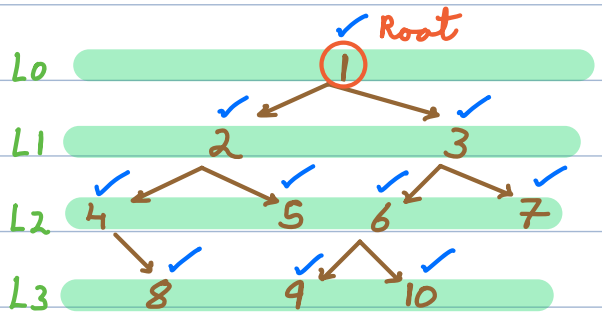
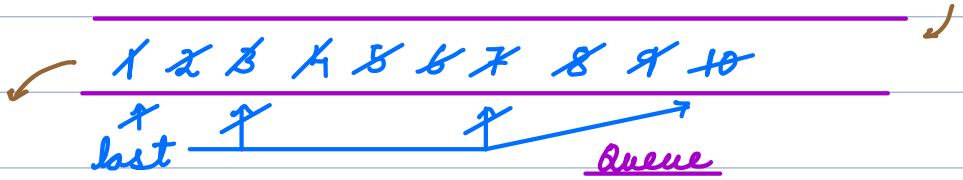
Level Order Traversal

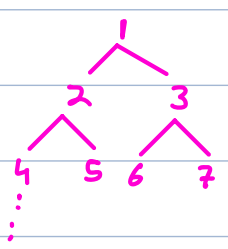


o/p → 1 2 3 4 5 6 7 8 9 10




```
last = root
q.enqueue(root)   // root != null
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)

    if( x == last && !q.isEmpty() ) {
        print("\n")
        last = q.rear()
    }
}
```

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



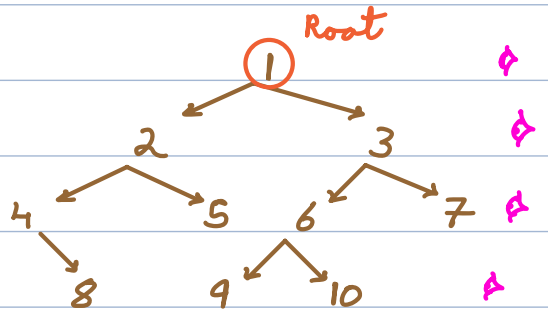
1
2
4
2^K

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

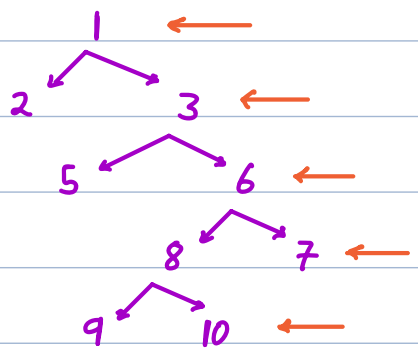
$\Rightarrow 2^{K+1} - 1 = N \Rightarrow 2 * 2^K = N+1$

$\Rightarrow 2^K = (N+1)/2$

Q → Print right view of binary tree.



o/p → 1 3 7 10



o/p → 1 3 6 7 10

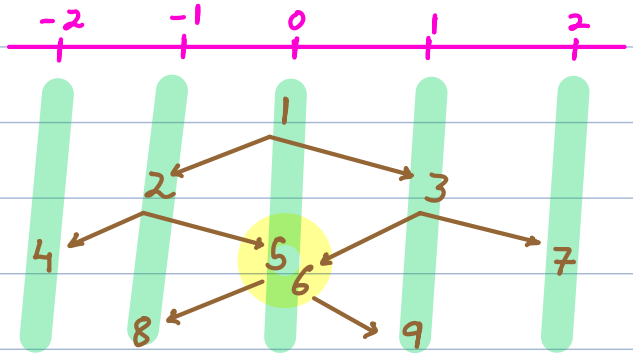
Last node of every level.

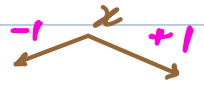
```
last = root
q.enqueue(root)   // root != null
while (!q.isEmpty()) {
    x = q.dequeue()

    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 → Left view (first node of every level)

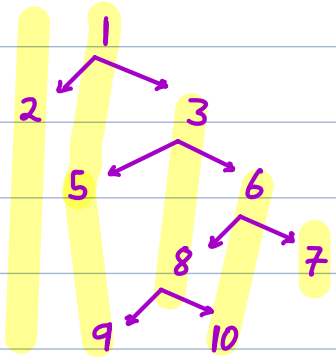
Vertical Order Traversal





dashing → first take left side data then right.

o/p → 4
2 8
1 5 6
3 9
7



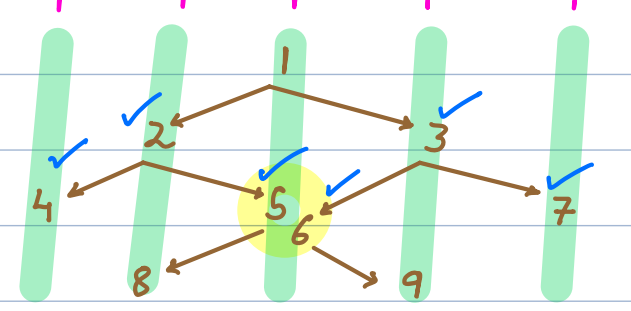
o/p → 2
1 5 9
3 8
6 10
7

