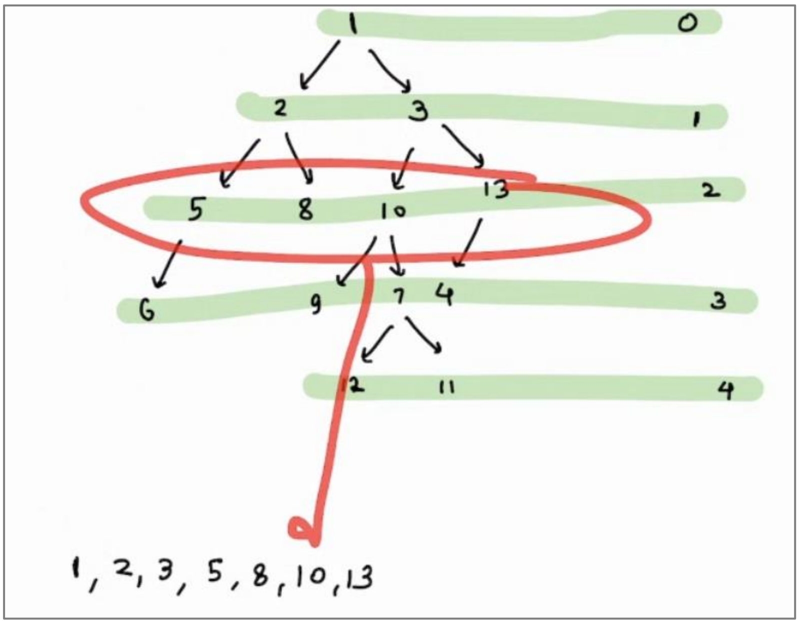
Trees 2: Views & Types

Level Order Traversal



All nodes from the level 0 to last level

Level order traversal

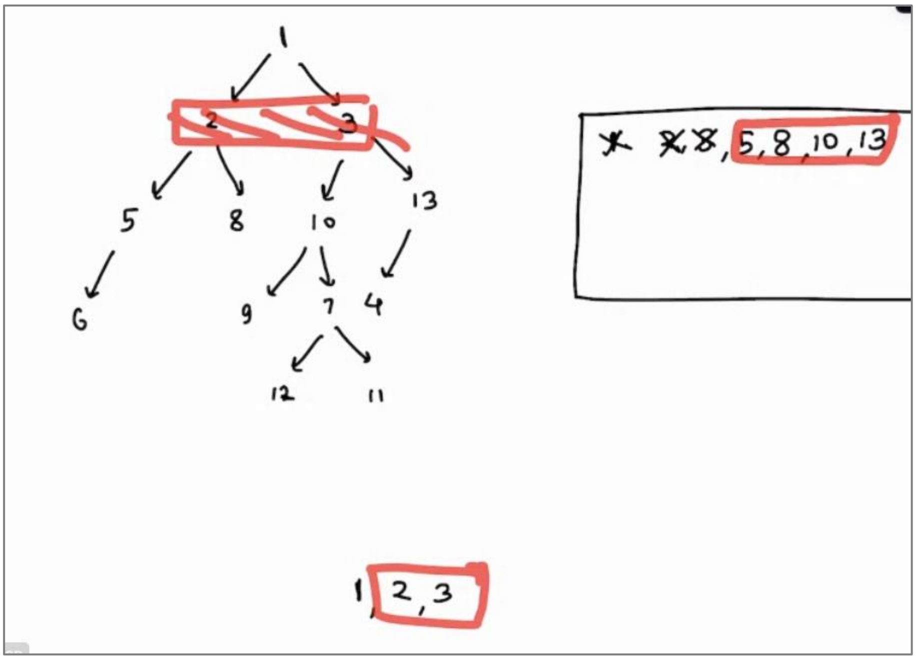
1, 2, 3, 5, 8, 10, 13, 6, 9, 7, 4, 12, 11

print each level from top to bottom. [left to right by default for each level]

[Inorder
pre → Depth First
post traversal]

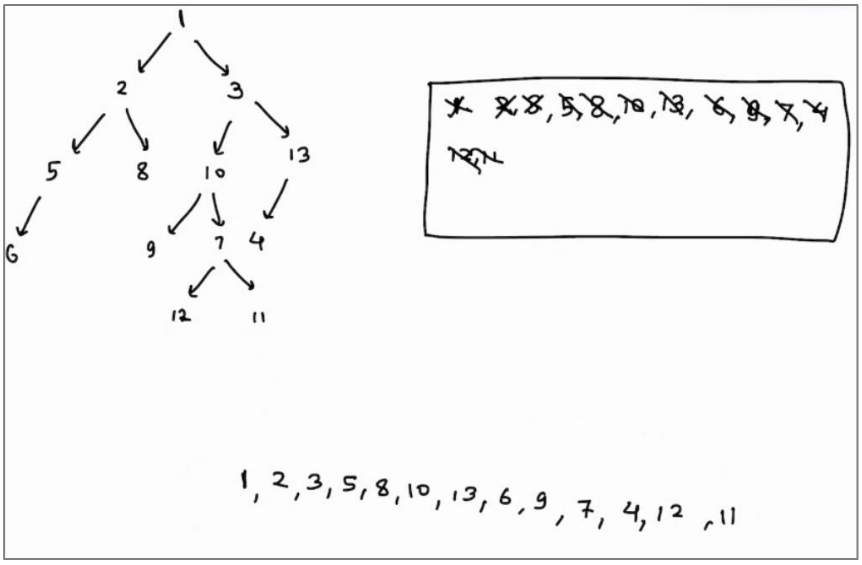
[level order → Breadth First traversal]

