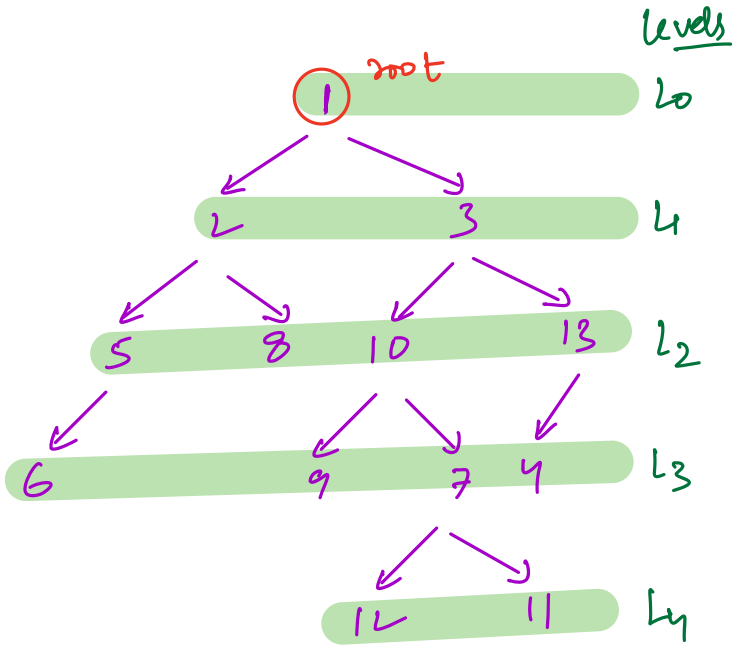
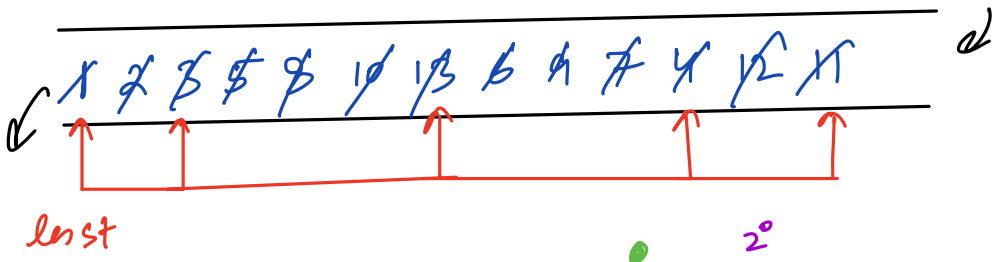
Trees 2 : Views & Types

level Order traversal

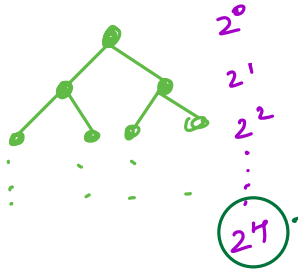
o/p → 1 ✓
2 ✓ 3 ✓
5 ✓ 8 ✓ 10 ✓ 13 ✓
6 ✓ 9 ✓ 7 ✓ 4 ✓
12 ✓ 11 ✓



fifo → queue




```
if (root == null)   return
q.enqueue(root)
last = root
while ( !q.isEmpty() ) {
    x = q.dequeue()
    print(x.data)
    if (x.left != null)   q.enqueue(x.left)
    if (x.right != null)  q.enqueue(x.right)
```



$N = 2^0 + 2^1 + \dots + 2^H = \frac{2^0(2^{H+1}-1)}{2-1}$

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

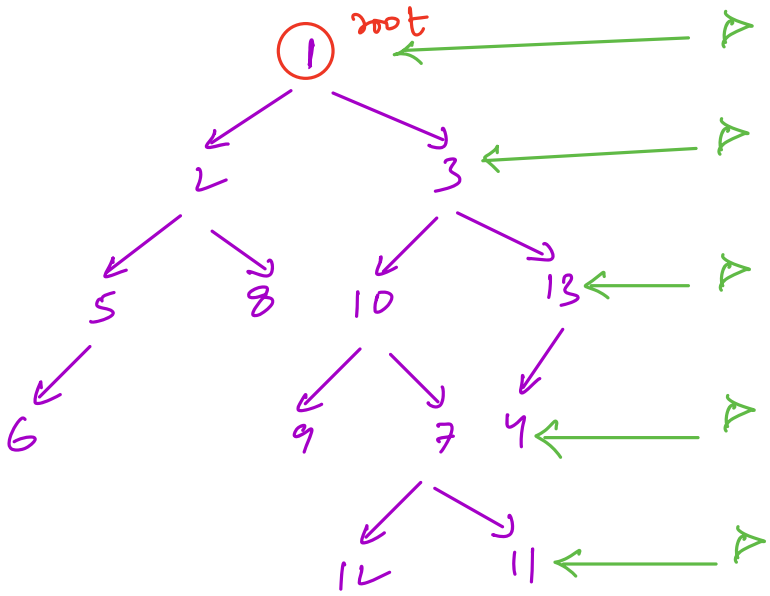
$N+1 = 2^{H+1}$

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

```
    if (x == last) {
        print(newline)  // change line
        if(!q.isEmpty()) last = q.rear()
    }
}
```

