[A - BST Operations](https://vjudge.net/problem/HackerRank-si-bst-operations" \t "_blank)

#include <iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

cout<<endl;

}

void post(){

postOrder(root);

}

void deleteNode(int data){

root=deleteBST(root,data);

}

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

if(!root) return root;

if(key < root->data){

root->left=deleteBST(root->left,key);

return root;

}

else if(key > root->data){

root->right=deleteBST(root->right,key);

return root;

}

if(root->left ==NULL){

Node\* temp=root->right;

delete root;

return temp;

}

else if(root->right == NULL){

Node\* temp=root->left;

delete root;

return temp;

}

Node\* curr=root->right;

while(curr->left != NULL)

curr=curr->left;

root->data=curr->data;

root->right=deleteBST(root->right,curr->data);

return root;

}

void searchNode(int key) {

if(searchBST(root,key)){

cout<<"Yes"<<endl;

}

else

cout<<"No"<<endl;

}

bool searchBST(Node\* root,int key){

if(root == NULL)

return false;

if(root->data == key)

return true;

else if(key > root->data)

return searchBST(root->right,key);

else

return searchBST(root->left,key);

return false;

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

};

int main()

{

int t,n,a,q,key;

cin>>t;

int i=1;

while(t--){

cout<<"Case #"<<i++<<":"<<endl;

Tree t;

cin>>q;

while(q--){

cin>>n;

if(n!=4)

cin>>key;

if(n==1){

t.insertNode(key);

}

else if(n==2)

t.deleteNode(key);

else if(n==3){

t.searchNode(key);

}

else{

t.pre();

}

}

}

return 0;

}

[B - Tree Traversals](https://vjudge.net/problem/HackerRank-si-tree-traversals)

#include <iostream>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

}

void post(){

postOrder(root);

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

};

int main()

{

int t,n,a;

cin>>t;

while(t--){

Tree t;

cin>>n;

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

cin>>a;

t.insertNode(a);

}

t.pre();

cout<<"\n";

t.in();

cout<<"\n";

t.post();

cout<<"\n\n";

}

return 0;

}

[C - Height of Tree](https://vjudge.net/problem/HackerRank-si-height-of-tree)

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

}

void post(){

postOrder(root);

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

int findHeightOfTree(Node\* root,int count,int ans){

if(root==NULL){

return count-1;

}

ans=max(findHeightOfTree(root->left,count+1,ans),findHeightOfTree(root->right,count+1,ans));

return ans;

}

int findHeight(){

return findHeightOfTree(root,0,INT\_MIN);

}

};

int main()

{

int t,n,a;

cin>>t;

while(t--){

Tree tr;

cin>>n;

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

cin>>a;

tr.insertNode(a);

}

cout<<tr.findHeight();

cout<<endl;

}

return 0;

}

[D - Depth of Tree Nodes](https://vjudge.net/problem/HackerRank-si-depth-of-tree-nodes)

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

}

void post(){

postOrder(root);

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

void findDepthOfTree(Node\* root,int count,int ans){

if(root->data == ans){

cout<<count<<" ";

}

else if(ans > root->data){

findDepthOfTree(root->right,count+1,ans);

}

else

findDepthOfTree(root->left,count+1,ans);

//return ans;

}

void findDepth(int val){

findDepthOfTree(root,0,val);

}

};

int main()

{

int t,n,a;

cin>>t;

while(t--){

Tree tr;

cin>>n;

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

cin>>a;

tr.insertNode(a);

tr.findDepth(a);

}

cout<<endl;

}

return 0;

}

[A - Level Order of Tree](https://vjudge.net/problem/HackerRank-si-level-order-of-tree)

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

}

void post(){

postOrder(root);

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

int findDepthOfTree(Node\* root,int count,int ans){

if(root->data == ans){

return count;

}

else if(ans > root->data){

return findDepthOfTree(root->right,count+1,ans);

}

return findDepthOfTree(root->left,count+1,ans);

//return ans;

}

void findDepth(int val){

findDepthOfTree(root,0,val);

}

void levelOrderBST(queue<Node\*>& q){

if(q.empty())

return;

int size=q.size();

while(size--){

cout<<q.front()->data<<" ";

if(q.front()->left != NULL)

q.push(q.front()->left);

if(q.front()->right != NULL)

q.push(q.front()->right);

q.pop();

}

cout<<"\n";

levelOrderBST(q);

}

void levelOrder(queue<Node\*>& q){

q.push(root);

levelOrderBST(q);

}

};

int main()

{

int t,n,a;

cin>>t;

while(t--){

Tree t;

cin>>n;

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

cin>>a;

t.insertNode(a);

}

queue<Node\*> q;

t.levelOrder(q);

cout<<endl;

}

return 0;

}

[B - Is Full Binary Tree](https://vjudge.net/problem/HackerRank-si-is-full-binary-tree)

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node\* left;

Node\* right;

Node(int data){

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

class Tree{

public:

Node\* root=NULL;

Node\* insert\_node\_at\_bst(Node\* root,int data){

if(root == NULL){

Node\* newNode=new Node(data);

return newNode;

}

else if(data < root->data){

root->left=insert\_node\_at\_bst(root->left,data);

}

else

root->right=insert\_node\_at\_bst(root->right,data);

return root;

}

void insertNode(int data){

root=insert\_node\_at\_bst(root,data);

}

void in(){

inOrder(root);

