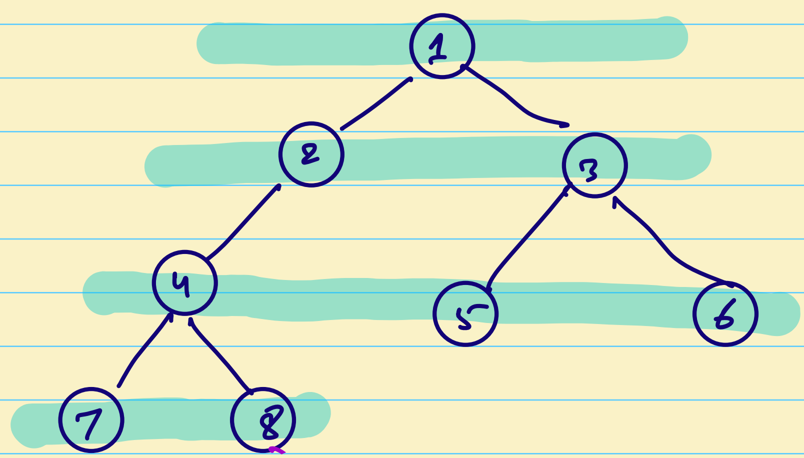
9:05 pm

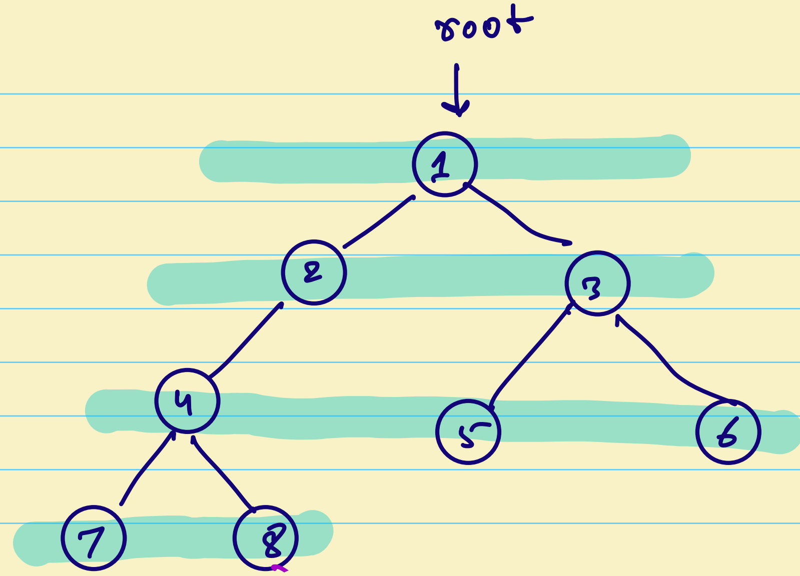
1) Post Order
2) Inorder
3) Preorder.

Level Order Travesal / Bread First Search.



LOT = 1, 2, 3, 4, 5, 6, 7, 8

Level		Node
0	→	1
1	→	2, 3
2	→	4, 5, 6
3	→	7, 8.



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

2

$\boxed{1, 2, 3, 4, 5, 6, 7, 8}$

Steps:

1) Create a queue.

2) add root to the queue.

3) deque

(4) print.

(5) add all the child.

(6) Do this till queue is empty

Pseudo code

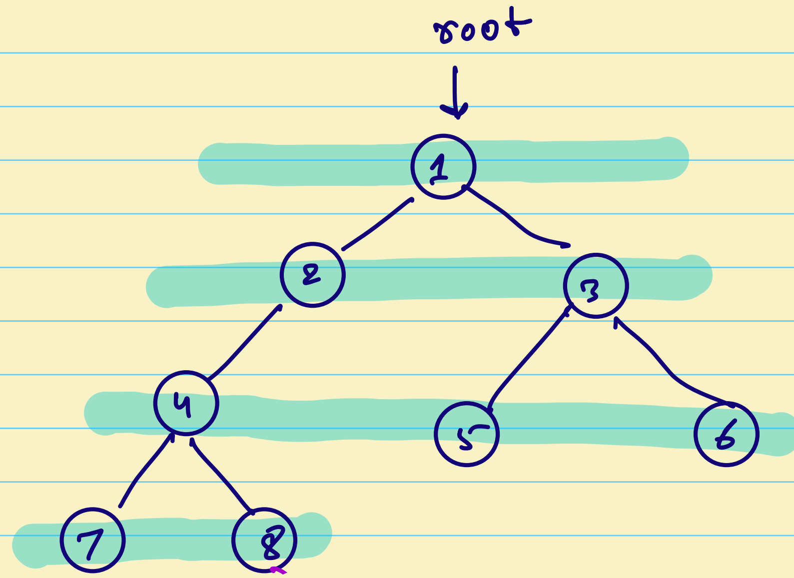
```
q = Queue();
q.enqueue(root);
while (!q.Empty())
{
        node = q.dequeue();
        print(node.data);
        if (node.left != null)
        q.enqueue(node.left)

        if (node.right != null)
        q.enqueue(node.right)
}
```

T.C: O(n)
S.C: O(n).

