Trees 2

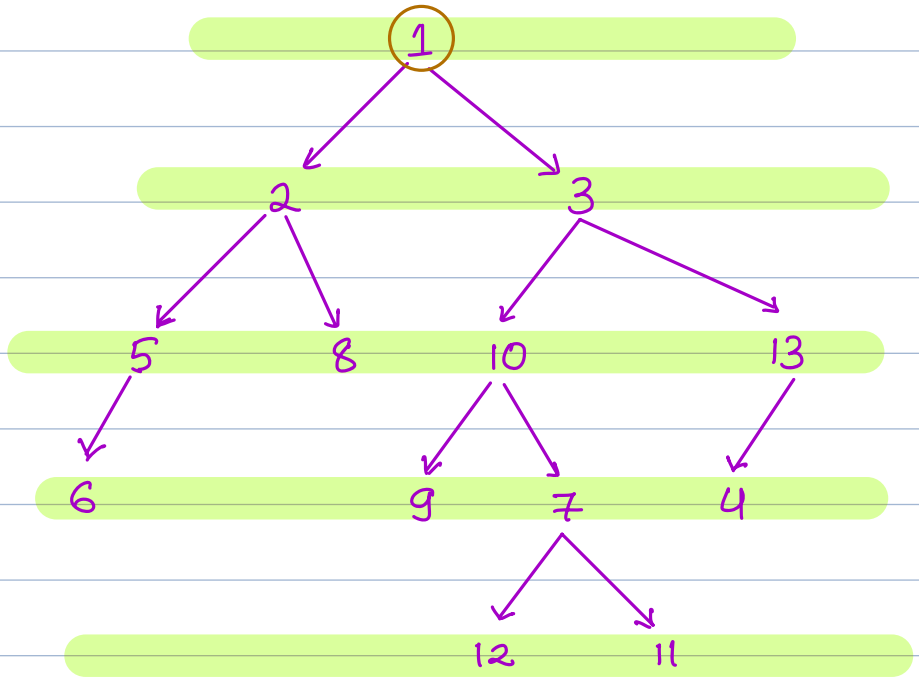
content

- Level Order Traversal
- Left view & Right view
- Vertical Order Traversal
- Top view & bottom view
- Types of Binary tree
- check if a binary tree is height balanced or not

Level Order Traversal

① 2 3 5 8 10 13 6 9 7 4 12 11

Levels in a Tree



Levels

L0

L1

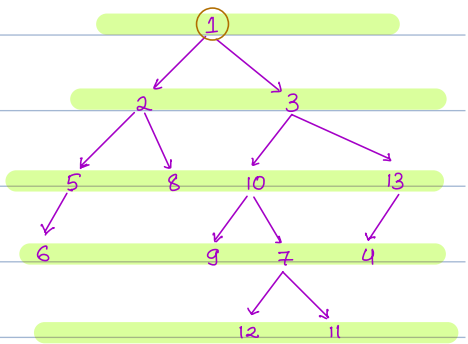
L2

L3

L4

To traverse level by level use queue

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

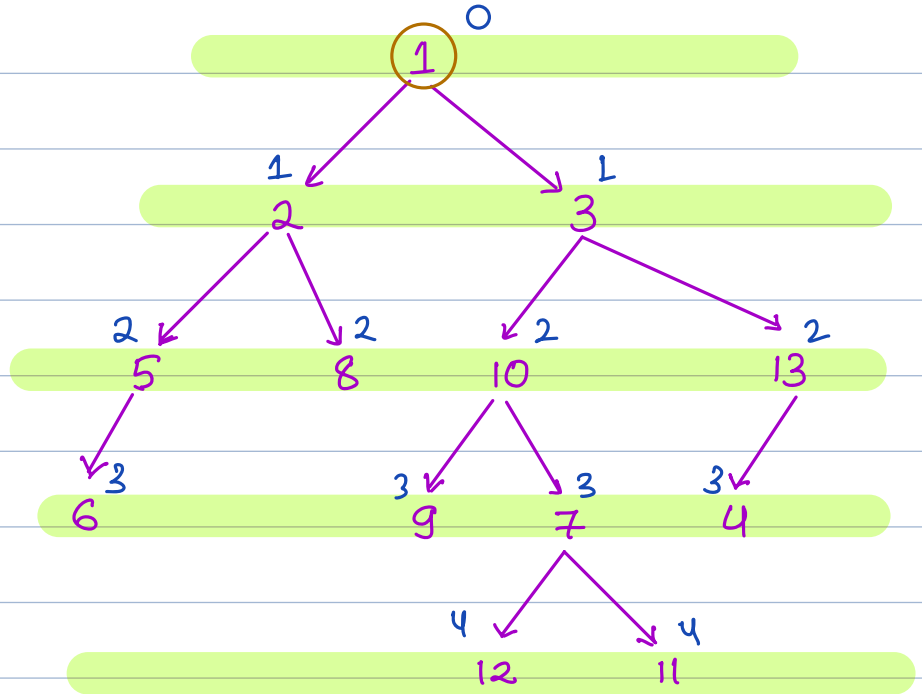



```
queue.enqueue(root)
while(!queue.isempty()) {
    node = queue.dequeue() // remove front.
    print(node.val)
    if(node.left != null) queue.enqueue(node.left)
    if(node.right != null) queue.enqueue(node.right)
}
```

