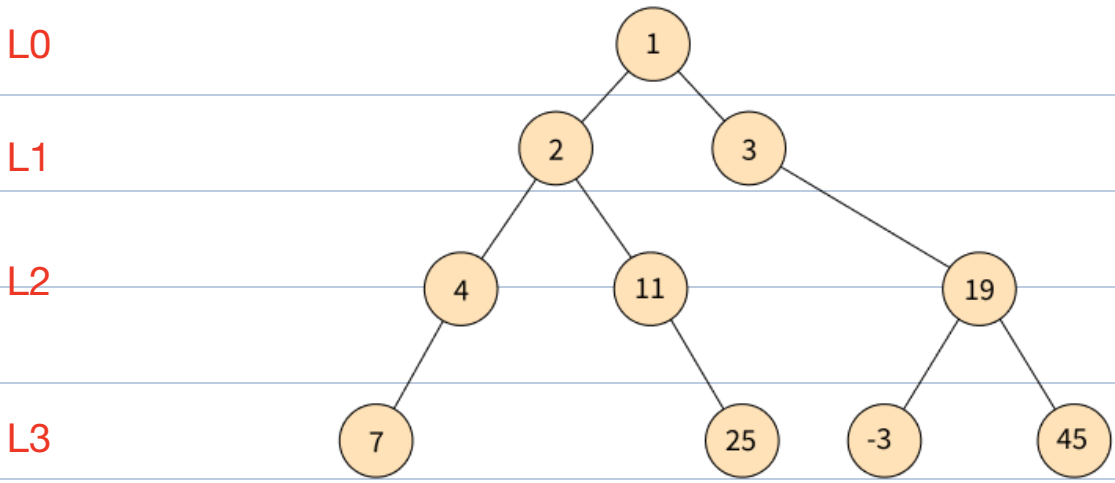
# Trees 2

TABLE OF CONTENTS

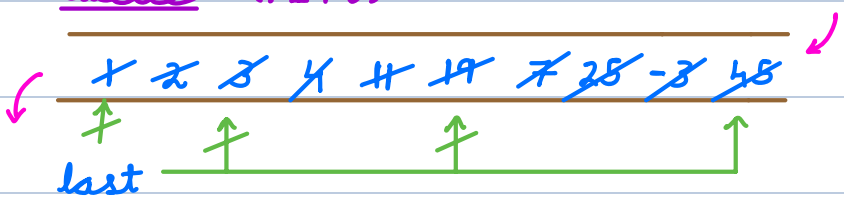


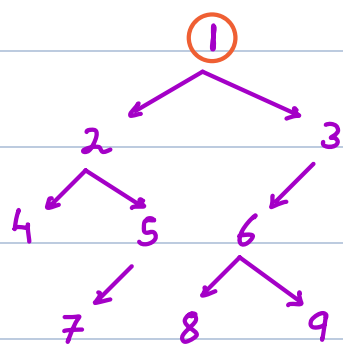
# Level Order Traversal




o/p → 1
2 3
4 11 19
7 25 -3 45

Queue (FIFO)





