**Largest BST**

**Medium**

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.  
**Note:**Here Size is equal to the number of nodes in the subtree.

**Example 1:**

**Input:**

  1

  / \

  4 4

  / \

  6 8

**Output:** 1

**Explanation:** There's no sub-tree with size

greater than 1 which forms a BST. All the

leaf Nodes are the BSTs with size equal

to 1.

**Example 2:**

**Input:** 6 6 3 N 2 9 3 N 8 8 2

  6

  / \

  6 3

  \ / \

  2 9 3

  \ / \

  8 8 2

**Output:** 2

**Explanation:** The following sub-tree is a

BST of size 2:

      2

    /    \

  N      8

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of the BST).

**Constraints:**  
1 ≤ Number of nodes ≤ 105  
1 ≤ Data of a node ≤ 106

**Company Tags**

[**Flipkart**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Flipkart) [**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon) [**Microsoft**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Microsoft) [**Samsung**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Samsung) [**D-E-Shaw**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=D-E-Shaw)

//{ Driver Code Starts

import java.io.\*;

import java.util.\*;

import java.math.\*;

class Node

{

int data;

Node left, right;

public Node(int d)

{

data = d;

left = right = null;

}

}

class CodingMaxima

{

static Node buildTree(String str)

{

// Corner Case

if(str.length() == 0 || str.equals('N'))

return null;

String[] s = str.split(" ");

Node root = new Node(Integer.parseInt(s[0]));

Queue <Node> q = new LinkedList<Node>();

q.add(root);

// Starting from the second element

int i = 1;

while(!q.isEmpty() && i < s.length)

{

// Get and remove the front of the queue

Node currNode = q.remove();

// Get the curr node's value from the string

String currVal = s[i];

// If the left child is not null

if(!currVal.equals("N"))

{

// Create the left child for the curr node

currNode.left = new Node(Integer.parseInt(currVal));

// Push it to the queue

q.add(currNode.left);

}

// For the right child

i++;

if(i >= s.length)

break;

currVal = s[i];

// If the right child is not null

if(!currVal.equals("N"))

{

// Create the right child for the curr node

currNode.right = new Node(Integer.parseInt(currVal));

// Push it to the queue

q.add(currNode.right);

}

i++;

}

return root;

}

public static void main(String args[]) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(br.readLine().trim());

while(t>0)

{

String s = br.readLine();

Node root = buildTree(s);

Solution T = new Solution();

System.out.println(T.largestBst(root));

t--;

}

}

}

// } Driver Code Ends

class Solution{

boolean f=true;

void array(Node root, ArrayList<Integer> ar){

if(root==null){

return ;

}

array(root.left, ar);

ar.add(root.data);

if(ar.size()>=2){

if(ar.get(ar.size()-2)>=ar.get(ar.size()-1)){

f=false;

return ;

}

}

array(root.right, ar);

}

int max=1;

void helper(Node root){

if(root==null)

return ;

ArrayList<Integer>ar=new ArrayList<>();

f=true;

array(root, ar);

if(f){

if(ar.size()>max){

max=ar.size();

}

}

helper(root.left);

helper(root.right);

}

int largestBst(Node root)

{

helper(root);

//helper(root.right);

return max;

}

}