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

Print all the levels in a separate line.

Levels in a Tree



Levels

L0

L1

L2

L3

L4

Output :

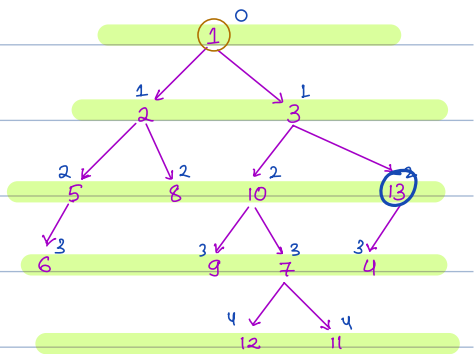
1

2 3

5 8 10 13

6 9 7 4

12 11



(1, 0)

(2, 1) (3, 1)

(5, 2) (8, 2) (10, 2) (13, 2)

(6, 3) (9, 3) (7, 3) (4, 3)

(12, 4) (11, 4)

```
class Pair {
    Node node
    int level
}
```

① Pair class

② Make two queue one for node & one for level

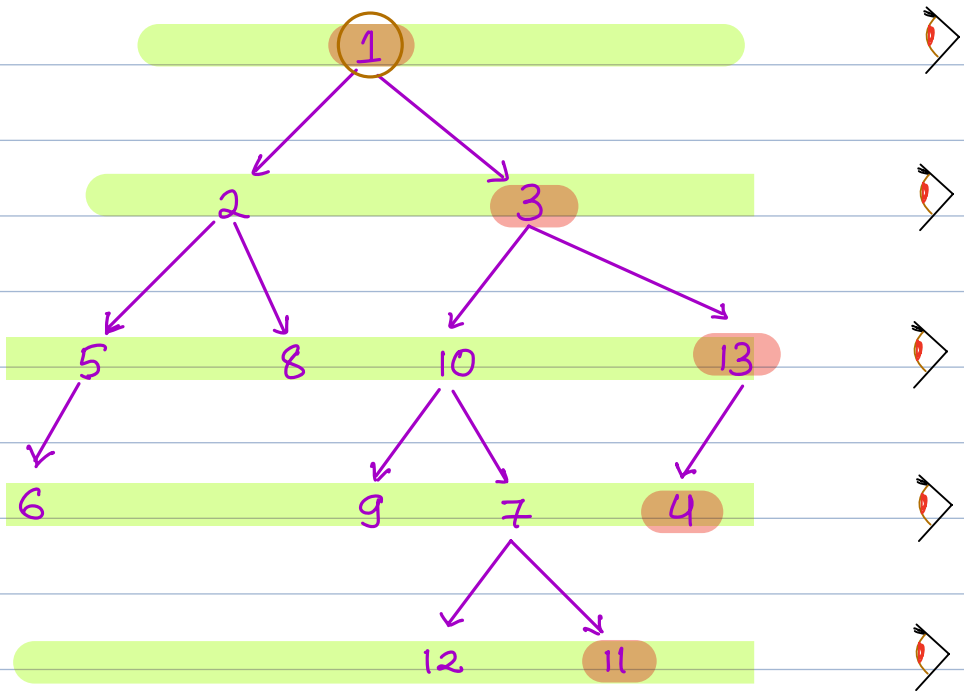
```
prelevel = 0
queue.enqueue(new Pair(root, 0))
while(!queue.isempty()) {
    x = queue.dequeue() // remove front.
    node = x.node
    level = x.level
    if (prelevel != level) {
        prelevel = level
        print("\n")
    }
    print(node.val)

    if (node.left != null) {
        queue.enqueue(new Pair(node.left, level+1))
    }
    if (node.right != null) {
        queue.enqueue(new Pair(node.right, level+1))
    }
}
```

Only print new line if levels are different.