last = 1 3 6 9

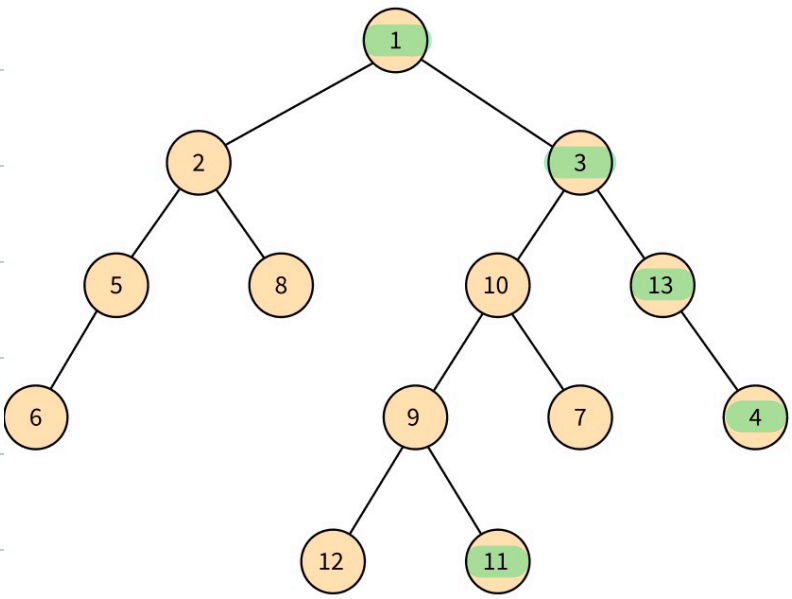
x = 1 2 3 4 5 6 7 8 9

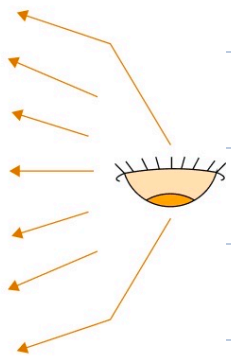
o/p → 1
2 3
4 5 6
7 8 9

```
q.enqueue(root)
last = root
while (!q.isEmpty()) {
    x = q.dequeue()
    print(x.val)
    if (x.left != null)  q.enqueue(x.left)
    if (x.right != null) q.enqueue(x.right)
    if (x == last && !q.isEmpty()) {
        print("\n")
        last = q.rear()
    }
}
```

TC = O(N)  SC = O(N)

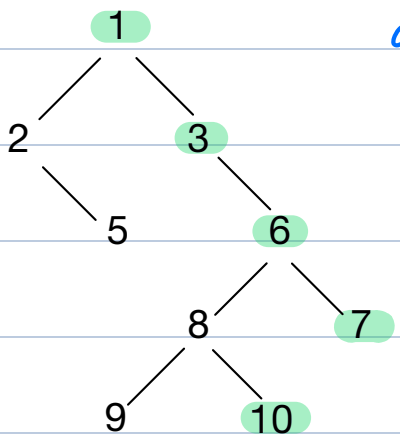
**< Question > :** Find right view of binary tree.



o/p → 1 3 13 4 11

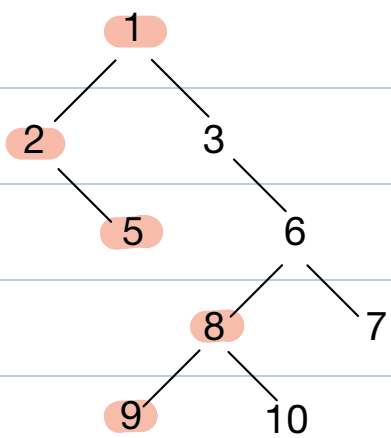
**Quiz :**




o/p → 1 3 6 7 10

Sol → Print last node of the level. ✓

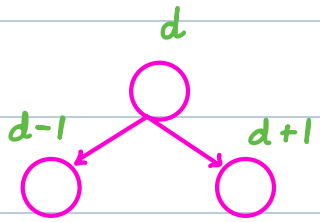
**< Question > :** Find left view of binary tree.




o/p → 1 2 5 8 9

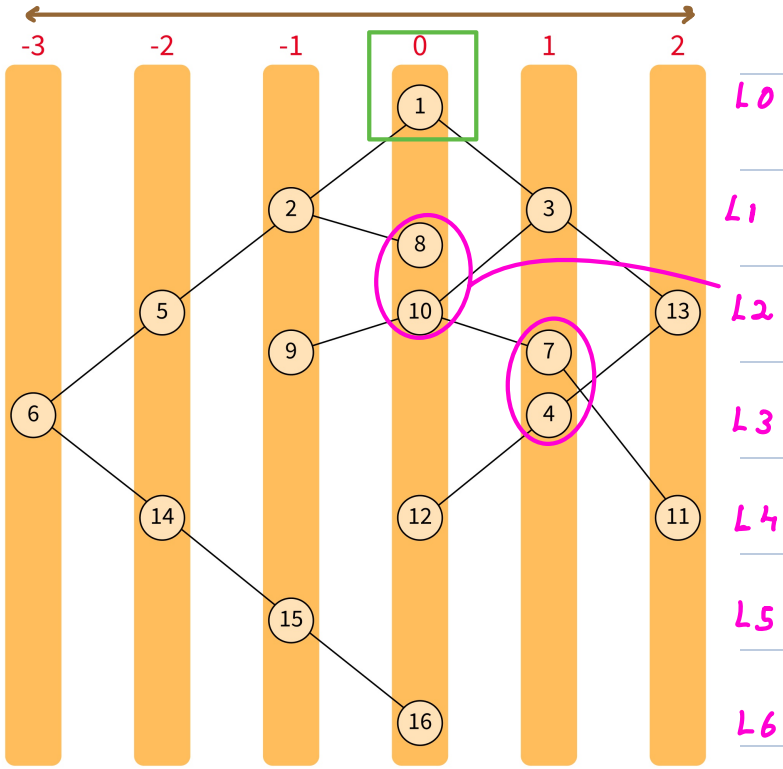
H.W ✓

# Vertical Order Traversal of Binary Tree




Print each vertical line top to bottom.