```
if (root == null) return
q.enqueue(root)    // root != null
hm.put(0, {root})
dist[root.data] = 0
while (!q.isEmpty()) {
    x = q.dequeue()
    if (x.left != null) {
        dist[x.left.data] = dist[x.data] - 1
        q.enqueue(x.left)
```

```
            hm.get(dist[x.left.data]).add(x.left)
        }
        if (x.right != null) {
            dist[x.right.data] = dist[x.data] + 1
            q.enqueue(x.right)
            hm.get(dist[x.right.data]).add(x.right)
        }
    }
}
```

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

```
// maxdist, mindist
for i → mindist to maxdist {
    al = hm.get(i)
    for (x : al){ print(x.data) }
    print("\n")
}
```

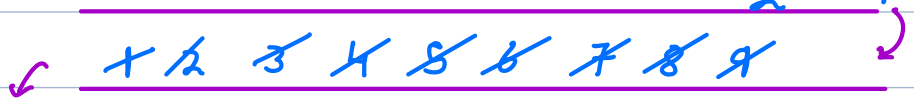


dist →

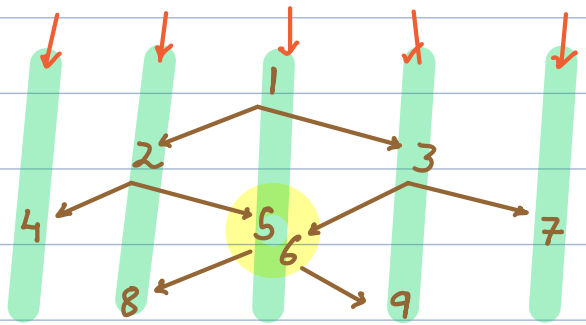
1	2	3	4	5	6	7	8	9
0	-1	1	-2	0	0	2	-1	1

map →

Key	Value
0	1, 5, 6
-1	2, 8
1	3, 9
-2	4
2	7



Q → Top view of binary tree.



o/p → 4 2 1 3 7

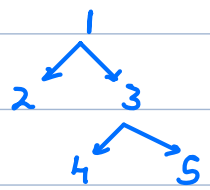
(first element of vertical line)

H.W → Bottom View (last element of vertical line)

Types of Binary Tree

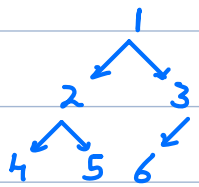
1) **Proper Binary Tree** → ∀ nodes, either 0 or 2 children.

Eg →



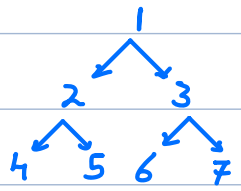
2) **Complete Binary Tree** → All levels are completely filled, except maybe for the last level where all nodes are as left as possible.

Eg →

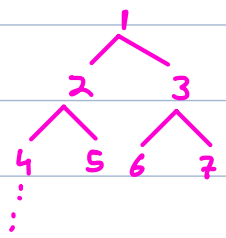


3) **Perfect Binary Tree** → All levels are completely filled.

Eg →



Find height of perfect binary tree with N nodes.



1 $\quad 1 + 2 + 4 + \ldots 2^K = N$

2 $\Rightarrow 2^{K+1} - 1 = N$

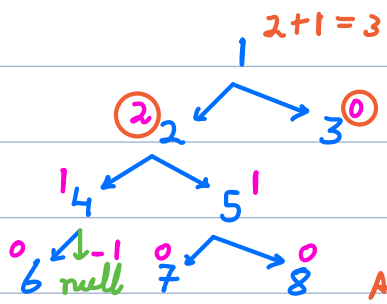
4 $\Rightarrow 2^{K+1} = N+1$

$2^K \Rightarrow K+1 = \log(N+1)$

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

Q → check if the given tree is height balanced.

$\forall$ nodes, $|height(left) - height(right)| <= 1$



height (leaf) = 0

height (null) = -1

Ans = false

height (2) − height (3) > 1

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

}

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