### Complete Binary Tree

**Topic Tags**

[**Queue**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Queue) [**Tree**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Tree) [**BFS**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=BFS) [**Data Structures**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Data%20Structures) [**Algorithms**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Algorithms)

Given a Binary Tree, write a function to check whether the given Binary Tree is Complete Binary Tree or not. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes should be as much close to left as possible.

**Example 1:**

**Input:**

1

/ \

2 3

**Output:**

Complete Binary Tree

**Example 2:**

**Input:**

1

/ \

2 3

\ / \

4 5 6

**Output:**

Not Complete Binary Tree

**Constraints:**  
1<=Number of Node<=100  
0 <= Data of a node <= 106

//{ Driver Code Starts

import java.util.LinkedList;

import java.util.Queue;

import java.io.\*;

import java.util.\*;

class Node{

int data;

Node left;

Node right;

Node(int data){

this.data = data;

left=null;

right=null;

}

}

class CompleteBT{

static Node buildTree(String str){

if(str.length()==0 || str.charAt(0)=='N'){

return null;

}

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

// Create the root of the tree

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

// Push the root to the queue

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

queue.add(root);

// Starting from the second element

int i = 1;

while(queue.size()>0 && i < ip.length) {

// Get and remove the front of the queue

Node currNode = queue.peek();

queue.remove();

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

String currVal = ip[i];

// If the left child is not null

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

// Create the left child for the current node

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

// Push it to the queue

queue.add(currNode.left);

}

// For the right child

i++;

if(i >= ip.length)

break;

currVal = ip[i];

// If the right child is not null

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

// Create the right child for the current node

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

// Push it to the queue

queue.add(currNode.right);

}

i++;

}

return root;

}

static void printInorder(Node root){

if(root == null)

return;

printInorder(root.left);

System.out.print(root.data+" ");

printInorder(root.right);

}

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

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

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

while(t-->0){

String s = br.readLine();

Node root = buildTree(s);

CodingMaxima g=new CodingMaxima ();

boolean a=g.isCompleteBT(root);

if(a==true)

System.out.println("Complete Binary Tree");

else

System.out.println("Not Complete Binary Tree");

}

}

}

// } Driver Code Ends

class CodingMaxima

{

boolean isCompleteBT(Node root)

{

if(root==null)

return true;

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

q.add(root);

// if(root.left==null || root.right==null)

// return true;

boolean flag=false;

while(!q.isEmpty()){

Node temp=q.poll();

if(flag && (temp.left!=null || temp.right!=null))

return false;

if(temp.left==null &&temp.right!=null)

return false;

if(temp.left!=null)

q.add(temp.left);

if(temp.right!=null)

q.add(temp.right);

else

flag=true;

}

return true;

}

}