TC: O(N)

SC: O(N)

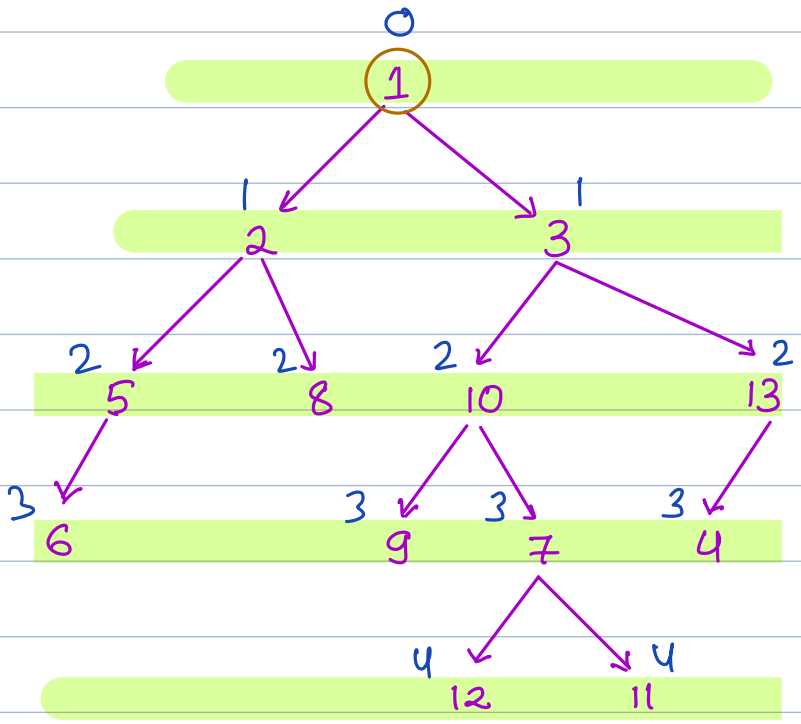
Print right view of binary tree.



Only print the rightmost nodes of every level.

Idea 1 Use the prev idea and also maintain a prev value

pnode = 11
plevel = 4



1, 3, 13, 4, 11 → pnode print after while loop

```
// Always handle this case
if (root == null) return -1

pnode = new Node(-1)
prelevel = 0
queue.enqueue(new Pair(root, 0))
while(!queue.isempty()) {
    x = queue.dequeue() // remove front.
    node = x.node
    level = x.level
    if (prelevel != level) {
        prelevel = level
        print(pnode.val)
    }
    pnode = node

    if (node.left != null) {
        queue.enqueue(new Pair(node.left, level+1))
    }
    if (node.right != null) {
        queue.enqueue(new Pair(node.right, level+1))
    }
}
print(pnode.val)
```

TC: O(N)