}

void pre(){

preOrder(root);

}

void post(){

postOrder(root);

}

void postOrder(Node\* root){

if(root == NULL)

return;

postOrder(root->left);

postOrder(root->right);

cout<<root->data<<" ";

}

void preOrder(Node\* root){

if(root == NULL)

return;

cout<<root->data<<" ";

preOrder(root->left);

preOrder(root->right);

}

void inOrder(Node\* root){

if(root == NULL)

return;

inOrder(root->left);

cout<<root->data<<" ";

inOrder(root->right);

}

int findDepthOfTree(Node\* root,int count,int ans){

if(root->data == ans){

return count;

}

else if(ans > root->data){

return findDepthOfTree(root->right,count+1,ans);

}

return findDepthOfTree(root->left,count+1,ans);

//return ans;

}

void findDepth(int val){

findDepthOfTree(root,0,val);

}

bool levelOrderBST(queue<Node\*>& q){

if(q.empty())

return true;

int size=q.size();

while(size--){

if((q.front()->left != NULL && q.front()->right == NULL) ||(q.front()->right != NULL && q.front()->left == NULL))

return false;

if(q.front()->left != NULL)

q.push(q.front()->left);

if(q.front()->right != NULL)

q.push(q.front()->right);

q.pop();

}

return levelOrderBST(q);

}

bool levelOrder(queue<Node\*>& q){

q.push(root);

return levelOrderBST(q);

}

bool isFullBST(Node\* root){

if(root==NULL)

return true;

if(!((root->left != NULL && root->right != NULL) || (root->right == NULL && root->left == NULL)))

return false;

return isFullBST(root->left) || isFullBST(root->right);

}

// bool isFullBST(Node\* root){

// if(root==NULL)

// return true;

// if((root->left != NULL && root->right != NULL) || (root->right == NULL && root->left == NULL))

// return isFullBST(root->left)|| isFullBST(root->right);

// return false;

// }

bool isFull(){

return isFullBST(root);

}

};

int main()

{

int t,n,a;

cin>>t;

while(t--){

Tree t;

cin>>n;

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

cin>>a;

t.insertNode(a);

}

queue<Node\*> q;

if(t.levelOrder(q))

cout<<"True"<<endl;

else

cout<<"False"<<endl;

}

return 0;

}

[C - Is Complete Binary Tree](https://vjudge.net/problem/HackerRank-si-is-complete-binary-tree)

//Level

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node \*left;

Node \*right;

Node(int val){

this->data = val;

this->left = NULL;

this->right = NULL;

}

};

class Tree{

public:

Node \*root=NULL;

Node \*create\_node\_and\_insert(Node \*root,int val){

if(root==NULL){

return new Node(val);

}

else if(val<root->data){

root->left=create\_node\_and\_insert(root->left,val);

}

else{

root->right=create\_node\_and\_insert(root->right,val);

}

return root;

}

void insert(int val){

root=create\_node\_and\_insert(root,val);

}

void post(){

post\_order(root);

}

void pre(){

pre\_order(root);

}

void in(){

in\_order(root);

}

void post\_order(Node \*root){

if(root == NULL){

return;

}

post\_order(root->left);

post\_order(root->right);

cout << root->data << " ";

return;

}

void in\_order(Node \*root){

if(root == NULL){

return;

}

in\_order(root->left);

cout << root->data << " ";

in\_order(root->right);

return;

}

void pre\_order(Node \*root){

if(root == NULL){

return;

}

cout << root->data << " ";

pre\_order(root->left);

pre\_order(root->right);

return;

}

void com(){

queue<Node \*> q;

isComplete(root,q);

}

void isComplete(Node \*root,queue<Node \*> q){

if(!root){

cout<<"Yes"<<endl;

return;

}

q.push(root);

while(!(q.empty())){

int qsize=q.size();

for(int i=0;i<qsize;i++){

Node \*nd=q.front();

if(nd==NULL){

while(!q.empty()){

if(q.front() == NULL)

q.pop();

else{

cout<<"No"<<endl;

return;

}

}

cout<<"Yes"<<endl;

return;

}

else{

q.push(nd->left);

q.push(nd->right);

q.pop();

}

}

}

}

};

int main()

{

int t,n,a,i;

cin>>t;

while(t--){

Tree BST;

cin>>n;

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

cin>>a;

BST.insert(a);

}

BST.com();

}

return 0;

}

[D - Is BST](https://vjudge.net/problem/HackerRank-si-is-bst)

//bst

#include <iostream>

#include <vector>

#include<bits/stdc++.h>

using namespace std;

bool isBSTUtil(const vector<int>& tree, int index, int min, int max) {

if (index >= tree.size()) {

return true;

}

int value = tree[index];

if (value <= min || value >= max) {

return false;

}

bool isLeftBST = isBSTUtil(tree, 2 \* index, min, value);

bool isRightBST = isBSTUtil(tree, 2 \* index + 1, value, max);

return isLeftBST && isRightBST;

}

bool isBST(const vector<int>& tree) {

return isBSTUtil(tree, 1, INT\_MIN, INT\_MAX);

}

int main() {

int T;

cin >> T;

while (T--) {

int N;

cin >> N;

vector<int> tree(N + 1);

for (int i = 1; i <= N; i++) {

cin >> tree[i];

}

if (isBST(tree)) {

cout << "True" << endl;

} else {

cout << "False" << endl;

}

}

return 0;

}