If there is overlap print the node coming from left side first.




o/p → 6

5 14

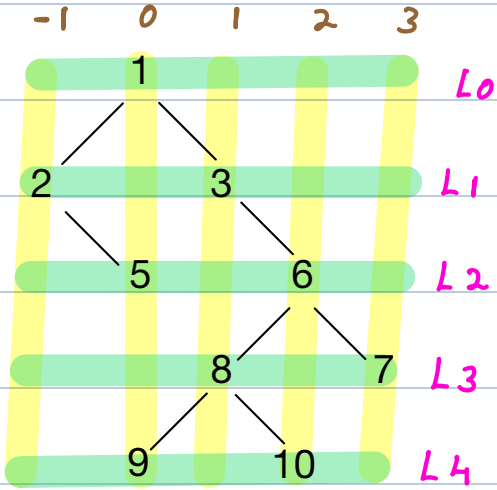
2 9 15

1 8 10 12 16

3 7 4

13 11

**Quiz :**




o/p → 2

1 5 9

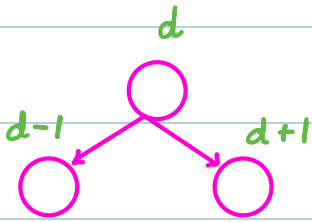
3 8

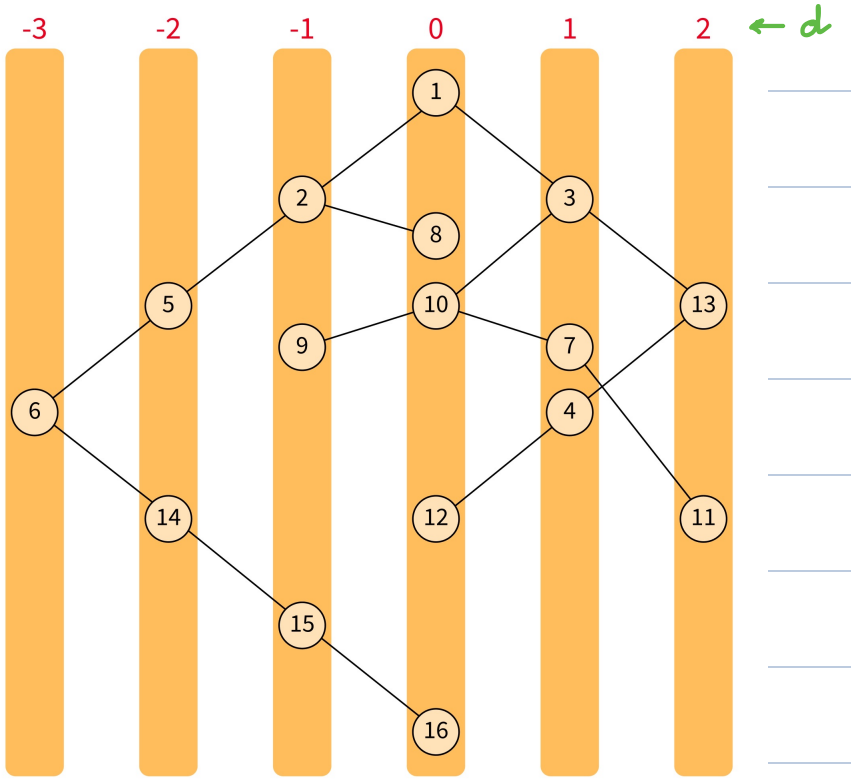
6 10

7

Print top to bottom

⇒ Level order Traversal




∀ nodes, store distance

⇒ Pair < Node, d > /

∀ i, d[i] → distance of 'i'. ✓

∀ level, store list of nodes ⇒ HashMap < Level, list of Nodes >

---

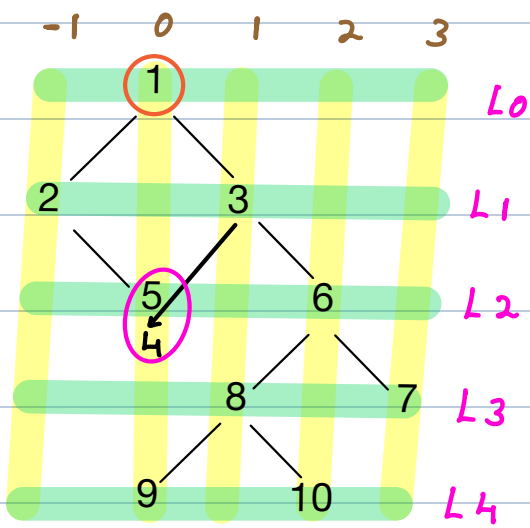
Queue

~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~4~~ ~~6~~ ~~8~~ ~~7~~ ~~9~~ ~~10~~

d =

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -1 | 1 | 0 | 0 | 2 | 3 | 1 | 0 | 2 |