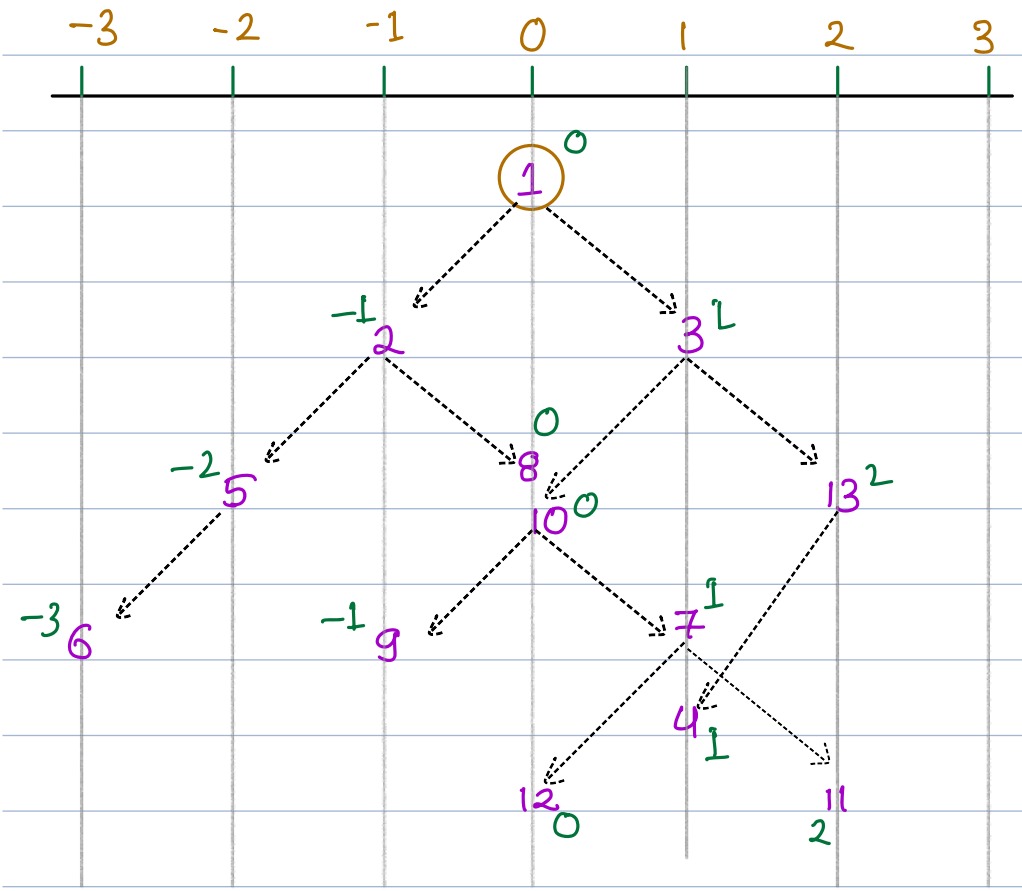
SC: O(N)

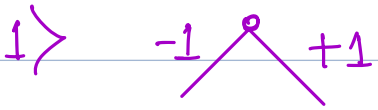
HW Print the left view of binary tree

Break 9:56

Vertical Order Traversal

{ store the ans in dynamic Array }



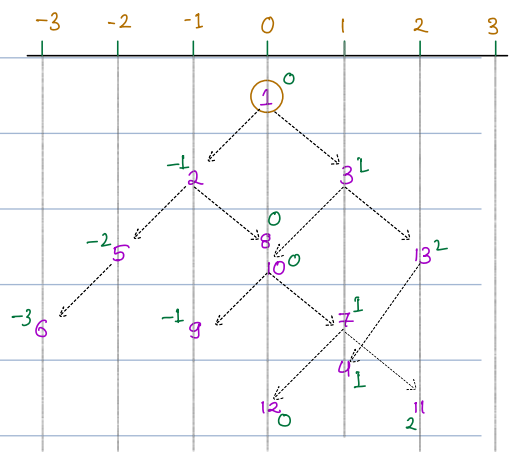
1) 

2) Print from top to bottom

3) Overlap : Give preference to left

-3	6			
-2	5			
-1	2	9		
0	1	8	10	12
1	3	7	4	
2	13	11		

- store (Node, vlevel)
- Maintain a hashmap
  key : vlevel
  value : Arraylist



-3	(6)			
-2	(5)			
-1	(2)	(9)		
0	(1)	(8)	(10)	(12)
1	(3)	(7)	(4)	
2	(13)	(11)		

