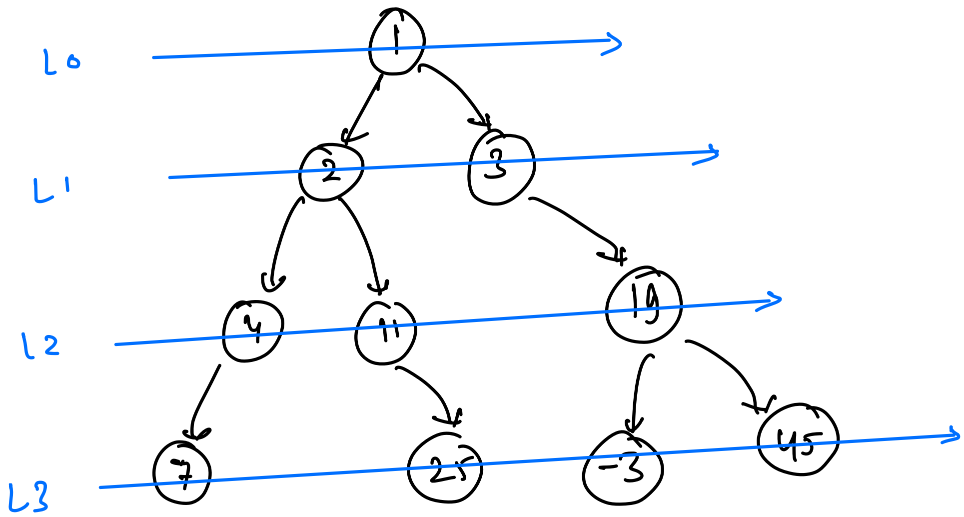
Level Order Traversal



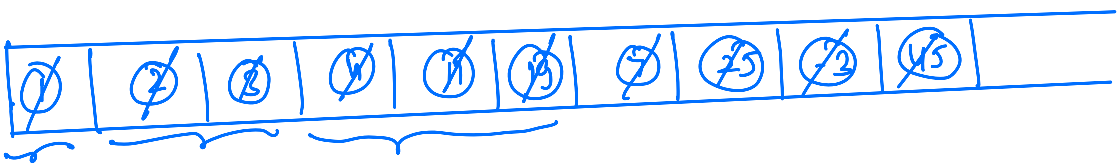
O/P. →

1

2 3

4 11 19

7 25 -3 45



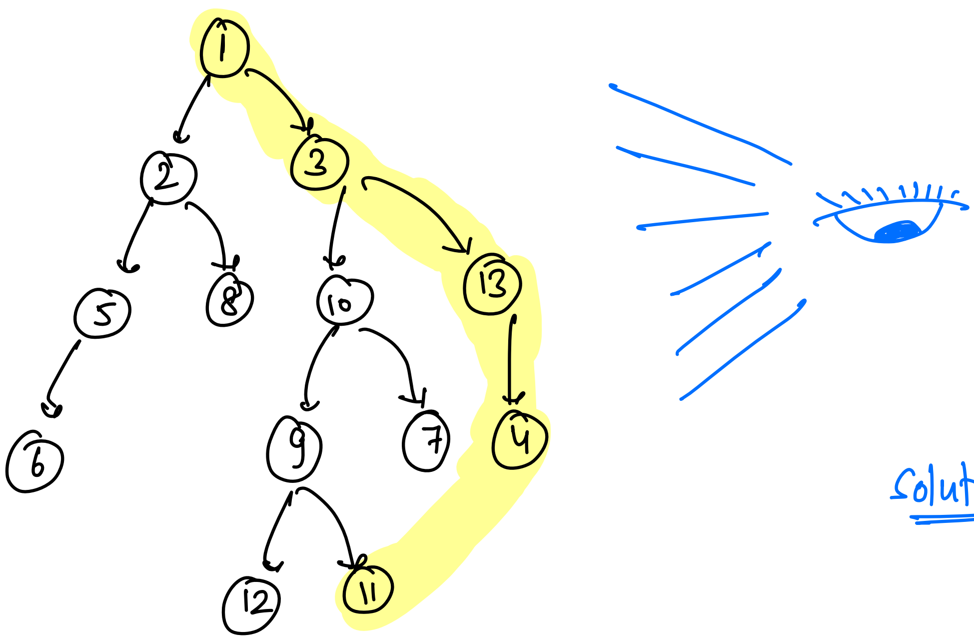
{
1
→ 2 3
→ 4 11 19
→ 7 25 -3 45
} → output.

code.

```
Queue<Node> q;
q.enqueue(root);
while( q.isEmpty() == false) {
    int sz = q.size();
    for( i = 1; i <= sz ; i++) {
        Node x = q.dequeue();
        print( x.val);
        if( x.left != NULL) { q.enqueue( x.left); }
        if (x.right != NULL){ q. enqueue (x.right); }
    }
    print("\n");
}
```

T.C → O(N)
S.C → O(N)

Q: Find right view of binary tree.