Print all levels seperated by new line.

ans:

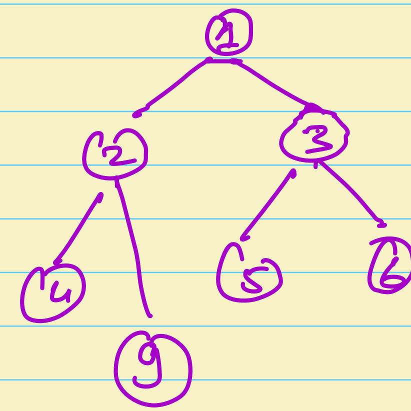
1
2, 3
4, 5, 6
7, 8.



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

queuesize.		no. of element on the level
1		1
2	$\propto$	2
3		3
2		2

```
q = Queue();
q.enqueue(root);
while (!q.Empty())
{
    n = q.size()
    for (i=0; i<n; i++)
    {
        node = q.dequeue();
        print(node.data);
        if (node.left != null)
        q.enqueue(node.left)

        if (node.right != null)
        q.enqueue(node.right)

    }
    print("\n");
}.
```



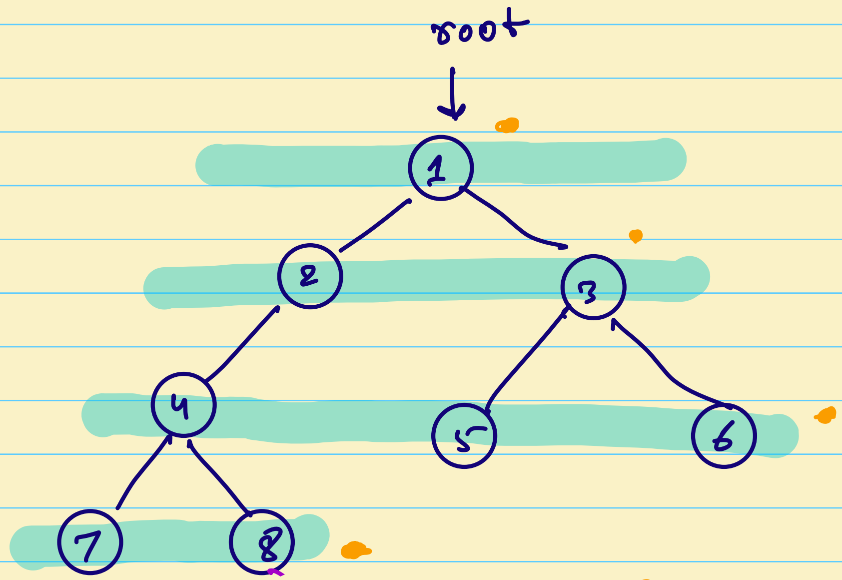
T.C: O(n)
S.C: O(n)

n=1   n=2   n=4.

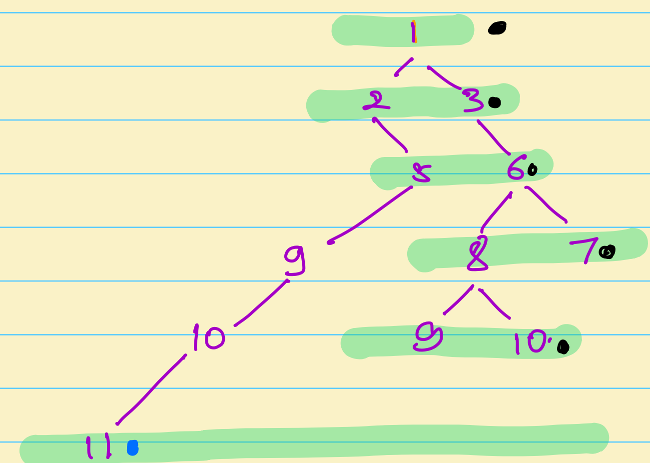
| ~~1~~ | ~~2~~ | ~~3~~ | 4 | 9 | 5 | 6 |

1
2 3
4 9 5 6

Problem Print the right view of the tree.