Breadth means Level here. Data in Level order traversal is printed level wise starting form root node



print root node.--> delete 1 → add child of 1 in data structure.

delete 2 → print 2 –> add child of 2



The favourable data structure is queue since we want deletion from head and insertion from tail → so here we will use queue

```
Queue < Node > q

q.push (head)

while ( q.Isempty() == False)
{
        Node tmp = q.front()
        q.dequeue()

        print ( tmp.data)

        If ( tmp.left != NULL)   q.push ( tmp.left)
        If ( tmp.right != NULL)  q.push ( tmp.right)
}

return
```

TC = O(N)

In worst case, 2 complete levels will be added in the queue so **SC = O(N)**

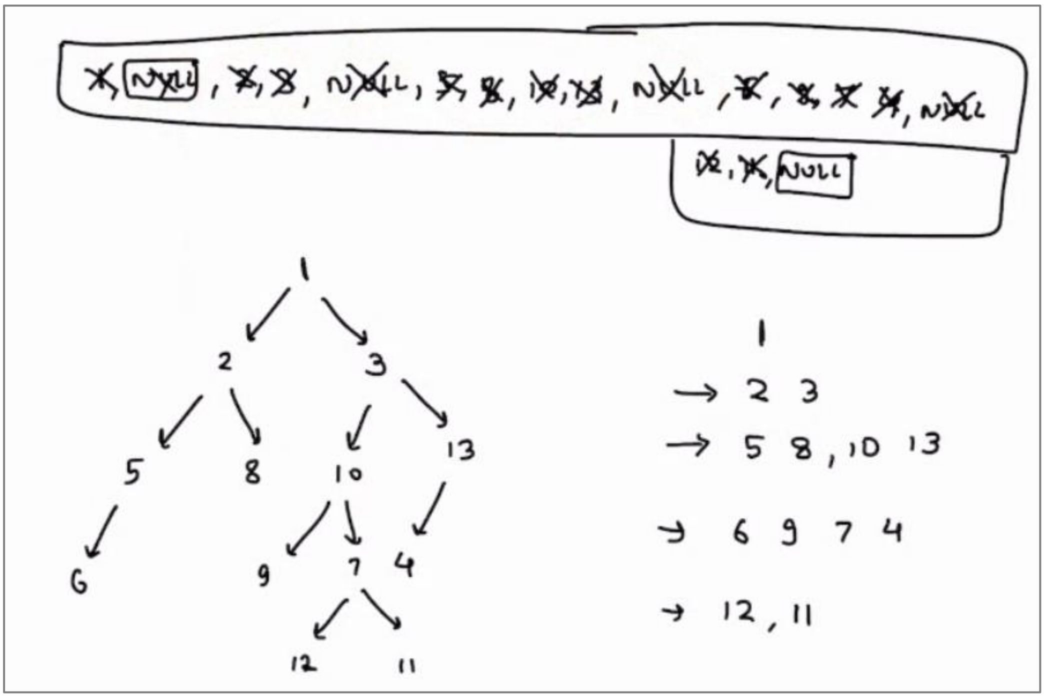
Print level wise in a new line

```
Queue < Node > q

q.push (head)

while ( q.Isempty() == False)
{
        int  cnt = q.size()

        for( i=1; i ≤ cnt ; i++)
        {
            Node tmp = q.front()
            q.dequeue()

            print ( tmp.data)

            If ( tmp.left != NULL)   q.push ( tmp.left)
            If ( tmp.right != NULL)  q.push ( tmp.right)
        }

        println ( )
}
```

TC = O(N)

