**Top View of Binary Tree**

**Medium**

Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. For the given below tree

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

Top view will be: 4 2 1 3 7  
**Note:**Return nodes from **leftmost**node to **rightmost**node. Also if 2 nodes are outside the shadow of the tree and are at same position then consider the extreme ones only(i.e. leftmost and rightmost).   
For ex - **1 2 3 N 4 5 N 6 N 7 N 8 N 9 N N N N N** will give **8 2 1 3** as answer. Here 8 and 9 are on the same position but 9 will get shadowed.

**Example 1:**

**Input:**

  1

 /    \

2      3

**Output:** 2 1 3

**Example 2:**

**Input:**

  10

   /      \

20        30

/   \    /    \

40   60 90    100

**Output:** 40 20 10 30 100

**Expected Time Complexity:**O(NlogN)  
**Expected Auxiliary Space:**O(N).

**Constraints:**  
1 ≤ N ≤ 105  
1 ≤ Node Data ≤ 105

**Company Tags**

[**Paytm**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Paytm) [**Ola Cabs**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Ola%20Cabs) [**Walmart**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Walmart)

//{ Driver Code Starts

//Initial Template for JAVA

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;

}

}

public class Tree {

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);

Solution ob = new Solution();

ArrayList<Integer> vec = ob.topView(root);

for(int x : vec)

System.out.print(x + " ");

System.out.println();

t--;

}

}

}

// } Driver Code Ends

//User function Template for Java

class Solution

{

static ArrayList<Integer> topView(Node root)

{

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

if (root == null) {

return arr;

}

Queue<Pair> q=new ArrayDeque<>();

Map<Integer, Integer> hm = new TreeMap<>();

q.add(new Pair(0,root));

while(!q.isEmpty()){

Pair current=q.poll();

if(!hm.containsKey(current.hd))

hm.put(current.hd, current.node.data);

if(current.node.left!=null)

q.add(new Pair(current.hd-1,current.node.left));

if(current.node.right!=null)

q.add(new Pair(current.hd+1,current.node.right));

}

for(Map.Entry<Integer, Integer> entry:hm.entrySet()){

arr.add(entry.getValue());

}

return arr;

}

static class Pair{

int hd;

Node node;

public Pair(int hd, Node node){

this.hd=hd;

this.node=node;

}

}

}