ans = 1, 3, 6, 8



1, 3, 6, 7, 10

Pseudo

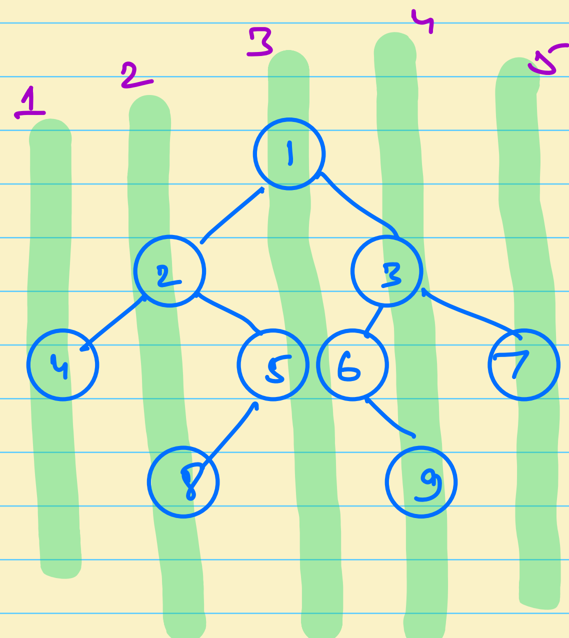
```
q = Queue();
q.enqueue(root);
while (!q.Empty() )
{
        n = q.size();
       for ( i=0; i<n; i++)
    {
        node = q.dequeue();
         if (i==n-1){
         print (node.data);
         }
        if (node.left != null)
       q. enqueue (node.left)

        if (node.right != null) )
       q. enqueue ( node.right

    }  print("\n");
};
```

Left View Tree : :

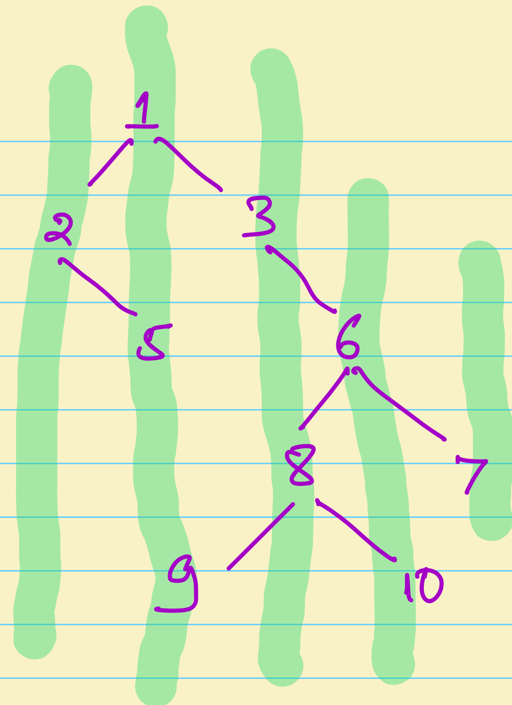
if (i == 0)
print (node.data) ;

T.C. O(n)
S.C. : O(n)

Ques **Vertical order Traversal**



1 : 4
2 : 2, 8
3 : 1, 5, 6
4 : 3, 9.
5 : 7



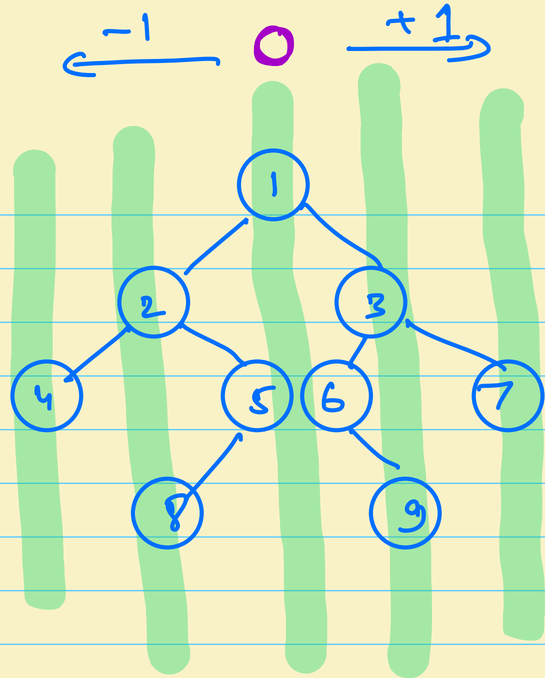
1: 2

2: 1 5 9

3: 3 8

4: 6 10

5: 7



hashmap

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

minValue = -2
maxValue = 2

node level.