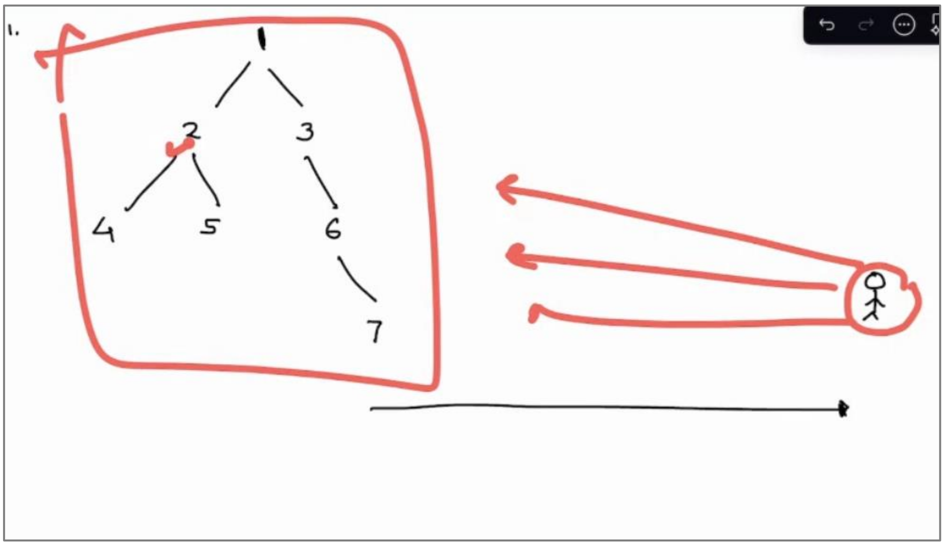
SC = O(N)

or you can add null after every level ends



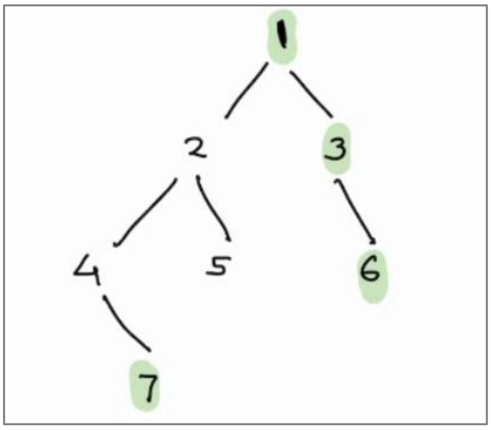
Problem Statement 1-

Suppose there is a building, and the right view of the building will be whatever a person can see form the right side of the building. SO in a tree, a person will see last node of each level.

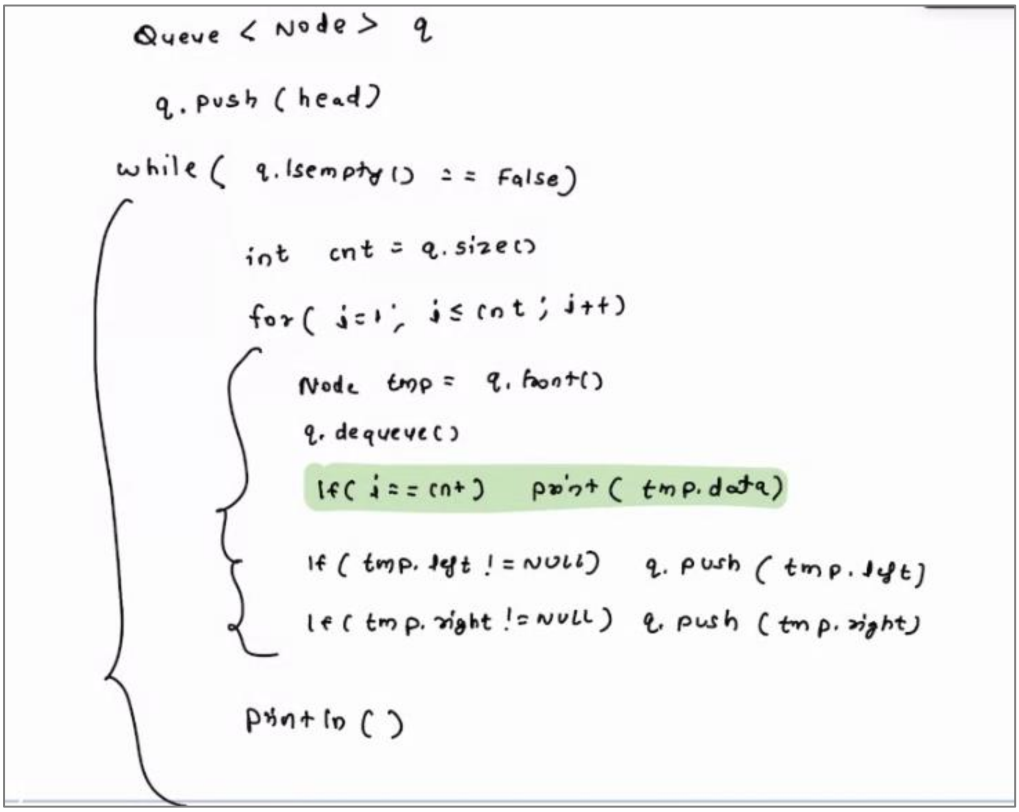


Right view ⇒ [1, 3, 6, 7]

Even for the tree below my right would be [1, 3, 6, 7]