O/P. →

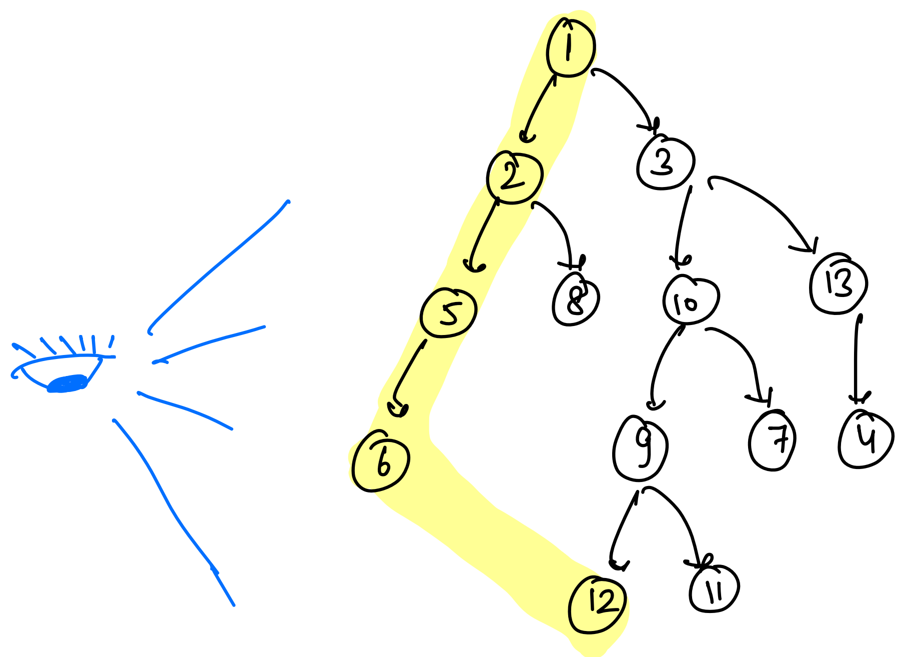
1, 3, 13, 4, 11

Solution - Print the last element of every level.

```
Queue<Node> q;
q.enqueue(root);
while(q.isEmpty() == false) {
    int sz = q.size();
    for(i = 1; i <= sz; i++) {
        Node x = q.dequeue();
        if(i == sz) { print(x.val); }
        if(x.left != NULL) { q.enqueue(x.left); }
        if(x.right != NULL) { q.enqueue(x.right); }
    }
    print("\n");
}
```

T.C → O(N)
S.C → O(N)

Q: Left View of Binary tree



o/p. → 1, 2, 5, 6, 12.