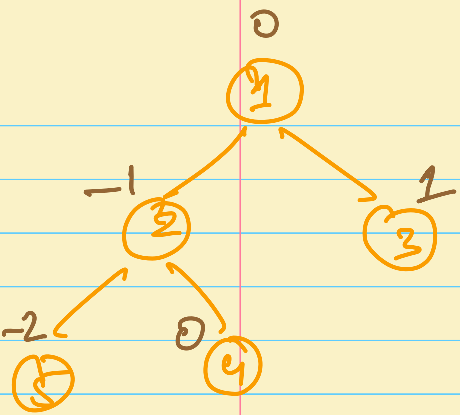
~~(1, 0)~~	~~(2, -1)~~	~~(3, 1)~~	~~(4, -2)~~	~~(5, 0)~~	~~(6, 0)~~	~~(7, 2)~~

~~(8, -1)~~ ~~(9, 1)~~

```
class Pair {
    Node. data;
    int level;
}
```

python.

(Node, level)



q= ~~(3,1)~~ ~~(5,-2)~~, ~~(4,0)~~

hm = {
0 = [1, 4]
-1 = [2]
1 = [3]
-2 = [5]
},

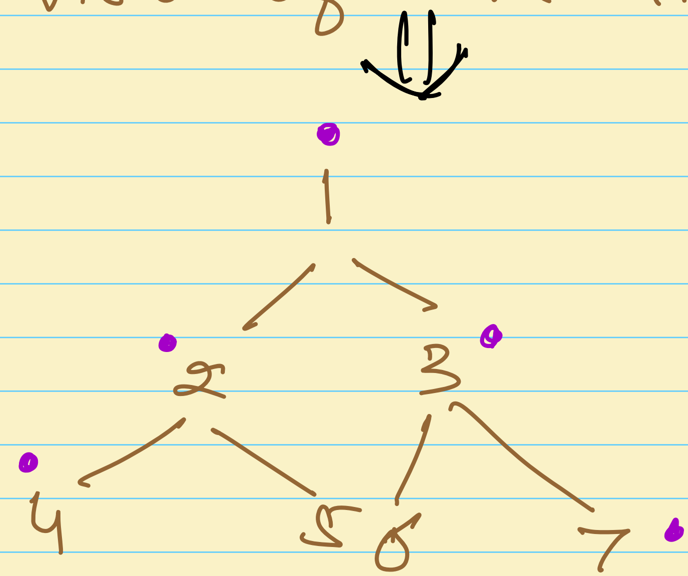
minVal = ~~0~~ ~~-1~~ -2
maxValue = ~~0~~ 1

```
q = Queue()
minValue = inf    max value : -inf.
q. enque( (root, 0) )
hm = hashmap().
while (!q. Emty() )
{
   node, level = q. deque()
   if (level not in hm)  hm[level] = [ ];
   hm[level]. append (node);
   minValue = min(minValue, level);
   maxValue = max (maxValue, level);
   if ( node. left != null)
   {
        q. enque. ( (node.left, level-1));
   }
   if (node. righ != null )
   {
        q. enque ( (node. righ, level+1));
   }
}
```

```
for ( i = minValue;  i <= maxValue ; i++ )
{
        print ( hm[ i ] );
}
}
```

T.C: O(N)
S.C: O(N)

Qus Top view of the Tree.

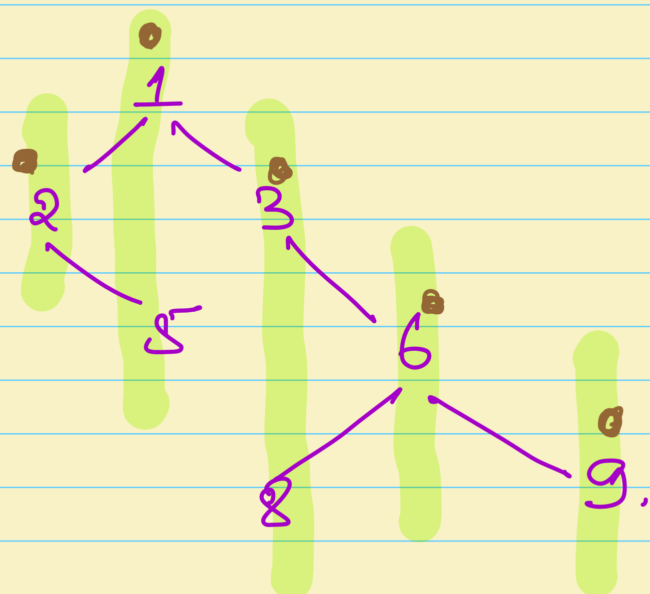


4, 2, 1, 3, 7

```
q =  Queue()
   minValue = inf    max value : -inf.
q. enque(( root, 0 ))
   hm= hashmap().
while  (!q. Empty() )
{
      node, level = q. deque()
      if (level not in hm)  hm[level] = node.

      minValue =  min(minValue, level);
      max Value =  max (maxValue, level);
      if ( node. left != null )
      {
           q. enque( ( node .left , level -1));
      }
      if (node. righ != null )
      {
           q. enque (  ( node. righ , level +1));
      }
}
```

```
for ( i = minValue;  i <= maxValue ; i++ )
{
    print ( hm[i] );
}
```