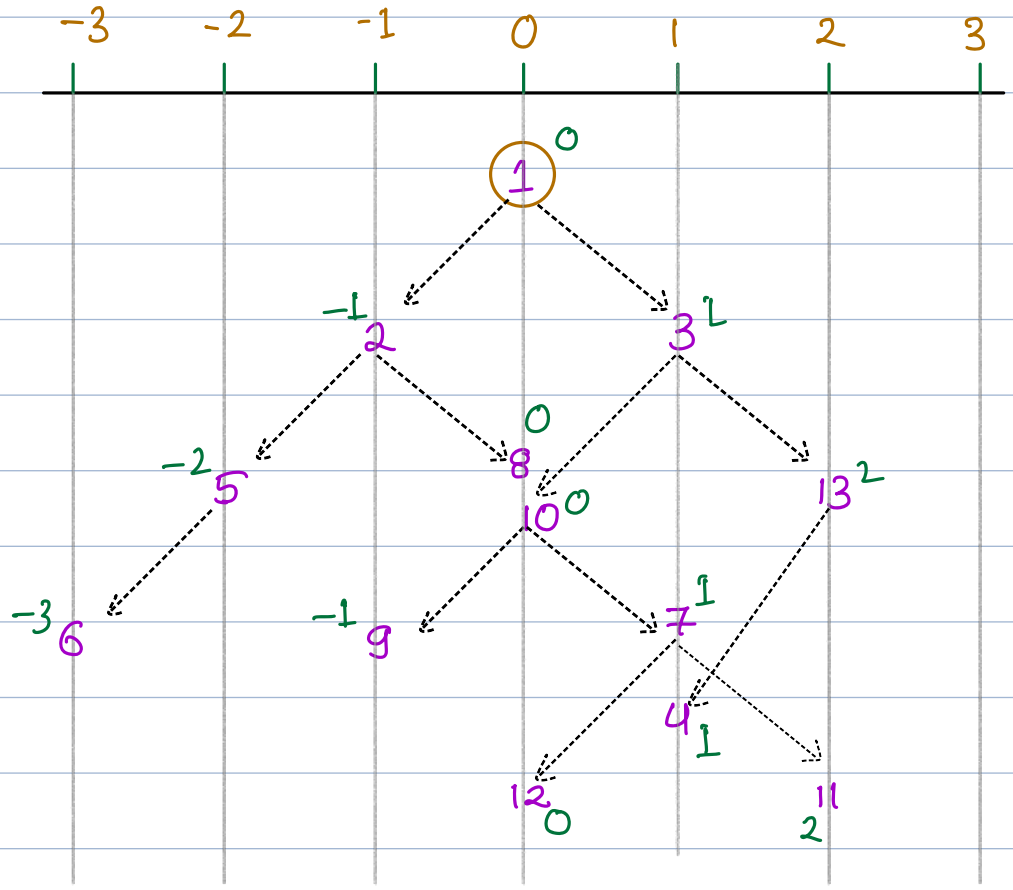
Pseudocode HW read more about getOrDefault

minV maxV

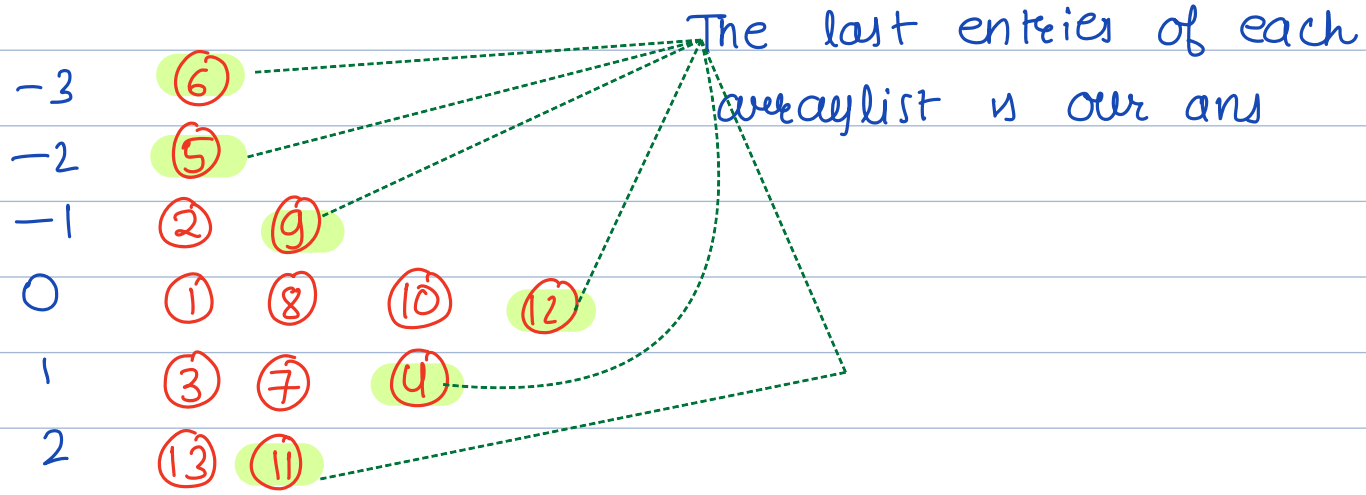
```
Map<Integer, ArrayList> vlevels = new HashMap<>()
queue.enqueue(new Pair(root, 0))
while(!queue.isempty()) {
    x = queue.dequeue() // remove front.
    node = x.node
    vlevel = x.level
    lst = vlevels.getOrDefault(vlevel, new ArrayList)
    lst.add(node.val)
    vlevels.put(vlevel, lst)
    minV = min(minV, vlevel)
    maxV = max(maxV, vlevel)

    if(node.left != null) {
        queue.enqueue(new Pair(node.left, vlevel - 1))
    }

    if(node.right != null) {
        queue.enqueue(new Pair(node.right, vlevel + 1))
    }
}

ArrayList< ArrayList<Integer>> ans = new Arrayli...

for(vlevel = minV; vlevel <= maxV; vlevel++) {
    lst = vlevels.getOrDefault(vlevel, new ArrayList)
    ans.add(lst)
}
```

