    }

    int size = st.size();

    for(int i =0 ;i<(size)/2;i++){

        temp.push(st.top());

        st.pop();

    }

    cout<<st.top()<<" Deleted"<<endl;

    st.pop();

    while(!temp.empty()){

        st.push(temp.top());

        temp.pop();

    }

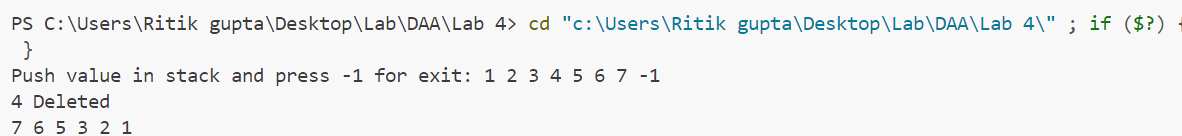
    while(!st.empty()){

        cout<<st.top()<<" ";

        st.pop();

    }}

**Output:**

****

**Q4.Next Max Height:**

#include <bits/stdc++.h>

using namespace std;

    void next\_height(vector<int>arr,vector<int>&ans){

        stack<int>st;

        for(int i =arr.size()-1; i>=0; i--){

            while(!st.empty() && st.top() <= arr[i]){

                st.pop();

            }

            if(st.empty())ans[i] = -1;

            else ans[i]=st.top();

            st.push(arr[i]);

        }

    }

int main(){

    vector<int> arr = {3,38,2,6,8,30,12};

    vector<int> ans(arr.size());

    next\_height(arr,ans);

    cout<<"Next max height is: ";

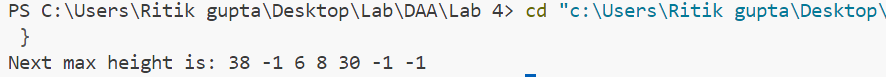
    for(int i :ans){

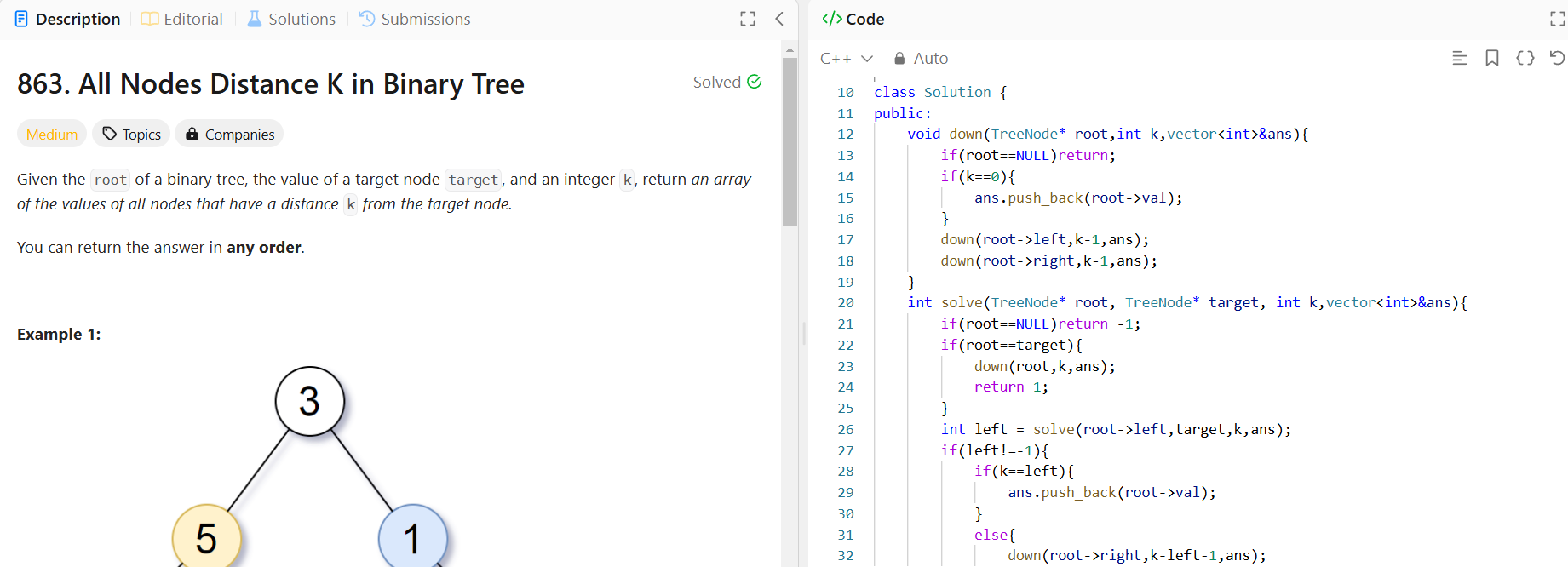
        cout<<i<<" ";

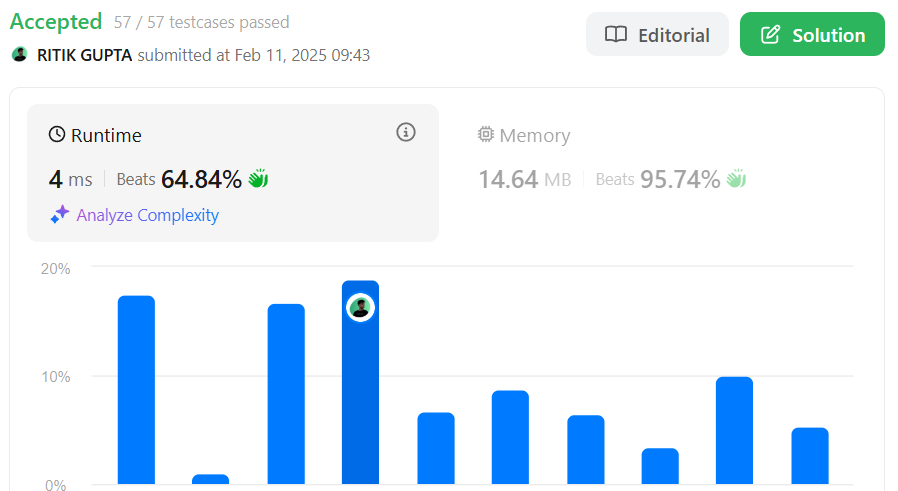
    }

}

**Output-**

****

**Q1. Print kth Distant Node**



**Q2.Maximum Matching of players**