Nodes → 1 to N




HashMap < dist, List of Nodes >

0 → {1, 5, 4, 9}

-1 → {2}

1 → {3, 8}

2 → {6, 10}

3 → {7}

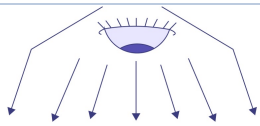
min_d → min horizontal distance // -1

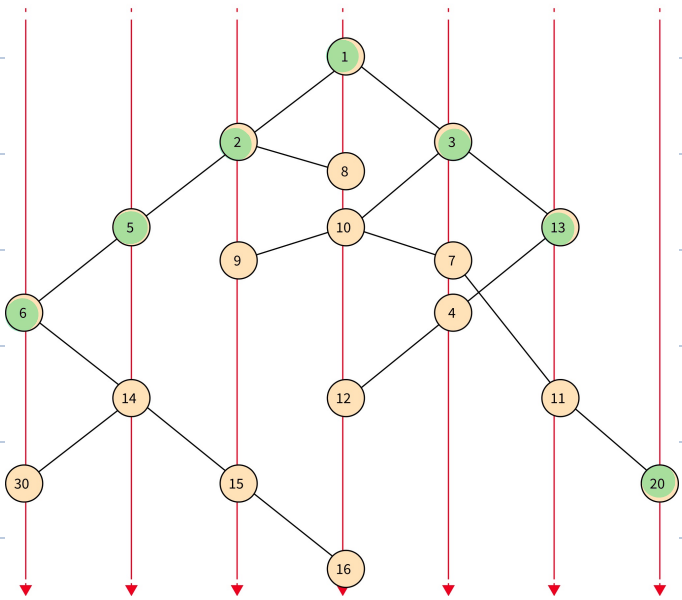
max_d → max horizontal distance // 3

```
for d → min_d to max_d {
    for x in hm.get(d) {  // for each loop
        print(x)
    }
    print("\n")
}
```

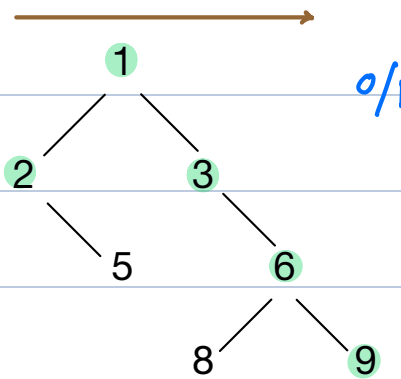
TC = O(N)  SC = O(N+N+N) = O(N)

**< *Question* > :** Find top view of binary tree.






**Quiz :**




o/p → 2 1 3 6 9

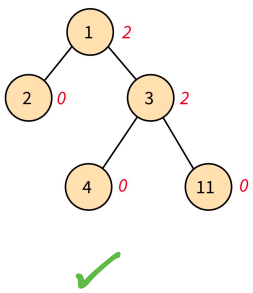
Sol → Print first element of every vertical line ⇒ first node ∀ list in hm. ✓

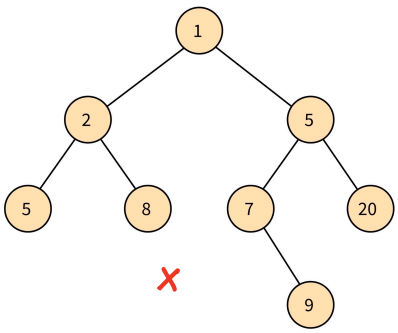
H.W → Bottom view .