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

Bottom view



6 5 9 12 4 2



HW — Do the same for top view

Types of Binary Tree

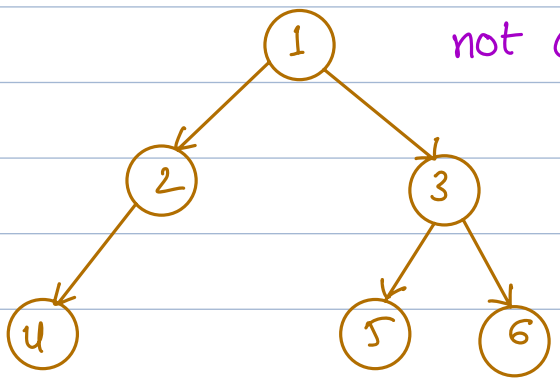
1> **Proper binary tree**

Every node has either 0 or 2 children

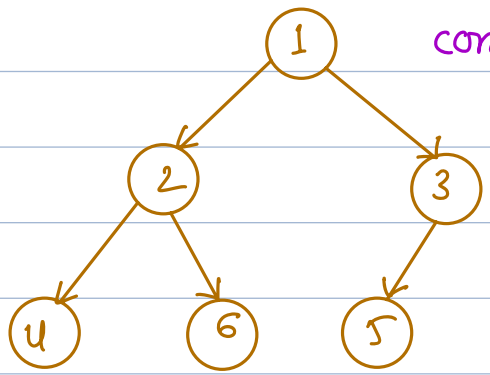
2> **Complete Binary Tree**

All levels filled, and last level maybe filled.

left to right filling



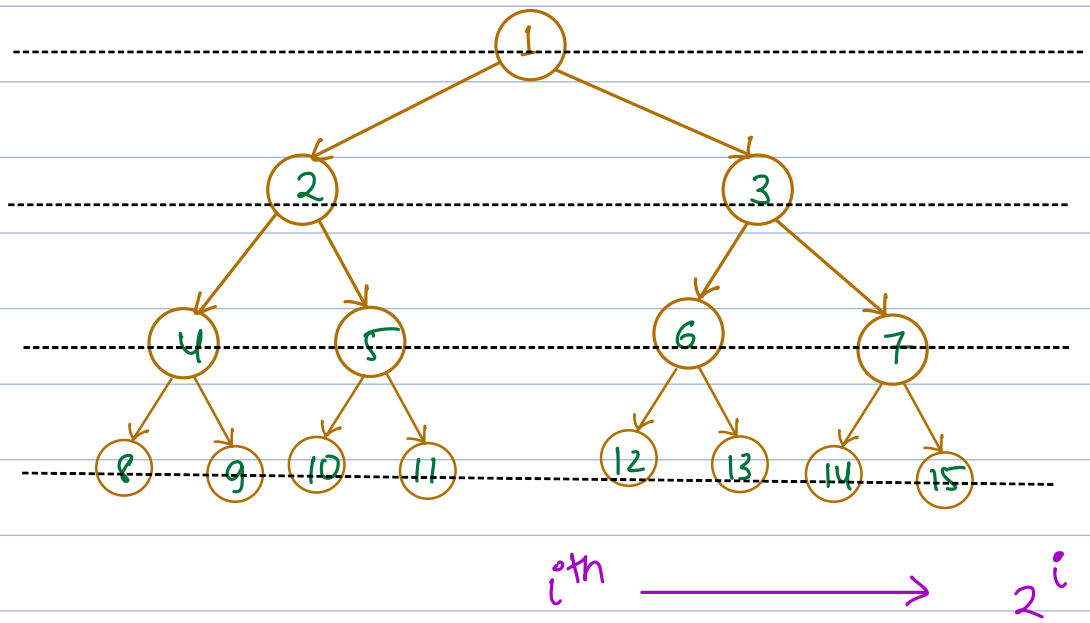
not complete



complete.