TC = O(N)
SL = O(N)

Ques → Print the right view of binary tree.



o/p → 1 3 13 4 11

Soln → print last node of every level

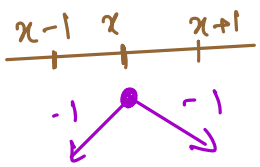
```
if(root == null)  return
q.enqueue(root)
last = root
while(!q.isEmpty()) {
    x = q.dequeue()
    if(x.left != null)  q.enqueue(x.left)
    if(x.right != null)  q.enqueue(x.right)
```

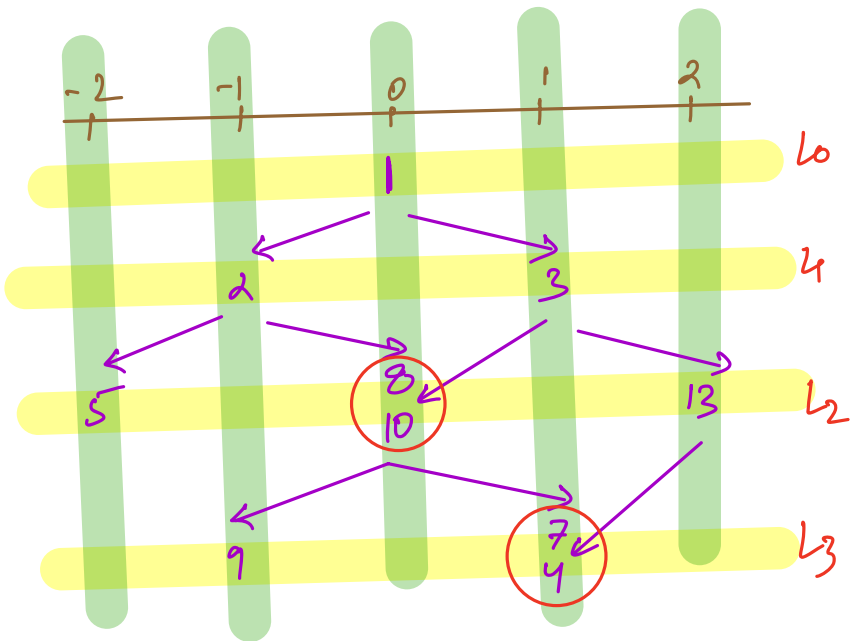
```
if (x == last) {
    print(x.data)
    if(!q.isEmpty()) last = q.rear()
}
}
```

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

HW → print left view of binary tree

Vertical order traversal





Print each vertical line from top to bottom.

In case of overlap, first print element coming from left side

o/p → 5
2 9
1 8 10
3 7 4
13

top view

Hashmap / Treemap

dist → list of nodes from top to bottom

<int, list<node>>

(1,0) (2,-1) (3,1) (5,-2) (8,0) (10,0) (13,2) (9,-1) (7,1) (4,1)

key	value
0	1, 8, 10
-1	2, 9
1	3, 7, 4
-2	5
2	13

sorted key (dist) ⇒ Treemap

TC = O(log N) # operations ✗

↓

Hashmap

track mindist & maxdist

```
// level order traversal  <- TODO
for (d = mindist to maxdist) {
    nodeslist = mp.get(d)
    for (x in nodeslist) {
        print(x.data)
    }
    print(newline)
}
```

TC = O(N)

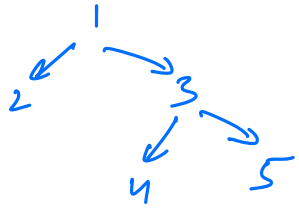
SC = O(N)

HW → print top view ← first node # dist

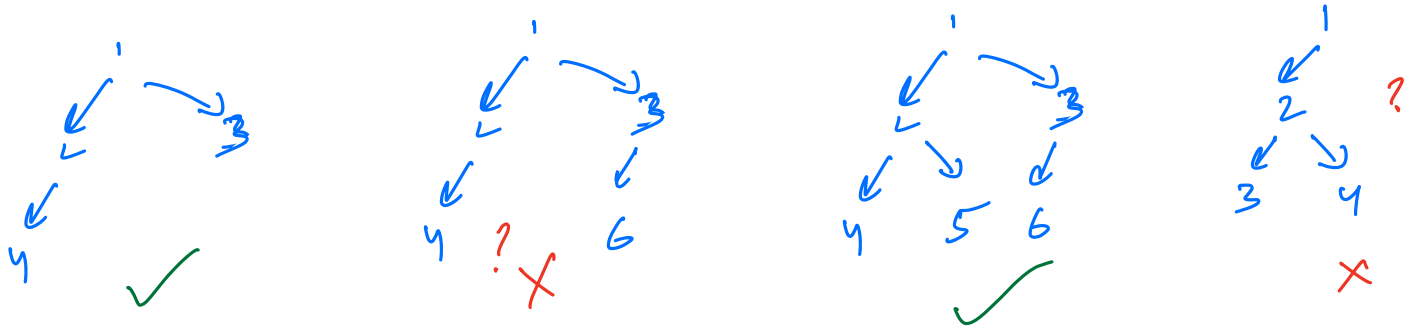
print bottom view ← last node # dist

Types of binary tree (w.r.t structure)

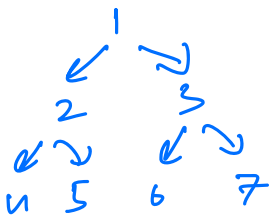
1. **Proper binary tree** → Every node has either 0 or 2 children.