Code for Right View

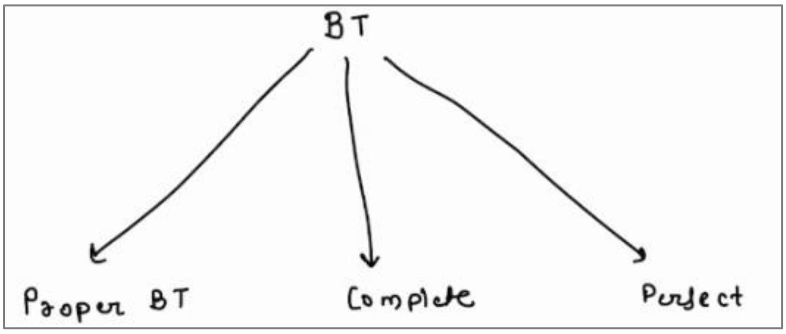


```
Queue < Node > q
  q.push ( head)
while ( q.Isempty() == False)
{
      int  cnt = q.size()
      for( i=1; i ≤ cnt ; i++)
      {
          Node tmp = q.front()
          q.dequeue()
          If( i == cnt)   print( tmp.data)

          If ( tmp.left != NULL)   q.push ( tmp.left)
          If ( tmp.right != NULL)  q.push ( tmp.right)
      }
      println ( )
}
```

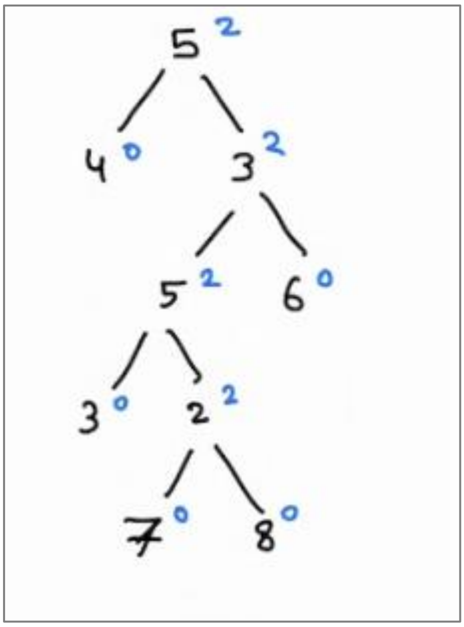
Code for Left View

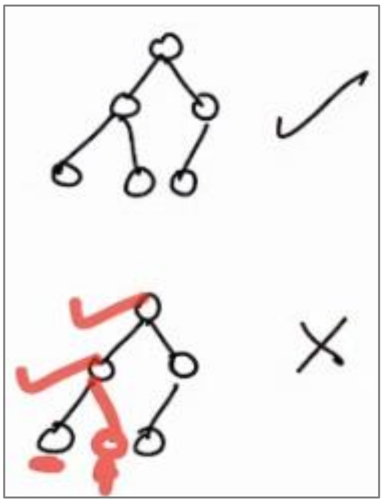
```
Queue < Node > q

q.push ( head)                                  TC: O(N)
                                                SC: O(N)
while( q.isempty() == False)
{
        int cnt = q.size()
        for( i=1; i<=cnt; i++)
        {
            Node tmp = q.front()
            q.dequeue()
            if( i == 1)    print( tmp.data)

            if ( tmp.left != NULL)    q.push ( tmp.left)
            if ( tmp.right != NULL)   q.push ( tmp.right)
        }

        println ( )
}
```

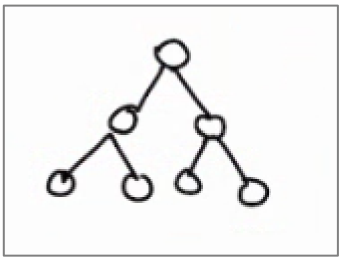
Types of Binary Tree



Proper BT	Complete	Perfect
Each nodes having 0 or 2 Child	All level completely filled Except possibly the last level [left to right]	Each level completely filled







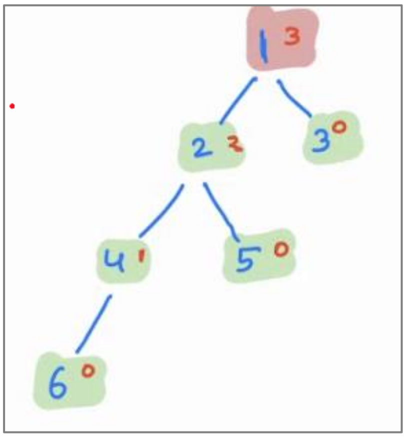
in perfect BT, only last levels have leaf nodes

Perfect will be complete & proper

Problem Statement 2-

Given a binary tree, check if it is height balanced or not?

