#include<bits/stdc++.h>

using namespace std;

class node{

public:

int data;

node\* right;

node\* left;

node(int d){

data = d;

right = NULL;

left = NULL;

}

};

class BST{

public:

bool isbst;

int min;

int max;

};

BST BstPair(node\* root){

if(root==NULL){

BST base;

base.isbst = true;

base.min = INT\_MAX;

base.max = INT\_MIN;

return base;

}

BST lf = BstPair(root->left);

BST rf = BstPair(root->right);

BST new\_node;

new\_node.isbst = lf.isbst && rf.isbst && (root->data>=lf.max && root->data<=rf.min);

new\_node.min = min(root->data,min(lf.min,rf.min));

new\_node.max = max(root->data,max(lf.max,rf.max));

return new\_node;

}

bool check(node\* root, long min, long max){

//base case

if(root==NULL) return true;

//check if curr node>min and node<max else return false

if(root->data<=min || root->data>=max) return false;

//recursively move left and update max value to root->val

// and similarly move to right and update min to root->val

return check(root->left,min,root->data) && check(root->right,root->data,max);

}

bool isValidBST(node\* root) {

return check(root,LONG\_MIN,LONG\_MAX);

}

bool check(node\* root, node\* min, node\* max){

if(root==NULL) return true;

if(min!=NULL && root->val<=min->val) return false;

if(max!=NULL && root->val>=max->val) return false;

return check(root->left,min,root) && check(root->right,root,max);

}

bool isValidBST(node\* root) {

return check(root,NULL,NULL);

}

bool isBSTUtil(struct node\* root, node \*&prev)

{

// traverse the tree in inorder fashion and

// keep track of prev node

if (root)

{

if (!isBSTUtil(root->left, prev))

return false;

// Allows only distinct valued nodes

if (prev != NULL && root->data <= prev->data)

return false;

prev = root;

return isBSTUtil(root->right, prev);

}

return true;

}

bool isBST(node \*root)

{

node \*prev = NULL;

return isBSTUtil(root, prev);

}

int main(){

//1 2 3 -1 4 -1 -1 -1 5 -1 -1

//1 2 3 -1 4 -1 -1 -1 5 6 -1 -1 7 -1 -1

//bst pair-> 4 2 1 -1 -1 3 -1 -1 6 5 -1 -1 7 -1 -1

node\* root = insertnode();

if(isValidBST(root)){

cout<<"BST"<<endl;

}

else{

cout<<"NOT BST"<<endl;

}

// BST ans = BstPair(root);

// if(ans.isbst){

// cout<<"YES"<<endl;

// }

// else{

// cout<<"NO"<<endl;

// }

}