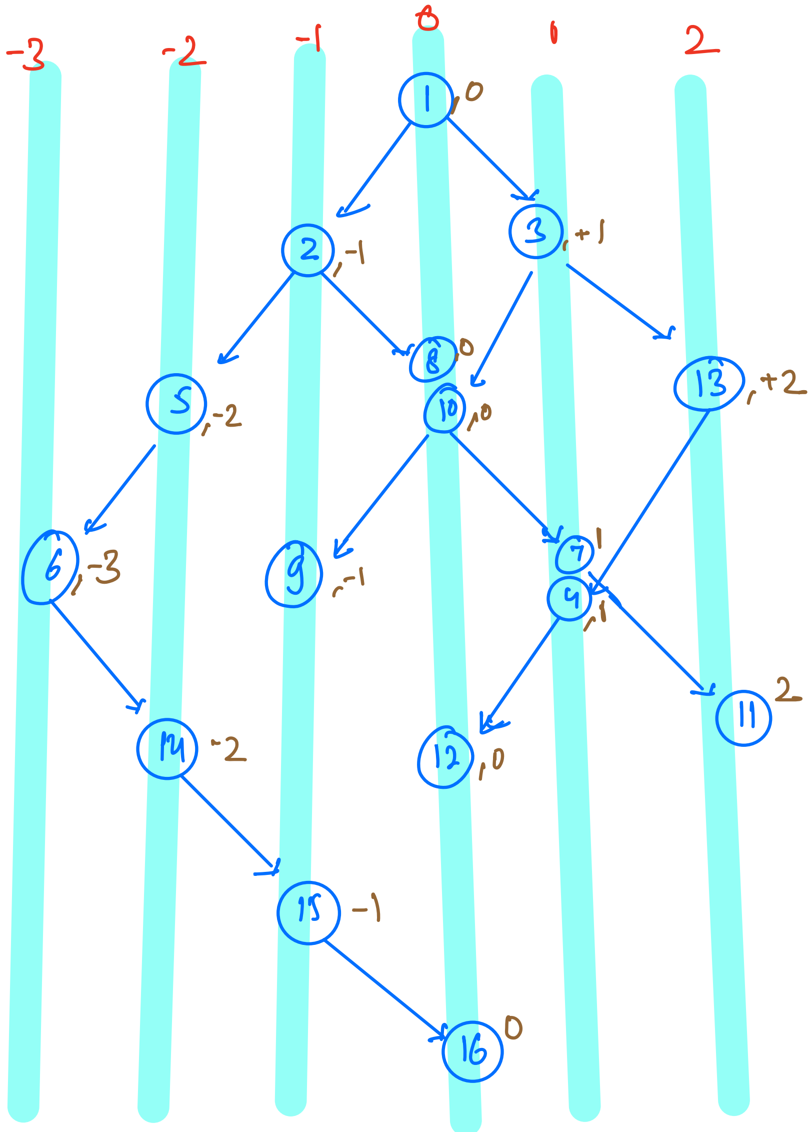
Solution → print first element of every level.

```
Queue<Node> q;
q.enqueue(root);
while ( q.isEmpty() == false) {
    int sz = q.size();
    for( i = 1; i <= sz ; i++) {
        Node x = q.dequeue();
        if( i == 1 ) { print( x.val); }
        if( x.left != NULL) { q.enqueue( x.left); }
        if (x. right != NULL){ q. enqueue (x.right); }
    }
    print("\n");
}
```

$$\begin{bmatrix} T.C \rightarrow O(N) \\ S.C \rightarrow O(N) \end{bmatrix}$$

Microsoft

Vertical Order Traversal of Binary Tree



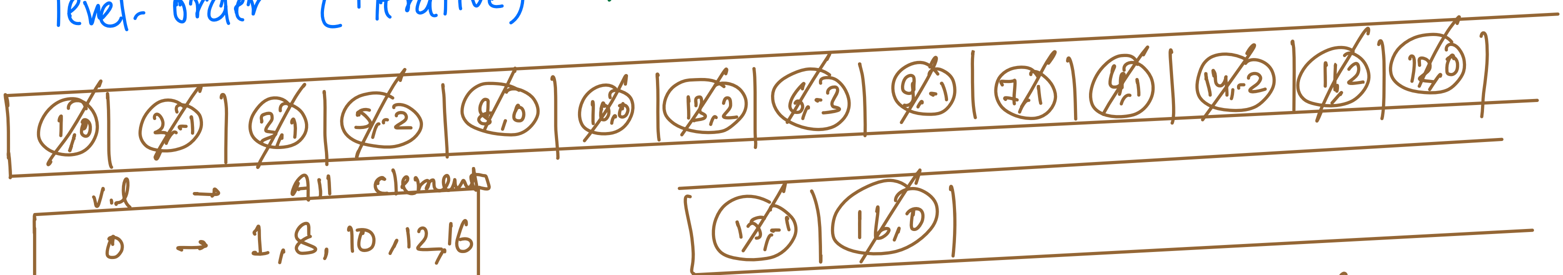
o/p. →

[
6
5 14
2 9 15
1 8 10 12 16
3 7 4
13 11
]

v.l	All nodes
0	→ 1 ,16 ,8 ,10 ,12
-1	→ 2 ,15 ,9
-2	→ 5 ,14
-3	→ 6
1	→ 3 ,7 ,4
2	→ 11 ,13

Pre-order (recursion) X

level-order (iterative) ✓



v.l →	All elements
0	→ 1, 8, 10 ,12, 16
-1	→ 2 ,9 ,15
1	→ 3 ,7 ,4.
-2	→ 5 ,14
2	→ 13 , 11
-3	→ 6