3> **Perfect binary tree**

All the levels are completely filled.



what is the height of perfect binary tree with N nodes ?

$$N = 2^0 + 2^1 + 2^2 + \ldots\ldots 2^H$$

$$N = \frac{2^{H+1} - 1}{2 - 1}$$

$$N = 2^{H+1} - 1$$

$$2^{H+1} = N + 1$$

$$H + 1 = \log(N+1)$$

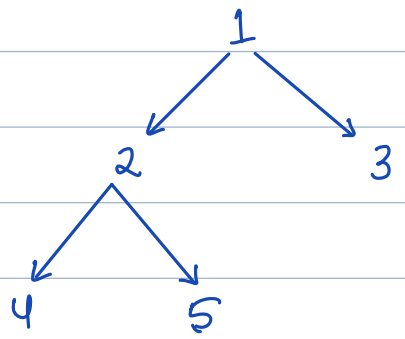
$$H = \log(N+1) - 1$$

Check if the given tree is height balanced ?

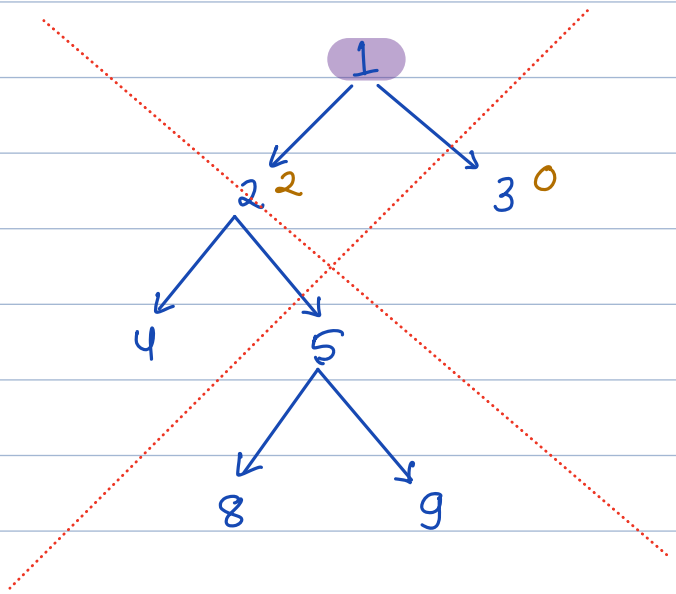
$\forall$ nodes

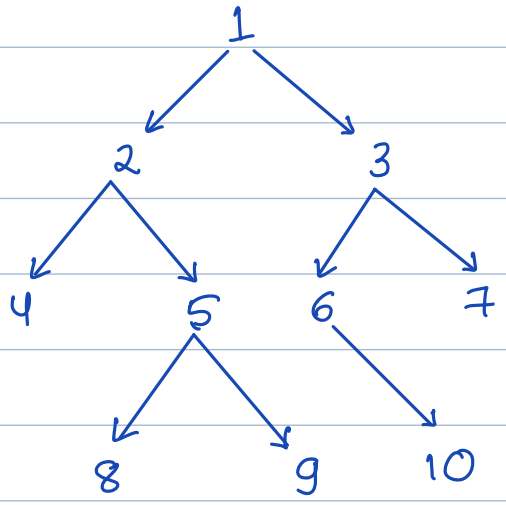
height of left child	—	height of right child	<= 1

Eg :