Height balanced: For all nodes, abs (height of left child – height of right child) <=1



For 3,6,5 no child node → 0

For node 2, height of 4 – height of 5 =1

For node 1, height of 2 – height of 3 = 2

This tree is not balanced

I will check height for my left child and right child and then my root node will calculate for left and right child

Brute Force

```
bool IsheightBalanced ( Node root )
{
    leftheight = height ( root.left)
    right ''   =   ''      '' .right)

    if ( abs ( leftheight - rightheight) > 1)
                  return False

    return  IsheightBalanced ( root.left)
                     &&
            IsheightBalanced ( root.right)
}
```

```
int   height ( root )
{
    return  1 + max [ height ( root.left)
                      height ( root.right) ]
}
```

Tc : $O(N^2)$

Height function will require N time and heightbalanced would require N times → so N^2 times total

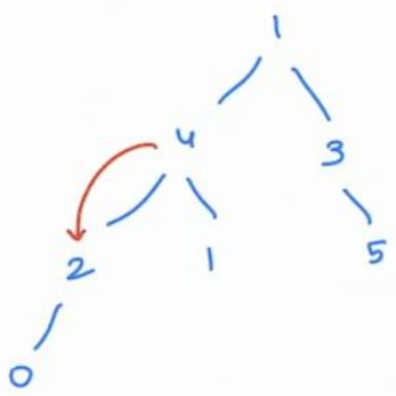
```
int height( root , bool ans )
{
    if (root == NULL) return -1

    l = height( root.left)
    r = height( root.right)
    if ( abs(l-r) > 1) ans = False

    return 1 + max(l, r)
}
```

TC: O(N)
SC: O(N)

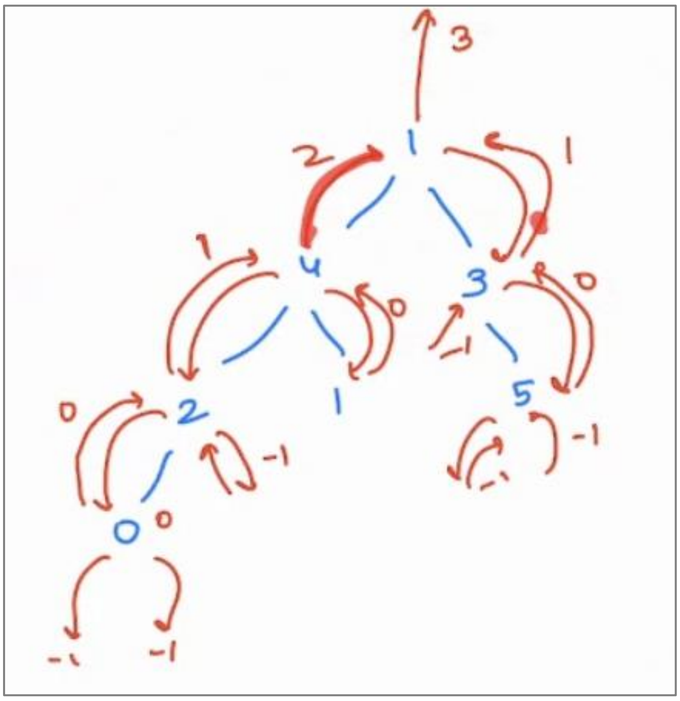
```
boolean main()
{
    ans = True

    height( root, &ans)

    return ans
}
```

Since we are not updating my ans to true in my height function, here once ans is false, it will be false. In height function, we are returning height only →we will not make any changes there

height(1) = height(4)
height(3)



Height(1) will require height of 4 &3. Height 4 will require height of 2 &1 and so on.



recursive calls. Before returning height, we must check whether height of l – r <=1

Problem 1- Balanced Binary Tree

Problem Description
Given a root of binary tree **A**, determine if it is height-balanced.
A height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.
Problem Constraints
1 <= size of tree <= 100000
Input Format
First and only argument is the root of the tree A.
Output Format
Return 0 / 1 (0 for false, 1 for true) for this problem.
Example Input
Input 1:

```
  1
 / \
2   3
```

Input 2:

```
     1
    /
   2
  /
 3
```

Example Output

Output 1:

1

Output 2:

0

Example Explanation

Explanation 1:

It is a complete binary tree.

Explanation 2:

Because for the root node, left subtree has depth 2 and right subtree has depth 0.

Difference = 2 > 1.

Actual Code

```
public class Solution {
  public int isBalanced(TreeNode A) {

    return (height(A) != -1)?1:0;
  }
  private int height(TreeNode A){
    if(A ==null) return 0;

    int l = height(A.left);
    if(l == -1) return -1;

    int r = height(A.right);
    if (r == -1) return -1;

    if(Math.abs(l-r)>1) return -1;

    return 1+Math.max(l,r);
  }
}
```

Problem Statement – 2 Level Order

Problem Description

Given a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Problem Constraints

1 <= number of nodes <= 10^5

Input Format

First and only argument is root node of the binary tree, A.

Output Format

Return a 2D integer array denoting the level order traversal of the given binary tree.

Example Input

Input 1:

```
    3
   / \
  9  20
    /  \
   15   7
```
Input 2:
```
   1
  / \
 6   2
    /
   3
```
Example Output

Output 1:
```
 [
   [3],
   [9, 20],
   [15, 7]
 ]
```
Output 2:
```
 [
   [1]
   [6, 2]
   [3]
 ]
```
Example Explanation

Explanation 1:

Return the 2D array. Each row denotes the traversal of each level.