Pair<Node, vl>

```
class Pair {
    Node n;
    int vl;
}
```

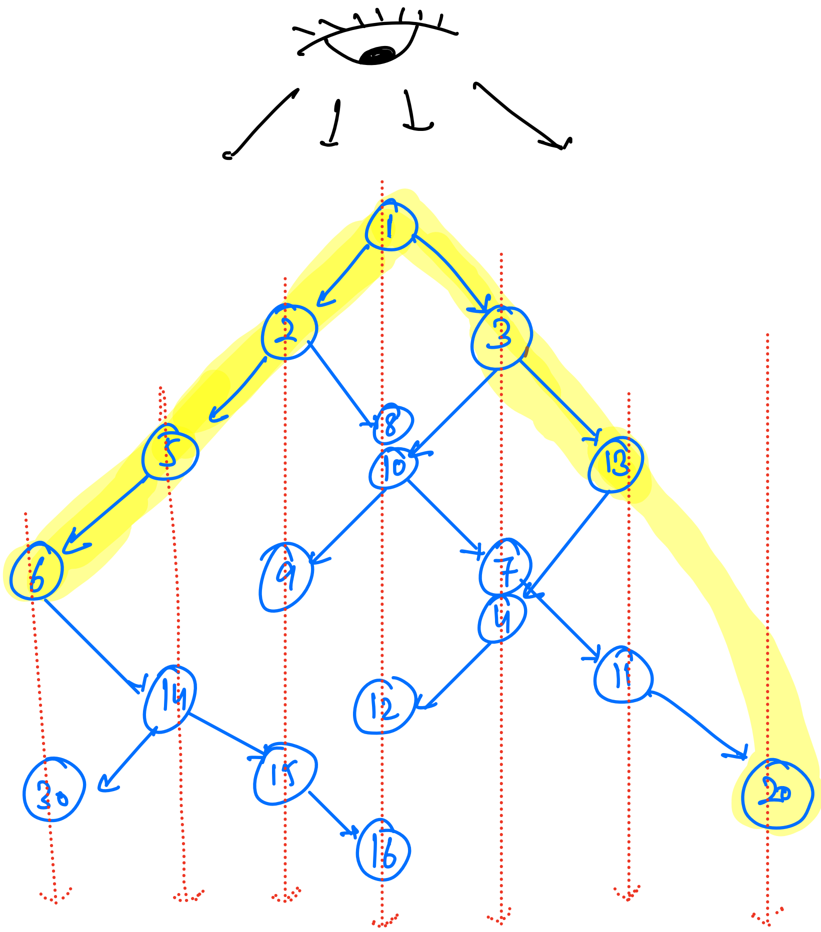
code →

```
Pair rootpair = new Pair( root, 0);

Queue < Pair> q ;     q.enqueue(root pair);

Hashmap< int, list<int> > map ;    minvl = 0 , maxvl = 0

while ( q.isEmpty() == false){

    Pair rp = q.dequeue();

    minvl = Min(minvl, rp.vl) , maxvl = Max(maxvl, rp.vl);

    Insert rp.node.val in the hashmap against rp.vl

    if( rp.node.left != NULL){
        q.enqueue( new Pair(rp.node.left, rp.vl -1 ));
    }

    if( rp.node.right != NULL){
        q.enqueue( new Pair(rp.node.right, rp.vl +1 ));
    }
}

for( i = minvl; i <= maxvl ; i++){
        print( map[i]);
        print("\n");
}
```

T.C → O(N)

S.C → O(N)

Q. Top View of Binary Tree



o/p. →

[6, 5, 2, 1, 3, 13, 20]

v.l	→	All elements
0	→	1, 8, 10, 12, 16
-1	→	2, 9, 15
1	→	3, 7, 4
-2	→	5, 14
2	→	13, 11
-3	→	6
3	→	20

solution → print first node of every vertical level.

Bottom View of a Binary tree

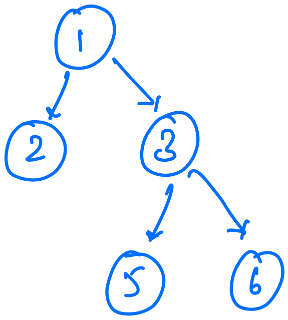
solution → print last node of every vertical level.

Types of Binary Tree (structure)

① Proper/ Full Binary Tree

For every node, either 0 or 2 children.