# Types of Binary Tree [ Structure ]

## 1. Proper/ Full Binary Tree

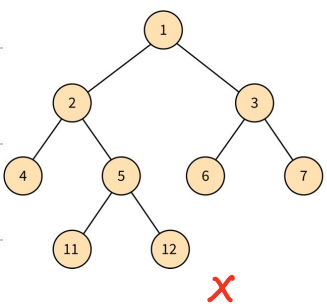
*Every node has either 0 or 2 children.*

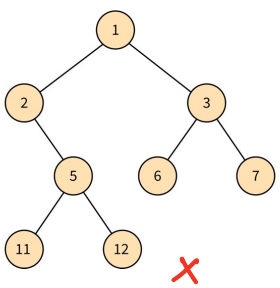


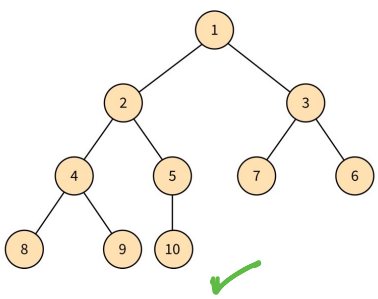



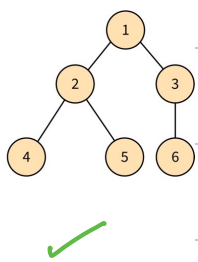

## 2. Complete Binary Tree (C.B.T)

*All levels must be completely filled except possibly the last level and the last level must be filled from left to right.*



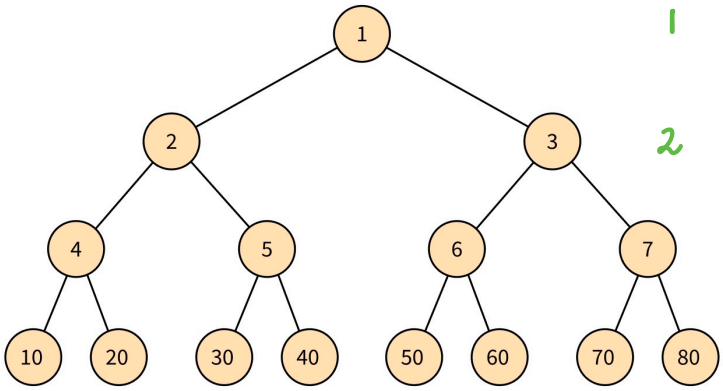



## 3. Perfect Binary Tree

*All levels are completely filled.*




$1 \quad 2^0$

$2 \quad 2^1$

$4 \quad 2^2$

$8 \quad 2^3$

$\vdots$

$2^H$

$N = 15$

$H = 3$

Q → Given a perfect binary tree with N nodes, find height of the tree.

$$1 + 2 + 4 + \ldots + 2^H = N$$

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

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

$$\Rightarrow H + 1 = \log_2 (N+1)$$

$$\Rightarrow \boxed{H = \log_2 (N+1) - 1} \rightarrow O(\log(N))$$

# Balanced Binary Tree

For all nodes

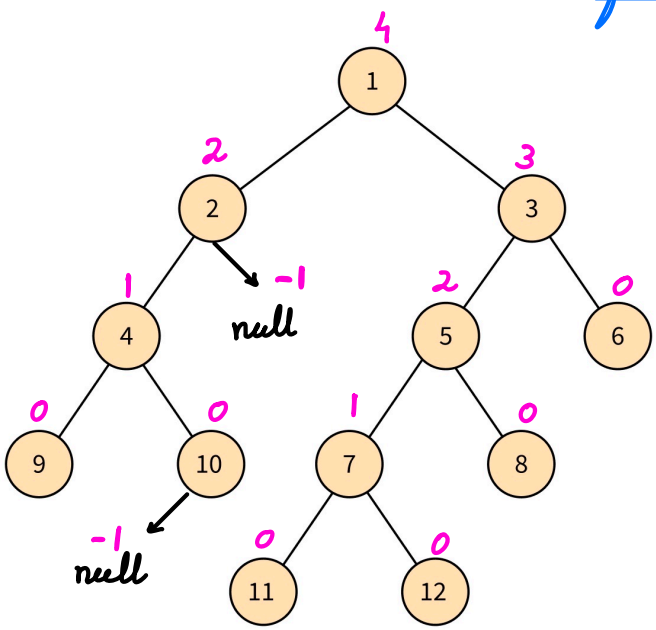
| ht. of left child − ht. of right child | ≤ 1 |
|---|---|

$|1 - (-1)| = 2 \neq 1$

**< Question > :** Given a binary tree, check if it is balanced or not.

Ans = false

Height → Distance of node to farthest leaf.




Left Right Node

isB = true

```
int height(root) {
    if (root == null) return -1
    L = height(root.left)
    R = height(root.right)
    if (abs(L - R) > 1)   isB = false
    return max(L, R) + 1
}
```

TC = O(N)  SC = O(N)