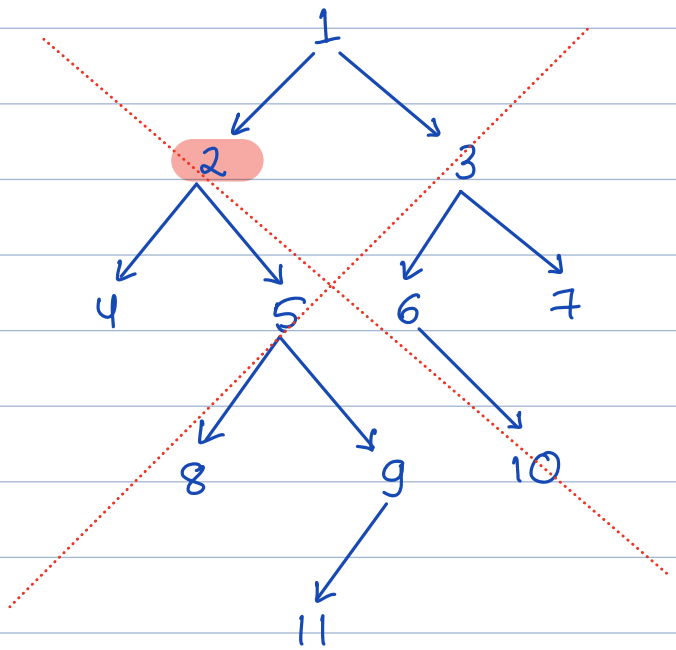


Height balanced

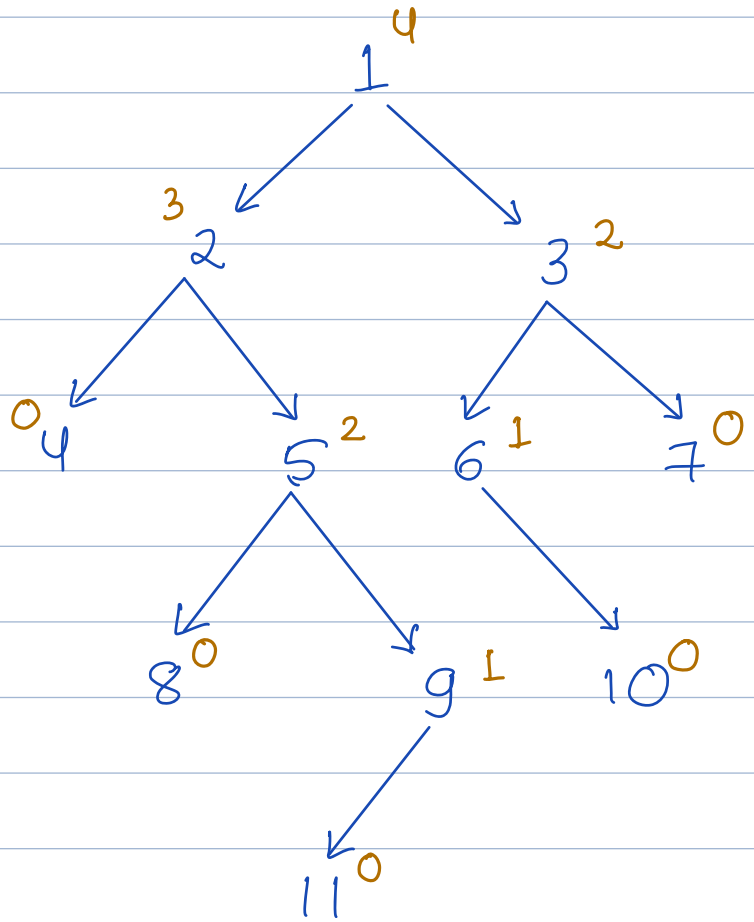




Height balanced



Traversal



Pseudo code

postorder

```
isBalanced = true

int height (root) {
    if (root == null) {
        return -1
    }

    lh = height(root.left)
    rh = height(root.right)
    if (abs(lh - rh) > 1) {
        isBalanced = false
    }
    h = max(lh, rh) + 1
    return h
}
```

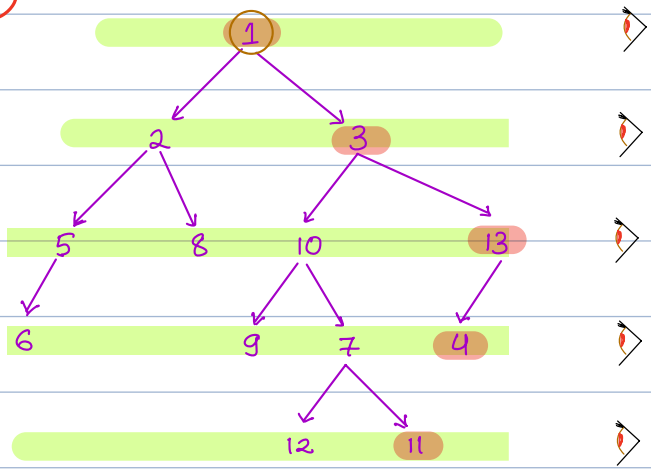
TC : O(N)

SC : O(H) ⟶ A better parameter

Dry Run

plevel = 4

pnode (11)



1 3 13 4