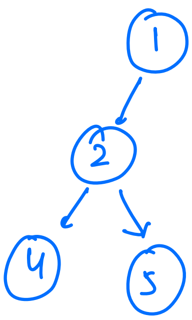
Eg-



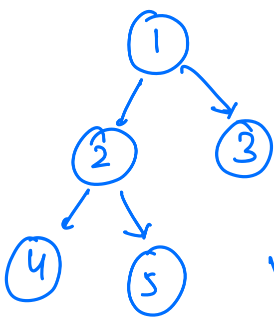
✓

② Complete Binary Tree [C.B.T]

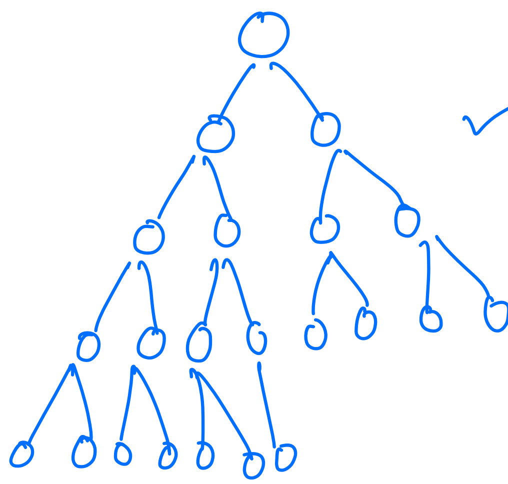
Every level must be completely filled except maybe the last level. In the last level, all nodes are filled from left to right.



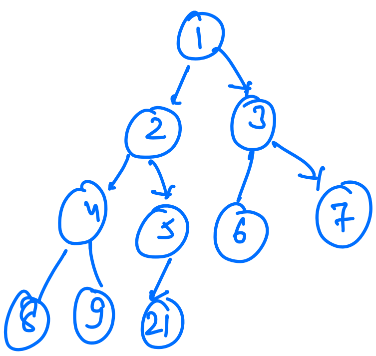
✗



✓

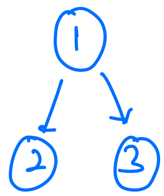


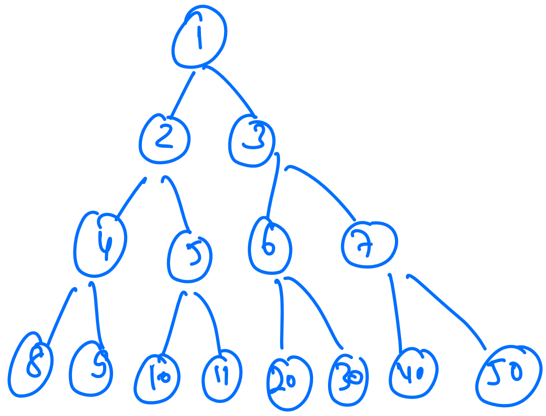
✓



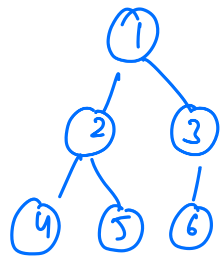
3. Perfect Binary Tree

All levels must be completely filled.





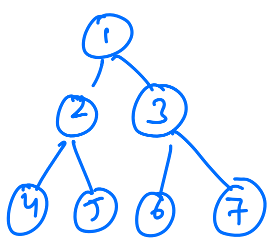
① Are all complete binary trees, also proper tree? <u>No</u>



C.B.T ✓

Proper ✗

② Are all perfect binary trees, also complete tree? <u>Yes.</u>



Perfect Binary Tree ✓

C.B.T ✓

③ Are all perfect binary trees, also proper tree?

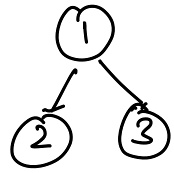
<u>Yes.</u>

Q: Given a perfect binary tree with N nodes. Find height of the tree.