Quiz



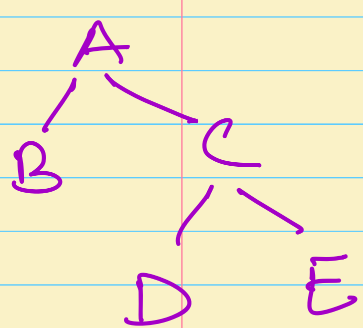
$\boxed{2,\ 1,\ 3,\ 6,\ 9}$

Type Of Binary Tree:

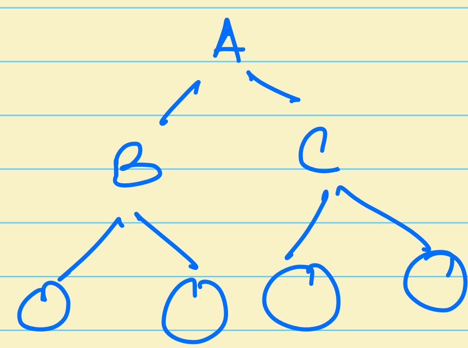
(1) Proper Binary tree: (strict Binary tree).

Every node have either 0 or 2 childs.

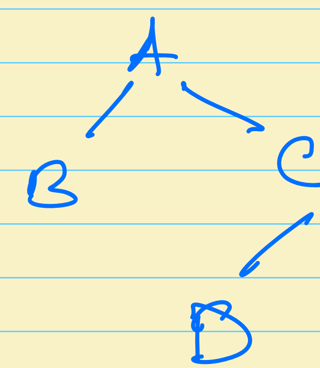
(never 1 child).



Proper Binary tree



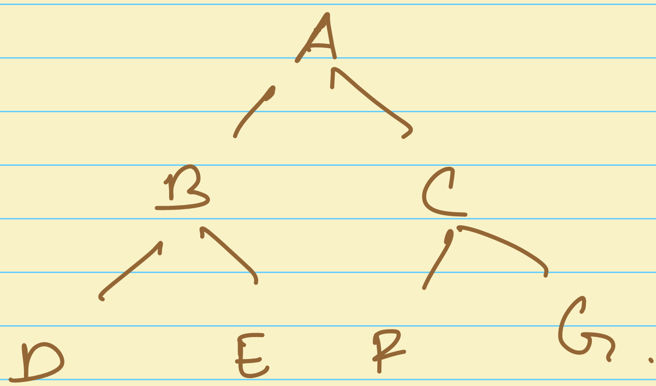
Proper Binary tree.



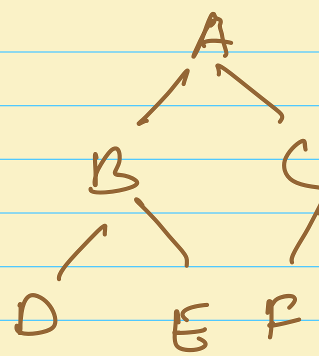
Not Proper Binary tree.

(2) Complete Binary tree.

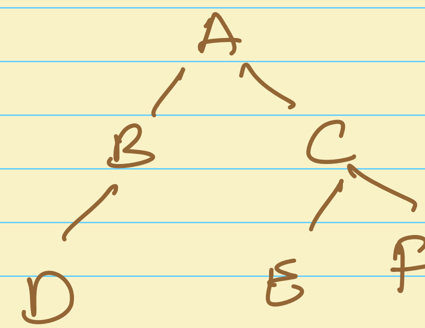
All the levels are fixed except possibly the last level. which is filled from left to right.



