1. Inorder traversal of BST

vector<int> inOrder(Node \*root) {

vector<int> ans, temp;

if(root==NULL) return ans;

temp = inOrder(root->left);

ans.insert(ans.end(), temp.begin(), temp.end());

ans.push\_back(root->data);

temp = inOrder(root->right);

ans.insert(ans.end(), temp.begin(), temp.end());

return ans;

}

1. Level order traversal of BST

vector<int> levelOrder(struct Node\* node) {

vector<int> ans;

queue<Node\*> q;

q.push(node);

while(!q.empty())

{

Node \*temp = q.front();

ans.push\_back(temp->data);

if(temp->left!=NULL)

q.push(temp->left);

if(temp->right!=NULL)

q.push(temp->right);

q.pop();

}

return ans;

}

1. Insert node in BST

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

if(root==NULL)

{

Node \*temp = new Node(Key);

return temp;

}

if(root->data>Key)

root->left = insert(root->left, Key);

else if(root->data<Key)

root->right = insert(root->right, Key);

return root;

}

1. Search a node in BST

bool search(Node\* root, int x) {

if(root==NULL) return false;

if(x==root->data) return true;

else if(x>root->data)

return search(root->right, x);

else

return search(root->left, x);

}

1. Minimum element in BST

int minValue(Node\* root) {

if(root==NULL) return -1;

while(root->left!=NULL)

root = root->left;

return root->data;

}

1. Find common nodes in 2 BST

vector<int> inOrder(Node \*root) {

vector<int> ans, temp;

if(root==NULL) return ans;

temp = inOrder(root->left);

ans.insert(ans.end(), temp.begin(), temp.end());

ans.push\_back(root->data);

temp = inOrder(root->right);

ans.insert(ans.end(), temp.begin(), temp.end());

return ans;

}

vector <int> findCommon(Node \*root1, Node \*root2)

{

vector<int> v1 = inOrder(root1), v2 = inOrder(root2), ans;

int i = 0, j =0;

while(i<v1.size()&&j<v2.size())

{

if(v1[i]==v2[j])

ans.push\_back(v1[i]);

else if(v1[i]<v2[j])

j--;

else

i--;

i++;

j++;

}

return ans;

}

1. Delete node from BST

Node \* minValueNode(Node\* root)

{

Node\* current = root;

while (current->left != NULL)

current = current->left;

return current;

}

Node \*deleteNode(Node\* root, int x){

if(root==NULL)

return root;

if(root->data>x)

root->left = deleteNode(root->left, x);

else if(root->data<x)

root->right = deleteNode(root->right, x);

else

{

if(root->left==NULL)

{

return root->right;

}

else if(root->right==NULL)

{

return root->left;

}

else

{

Node \*temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

}

return root;

}

1. Lowest Common Ancestor In BST

Node\* LCA(Node \*root, int n1, int n2)

{

if (root == NULL) return NULL;

if (root->data > n1 && root->data > n2)

return LCA(root->left, n1, n2);

if (root->data < n1 && root->data < n2)

return LCA(root->right, n1, n2);

return root;

}

1. Pair sum in BST

bool isPairSum(Node \*root, int sum, unordered\_set<int> &s)

{

if(root==NULL)return false;

if(isPairSum(root->left,sum,s)==true)

return true;

if(s.find(sum-root->data)!=s.end())

return true;

else

s.insert(root->data);

return isPairSum(root->right,sum,s);

}

bool findPair(Node\* root, int X)

{

unordered\_set<int> s;

return isPairSum(root,X,s);

}

1. Smaller on right

#include<bits/stdc++.h>

using namespace std;

int smallercount(int arr[], int n)

{

set<int> s;

int m = 0;

for(int i=n-1;i>=0;i--)

{

s.insert(arr[i]);

int dist=distance(s.begin(),s.find(arr[i]));

m=max(m,dist);

}

return m;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

for(int i=0;i<n;i++)

{

cin>>arr[i];

}

int res=smallercount(arr,n);

cout<<res<<endl;

}

return 0;

}

1. Floor in bst

int floor(Node\* root, int key) {

Node \*res = NULL, \*pos=root;

while(pos!=NULL)

{

if(pos->data==key)

return key;

if(pos->data>key)

pos = pos->left;

else

{

res = pos;

pos = pos->right;

}

}

if(res==NULL) return -1;

return res->data;

}

1. Ceil in BST

int findCeil(Node\* root, int key) {

if (root == NULL) return -1;

Node \*res = NULL, \*pos=root;

while(pos!=NULL)

{

if(pos->data==key)

return key;

if(pos->data<key)

pos = pos->right;

else

{

res = pos;

pos = pos->left;

}

}

if(res==NULL) return -1;

return res->data;

}