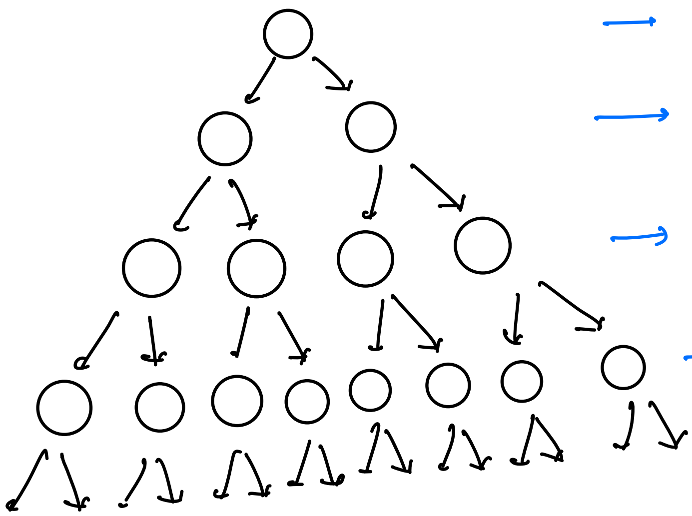
$\underline{\underline{N=1}}$

ht = 0

$\underline{\underline{N=3.}}$



ht = 1



→ 1 $[2^0]$

→ 2 $[2^1]$

→ 4 $[2^2]$

→ 8 $[2^3]$

→ 16 $[2^4]$

→ H $[2^H]$

$$2^0 + 2^1 + 2^2 + 2^3 + \text{— —} \; 2^H = N$$

$a = 1, r = 2$

no. of terms = h+1 terms

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

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

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

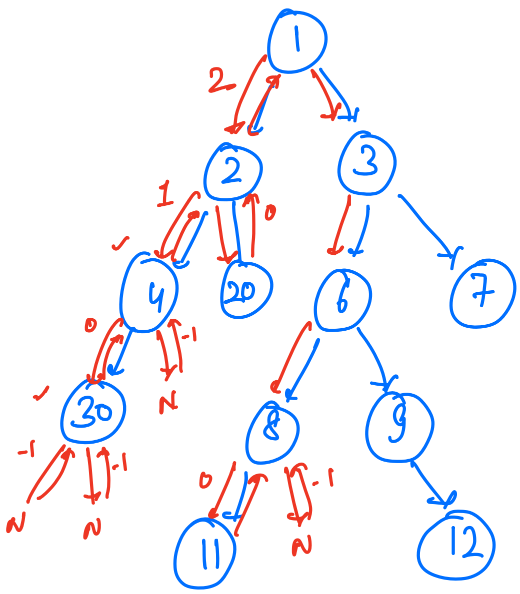
$[\log N \le \text{Ht. of tree} < N]$

Balanced Binary Tree

$\forall$ nodes

$$\left| \text{ht. of left child} - \text{ht. of right child} \right| \le 1$$

Q: Given a binary tree, check if it's balanced or not.



Not balanced

boolean isbalanced → true

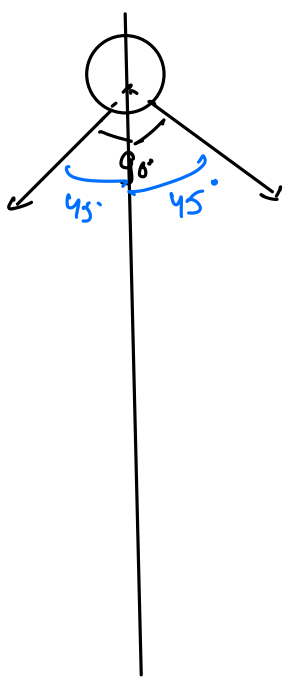
```
int height(Node root){
    if(root == NULL){ return -1 }
    lht = height(root.left);
    rht = height(root.right);
    if(Abs(lht - rht) > 1){ isbalanced = false }
    return Max(lht, rht) + 1;
}
```

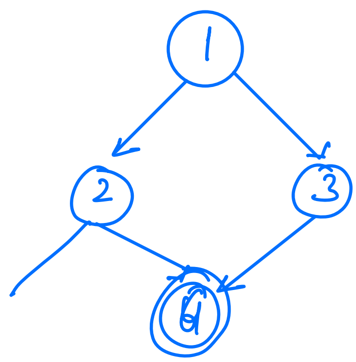
- utilise this function to check whether the tree is balanced or not.

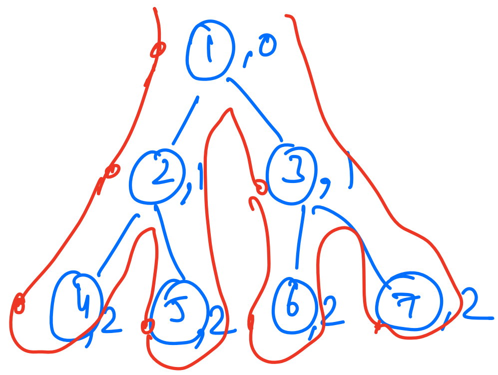
T.C → O(N)

S.C → O(ht. of tree)

#dry-run







level order with recursion?

0 → 1

1 → 2, 3

2 → 4, 5, 6, 7

→ In-order / pre-order traversal

T.C → O(N)

S.C → O(1)

⇓

Morris traversal.