complete Binary

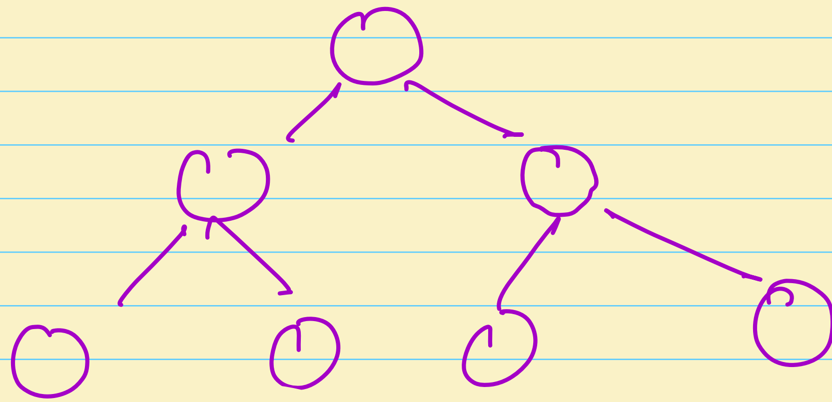


complete Binary



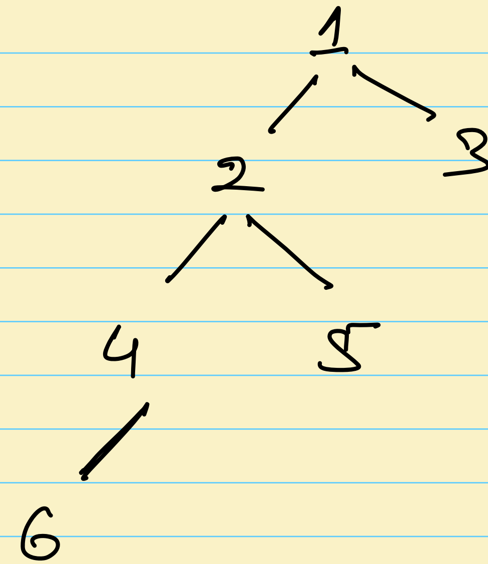
not complete Binary.

(3) Perfect Binary Tree.



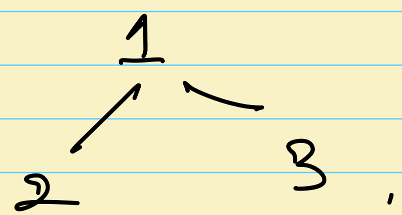
All the internal node have exactly 2 nodes. and all the leaf node are at the same level.

Que Find the heigh of a binary tree

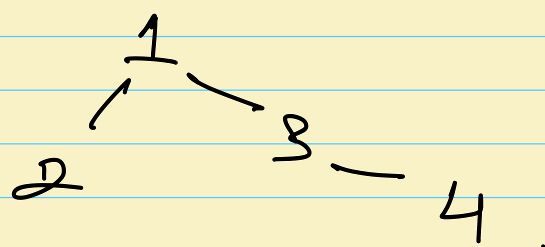


heigh from root = 3

height from 2 = 2



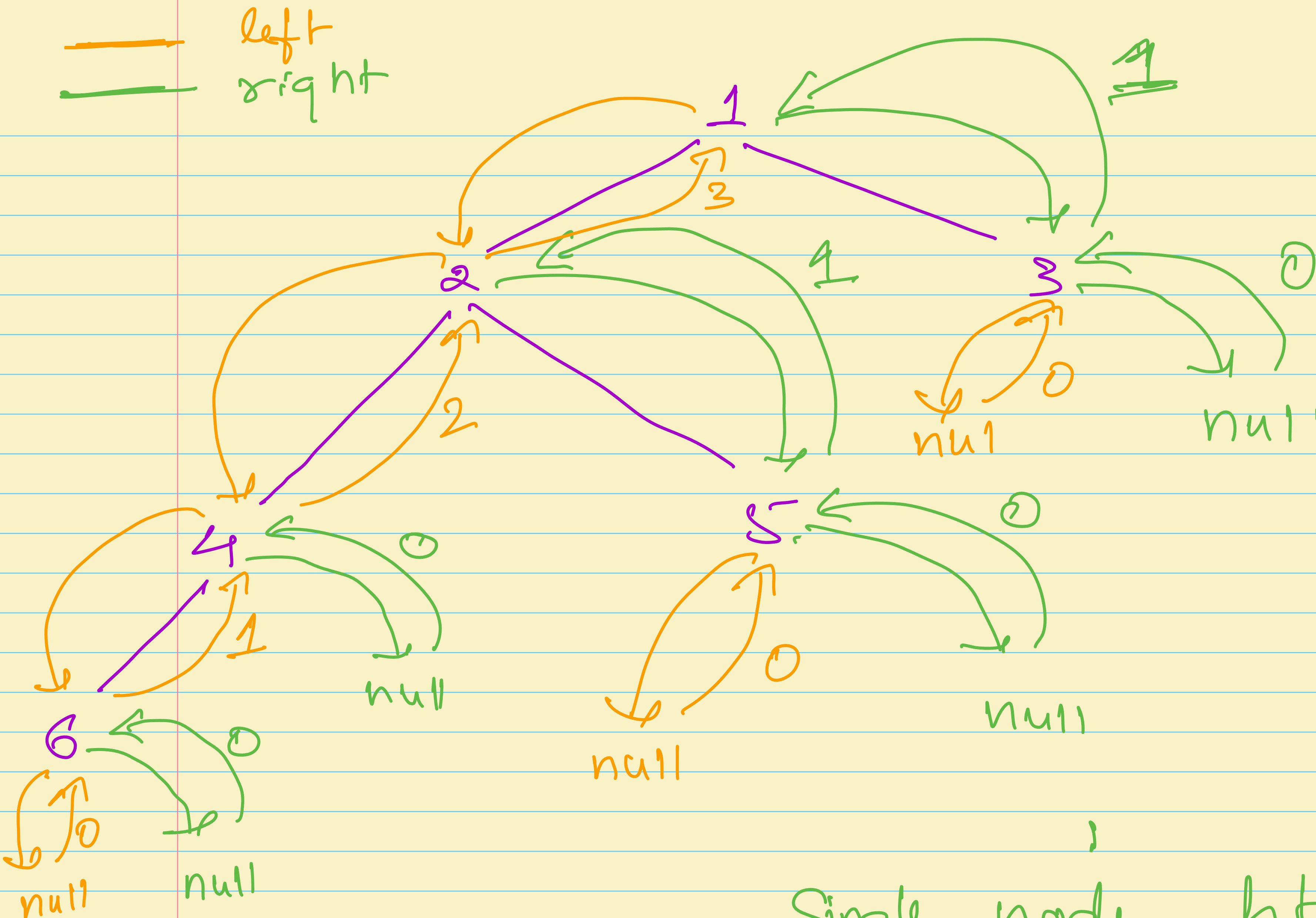
= heigh = 1.