Actual Code

```
public class Solution {
   public ArrayList<ArrayList<Integer>> solve(TreeNode A) {
      ArrayList<ArrayList<Integer>> res = new ArrayList<>();
      if(A==null) return res;
      Queue<TreeNode> q = new LinkedList<>();
      q.add(A);

      while(!q.isEmpty()){
         int cnt = q.size();
         ArrayList<Integer> currLevel = new ArrayList<>();

         for(int i =0; i<cnt; i++){
            TreeNode currNode = q.poll();
            currLevel.add(currNode.val);

            if(currNode.left!=null){
```

```
                q.add(currNode.left);
            }
            if(currNode.right!=null){
                q.add(currNode.right);
            }
        }
        res.add(currLevel);
    }
    return res;
  }
}
```

Problem Statement 3- Binary Tree From Inorder And Postorder

Problem Description

Given the inorder and postorder traversal of a tree, construct the binary tree.

NOTE: You may assume that duplicates do not exist in the tree.

Problem Constraints

1 <= number of nodes <= 10^5

Input Format

First argument is an integer array A denoting the inorder traversal of the tree.

Second argument is an integer array B denoting the postorder traversal of the tree.

Output Format

Return the root node of the binary tree.

Example Input

Input 1:

A = [2, 1, 3]

B = [2, 3, 1]

Input 2:

A = [6, 1, 3, 2]

B = [6, 3, 2, 1]

Example Output

Output 1:

```
  1
 / \
2   3
```

Output 2:

```
  1
 / \
6   2
   /
  3
```

Example Explanation

Explanation 1:

Create the binary tree and return the root node of the tree.

Actual Code

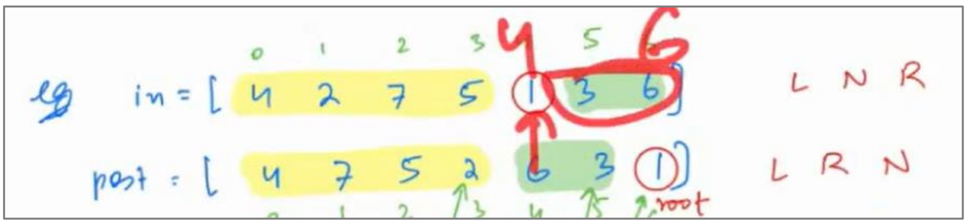
```
public class Solution {
   public TreeNode buildTree(ArrayList<Integer> A, ArrayList<Integer> B) {
      if(A == null || B ==null || A.size() != B.size()){
         return null;
      }
      HashMap<Integer, Integer> map = new HashMap<>();
      for(int i =0; i<A.size(); i++){
         map.put(A.get(i),i);
      }

      return build(A, B, 0, A.size()-1, B.size()-1, map);
   }
   private TreeNode build(ArrayList<Integer>A, ArrayList<Integer>B, int inL, int inR, int postR,
HashMap<Integer, Integer>map){

      if(inL > inR) return null;
      TreeNode root = new TreeNode(B.get(postR));

      int idx = map.get(B.get(postR));  // *idx is the index of root node in inorder array
      int cntR = inR - idx;
      root.left = build(A, B, inL, idx-1, postR-cntR-1, map);
      root.right = build(A, B, idx+1, inR, postR-1, map);
      return root;
   }
}
```

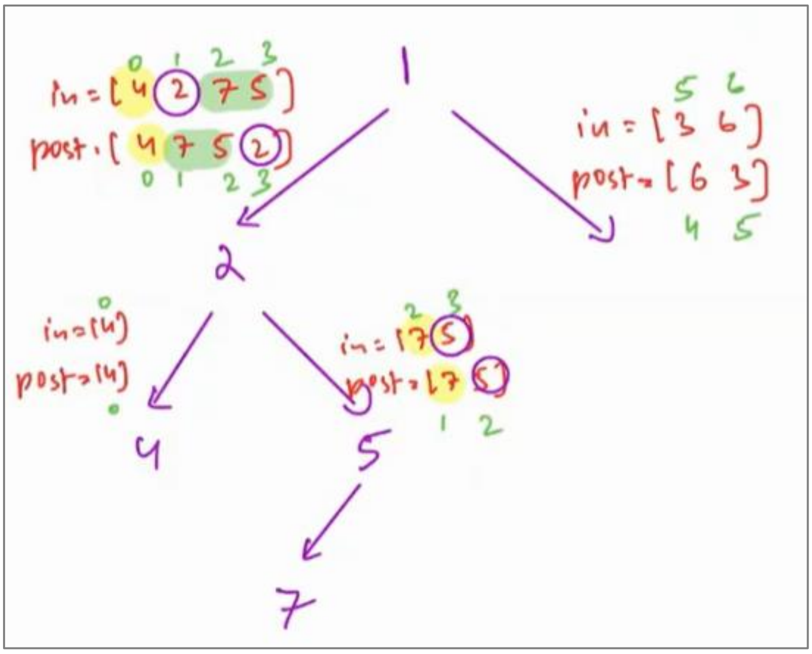
//* idx is the index of root node in inorder array which is 4. So from B.get(postR) we get of root node element which is 1 and search in Hashmap <1,4> where we get index of root node = 4 corresponding to search of root node element =1.