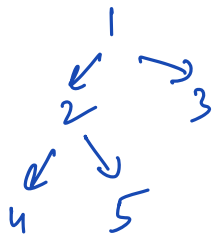


2. **Complete binary tree** → All levels are complete except maybe the last level which is filled from left to right.



3. **Perfect binary tree** → All levels are complete.
(full binary tree)

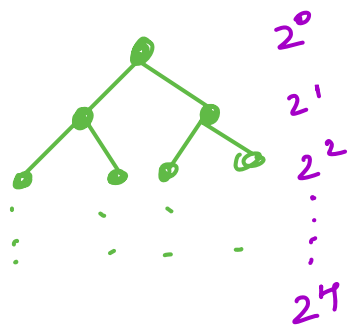




Proper BT ✓

Complete BT ✓

Perfect BT ✗



$$N = 2^0 + 2^1 + \dots + 2^H = \frac{2^0(2^{H+1}-1)}{2-1}$$

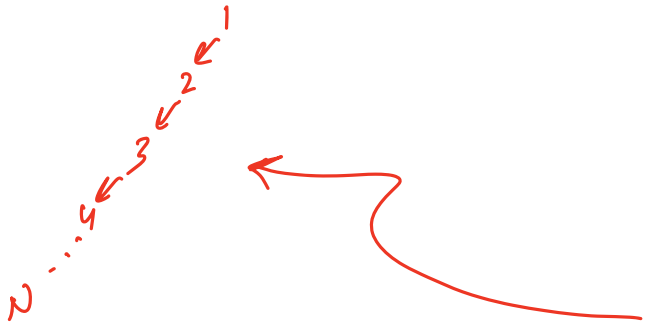
$= 2^{H+1} - 1$

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

$\Rightarrow H = O(\log(N))$ ← min height of BT

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

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



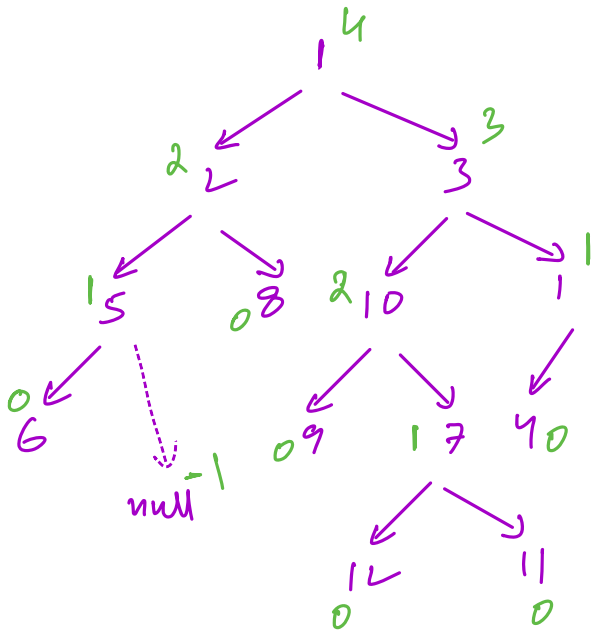
max height of BT = N

Height balanced tree

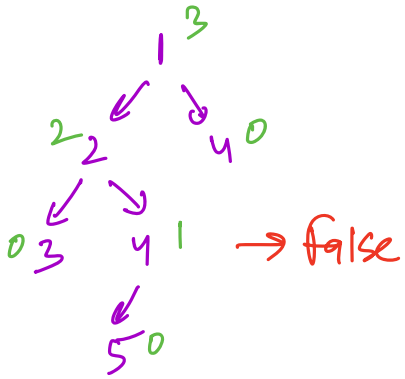
∀ nodes,

$$\left| \text{height of left child} - \text{height of right child} \right| <= 1$$

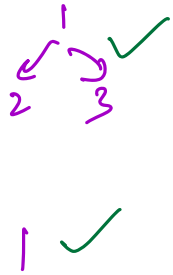
Ques → check if the given binary tree is height balanced?



o/p → True (height balanced)



→ false



height(node) = max(height(left), height(right)) + 1

Postorder → L R N

isBalanced = true

int height(root) { (root)

if (root == null) return -1

L = height(root.left)

R = height(root.right)

if(abs(L-R) > 1) isBalanced = false

return max(L,R) + 1

}

TC = O(N)

SC = O(H)