heigh = 2

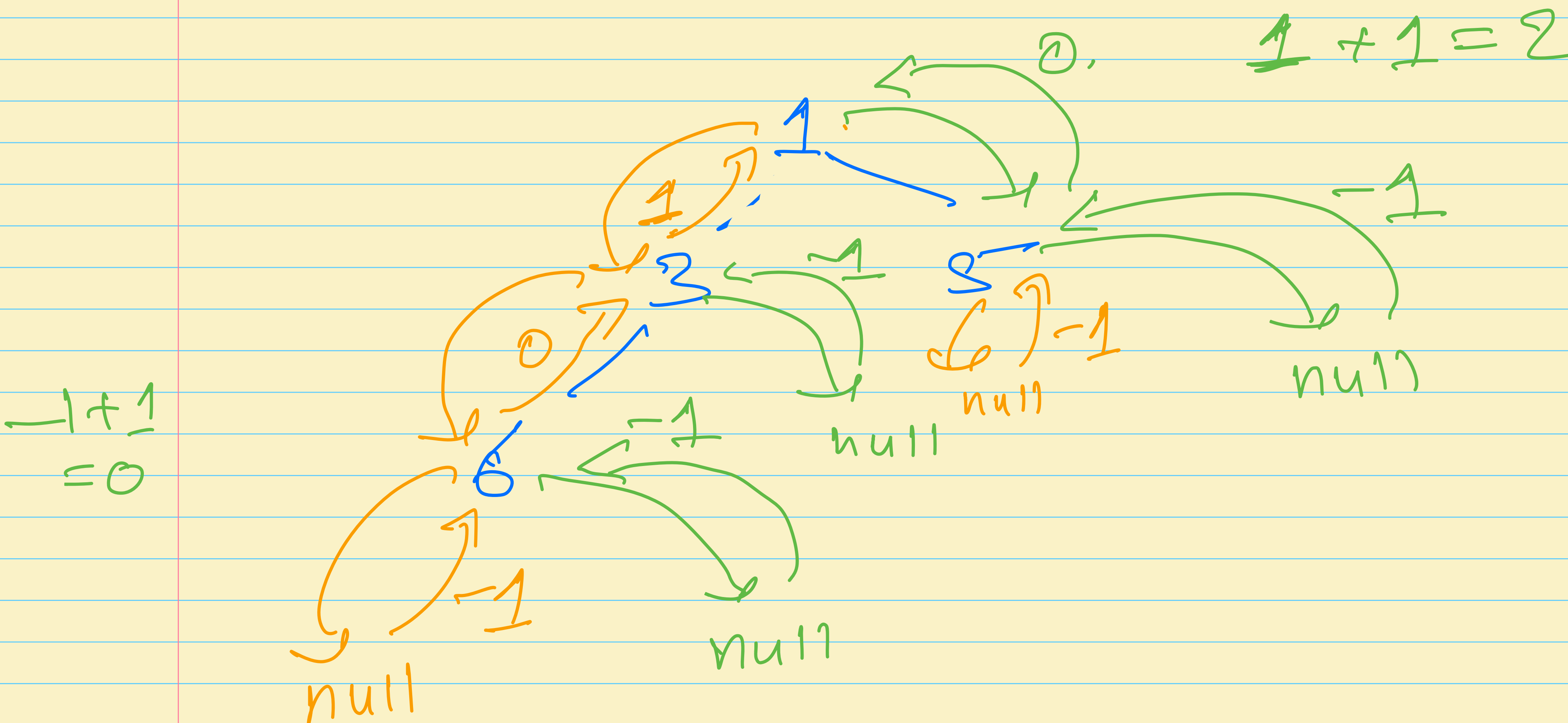
Hieigth of tree = max (ht. of left subtree , ht. of right subtree) + 1