//*cntR = inR-idx
if idx of root node in inorder array = 4 and inR(right array index of inorder array) = 6 → 6-4 =2. There are two elements in the right array

//root.left = build(A, B, inL, idx-1, postR-cntR-1, map);
For left subarray → inL =0, inR = idx -1 (index of root node -1) which is 3
and postR = postR-cntR-1 = from postorder array. 6 -2 -1 = 3 so would end index of right side

// root.right = build(A, B, idx+1, inR, postR-1, map);
inL = idx +1 (in right subarray using inorder array) inL will be 5
inR = 6
postR = postR-1 since in postorder array the last element will be root node/parent node. So one element before root node is the index of right subarray





Problem Statement 4 - Left View of Binary tree

Problem Description

Given a binary tree of integers. Return an array of integers representing the left view of the Binary tree.

Left view of a Binary Tree is a set of nodes visible when the tree is visited from Left side

NOTE: The value comes first in the array which have lower level.

Problem Constraints

1 <= Number of nodes in binary tree <= 100000

0 <= node values <= 10^9

Input Format

First and only argument is a root node of the binary tree, A.

Output Format

Return an integer array denoting the left view of the Binary tree.

Example Input

Input 1:

```
      1
    /  \
   2    3
  / \  / \
 4   5 6  7
/
8
```

Input 2:

```
      1
     / \
    2   3
     \
      4
```

\
5

Example Output

Output 1:

[1, 2, 4, 8]

Output 2:

[1, 2, 4, 5]

Example Explanation

Explanation 1:

The Left view of the binary tree is returned.

```
public class Solution {
    public ArrayList<Integer> solve(TreeNode A) {
        Queue<TreeNode> q = new LinkedList<>();
        ArrayList<Integer> res = new ArrayList<>();
        q.add(A);
        while(!q.isEmpty()){
            int cnt = q.size();
            for(int i =1; i<=cnt; i++){
                TreeNode tmp = q.poll();
                if(i ==1){
                    res.add(tmp.val);
                }
                if(tmp.left != null) q.add(tmp.left);
                if(tmp.right !=null) q.add(tmp.right);

            }
        }
        return res;

    }
}
```

Problem Statement 5 - Binary Tree From Inorder And Preorder

Given preorder and inorder traversal of a tree, construct the binary tree.

NOTE: You may assume that duplicates do not exist in the tree.

Problem Constraints

1 <= number of nodes <= 10^5

Input Format

First argument is an integer array A denoting the preorder traversal of the tree.

Second argument is an integer array B denoting the inorder traversal of the tree.

Output Format

Return the root node of the binary tree.

Example Input

Input 1:

A = [1, 2, 3]
B = [2, 1, 3]

Input 2:

A = [1, 6, 2, 3]
B = [6, 1, 3, 2]

Example Output

Output 1:

```
  1
 / \
2   3
```

Output 2:

```
  1
 / \
6   2
   /
  3
```

Example Explanation

Explanation 1:

Create the binary tree and return the root node of the tree.

Actual Code

```
public class Solution {
    public TreeNode buildTree(ArrayList<Integer> A, ArrayList<Integer> B) {
        if(A == null || B == null || A.size() != B.size()){
            return null;
        }

        HashMap <Integer, Integer> map = new HashMap<>();
        for(int i =0; i<B.size(); i++){
            map.put(B.get(i), i);
        }
        return build(B, A, 0, B.size()-1, 0, map);
    }
    private TreeNode build(ArrayList<Integer> B, ArrayList<Integer> A, int inL, int inR, int preL,
HashMap<Integer, Integer> map){

        if(inL > inR) return null;
        TreeNode root = new TreeNode(A.get(preL));
        int idx = map.get(A.get(preL));
        int cntL= idx-inL;

        root.left = build(B, A, inL, idx-1, preL+1, map);
        root.right = build(B, A, idx+1, inR, preL +cntL+1, map);
        return root;
    }
}
```

Problem Statement 6 - Serialize Binary Tree

Given the **root node** of a Binary Tree denoted by **A**. You have to Serialize the given Binary Tree in the described format.

Serialize means encode it into a **integer array** denoting the **Level Order Traversal** of the given Binary Tree.

NOTE:

? In the array, the NULL/None child is denoted by -1.

? For more clarification check the Example Input.

Problem Constraints

1 <= number of nodes <= 10^5

Input Format

Only argument is a A denoting the root node of a Binary Tree.

Output Format

Return an integer array denoting the Level Order Traversal of the given Binary Tree.