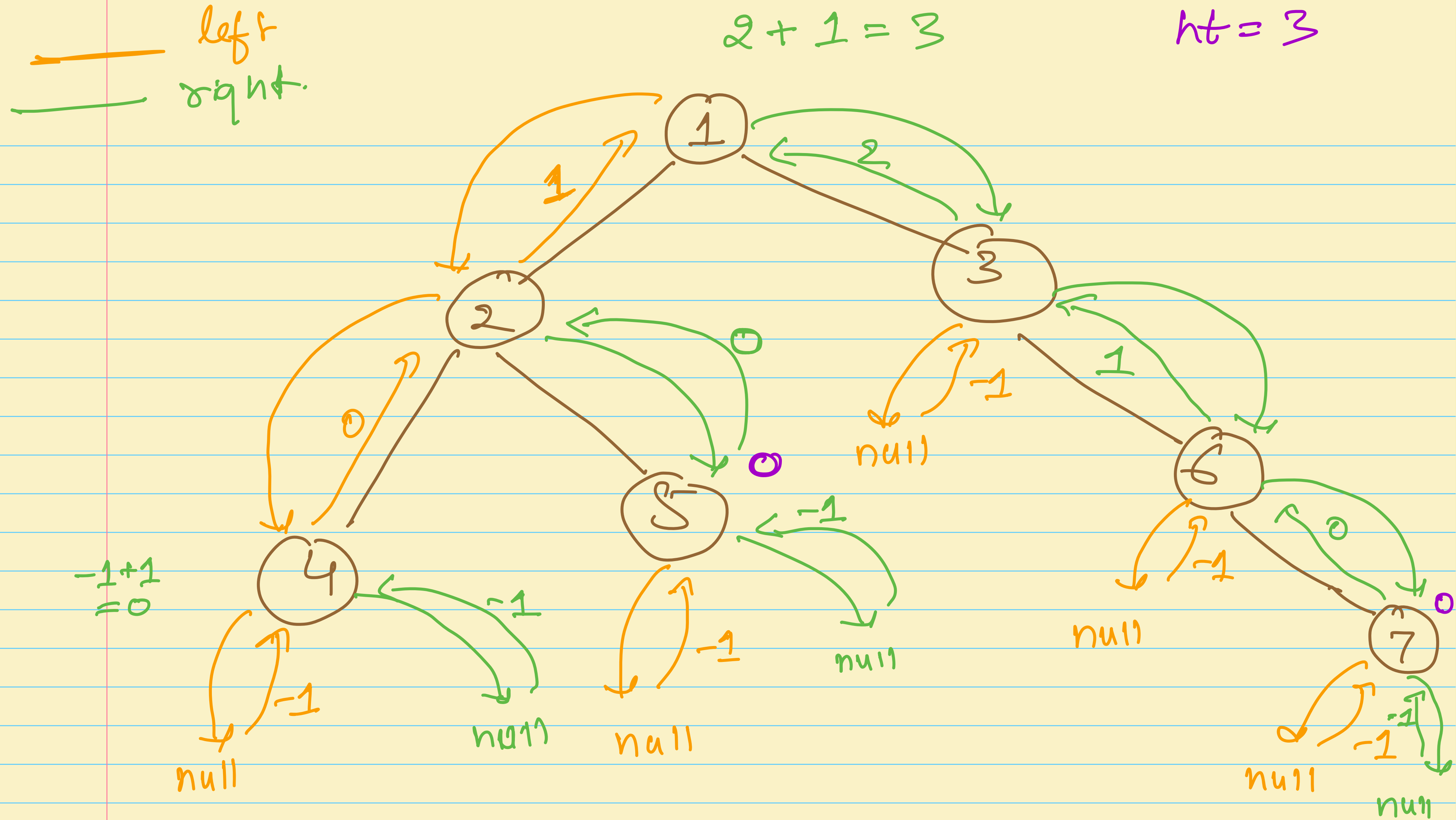
```
int heightOfTree ( root )
{
    if ( root == null) return -1;
    left ht = heightOfTree (root. left)
    right ht = heightOfTree (root. righ.)
    return max ( left ht , right ht ) + 1;
}.
```



Single node ht = 0

Single node height = 0





Ques Check whether binary tree is height balance,

$\boxed{\text{height of lef child} - \text{high of right child} \leq 1}$

H.W

return True.
else return False.