Example Input

Input 1:

```
      1
     /  \
    2    3
   / \
  4   5
```

Input 2:

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

Example Output

Output 1:

[1, 2, 3, 4, 5, -1, -1, -1, -1, -1, -1]

Output 2:

[1, 2, 3, 4, 5, -1, 6, -1, -1, -1, -1, -1, -1]

Example Explanation

Explanation 1:

The Level Order Traversal of the given tree will be [1, 2, 3, 4, 5 , -1, -1, -1, -1, -1, -1].

Since 3, 4 and 5 each has both NULL child we had represented that using -1.

Explanation 2:

The Level Order Traversal of the given tree will be [1, 2, 3, 4, 5, -1, 6, -1, -1, -1, -1, -1, -1].

Since 3 has left child as NULL while 4 and 5 each has both NULL child.

Actual Code

```
public class Solution {
  public ArrayList<Integer> solve(TreeNode A) {
  Queue<TreeNode> q = new LinkedList<>();
```

```
  ArrayList<Integer> res = new ArrayList<>();
  if(A== null){
    res.add(-1);
    return res;
  }
  q.add(A);
  while(!q.isEmpty()){
    TreeNode curr = q.poll();
    if(curr == null){
      res.add(-1);
    }else{
      res.add(curr.val);
      q.add(curr.left);
      q.add(curr.right);
    }
  }
  return res;
  }
}
```

Problem Statement 7 - Deserialize Binary Tree

You are given an integer array **A** denoting the **Level Order Traversal** of the Binary Tree.
You have to Deserialize the given Traversal in the Binary Tree and return the **root** of the Binary Tree.

NOTE:

- In the array, the NULL/None child is denoted by -1.
- For more clarification check the Example Input.

Problem Constraints

1 <= number of nodes <= 10^5

-1 <= A[i] <= 10^5

Input Format

Only argument is an integer array A denoting the Level Order Traversal of the Binary Tree.

Output Format

Return the root node of the Binary Tree.

Example Input

Input 1:

A = [1, 2, 3, 4, 5, -1, -1, -1, -1, -1, -1]

Input 2:

A = [1, 2, 3, 4, 5, -1, 6, -1, -1, -1, -1, -1, -1]

Example Output

Output 1:

```
   1
  / \
 2   3
```

```
   / \
  4   5
```
Output 2:
```
      1
     /  \
    2    3
   / \.  \
  4  5.  6
```
Example Explanation

Explanation 1:

Each element of the array denotes the value of the node. If the val is -1 then it is the NULL/None child.

Since 3, 4 and 5 each has both NULL child we had represented that using -1.

Explanation 2:

Each element of the array denotes the value of the node. If the val is -1 then it is the NULL/None child.

Since 3 has left child as NULL while 4 and 5 each has both NULL child.

Actual Code

```
public class Solution {
   public TreeNode solve(ArrayList<Integer> A) {
      Queue<TreeNode> q = new LinkedList<>();
      TreeNode tmp = new TreeNode(A.get(0));
      q.add(tmp);

      int i =1;
      while(i<A.size() && !q.isEmpty()){
         TreeNode curr = q.poll();

         int left_value = A.get(i);
         if(left_value != -1){
            curr.left = new TreeNode(left_value);
            q.add(curr.left);
         }
         i++;

         if(i < A.size()){
            int right_value = A.get(i);

            if(right_value != -1){
               curr.right = new TreeNode(right_value);
               q.add(curr.right);
            }
         }
```

```
            i++;
        }
        return tmp;
    }
}
```

Problem Statement 7 - Right View of Binary tree

Given a binary tree of integers denoted by root **A**. Return an array of integers representing the right view of the Binary tree.

Right view of a Binary Tree is a set of nodes visible when the tree is visited from Right side.

Problem Constraints

1 <= Number of nodes in binary tree <= 100000

0 <= node values <= 10^9

Input Format

First and only argument is head of the binary tree A.

Output Format

Return an array, representing the right view of the binary tree.

Example Input

Input 1:

```
        1
      /  \
     2    3
    / \  / \
   4   5 6  7
  /
 8
```

Input 2:

```
        1
       / \
      2   3
       \
        4
         \
          5
```

Example Output

Output 1:

[1, 3, 7, 8]

Output 2:

[1, 3, 4, 5]

Example Explanation

Explanation 1:

Right view is described.

Explanation 2:

Right view is described.

Actual Code

```
public class Solution {
    public ArrayList<Integer> solve(TreeNode A) {
        Queue<TreeNode> q = new LinkedList<>();
        ArrayList<Integer> res = new ArrayList<>();

        q.add(A);

        while(!q.isEmpty()){
            int cnt = q.size();
            for(int i = 1; i<=cnt; i++){
                TreeNode tmp = q.poll();
                if(i == cnt) res.add(tmp.val);

                if(tmp.left != null) q.add(tmp.left);
                if(tmp.right != null) q.add(tmp.right);
            }
        }
        return res;
    }
}
```