Welcome ☺

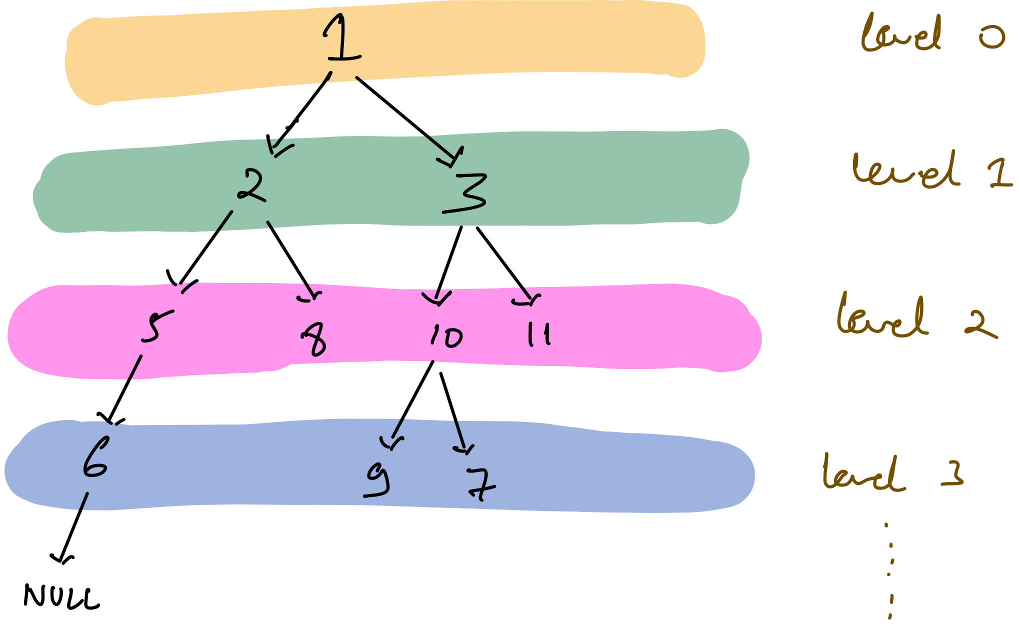
Agenda : Level Order Traversal

Questions Top View | Right View

Vertical Traversal

Type of Binary Trees

Level Order Traversal.



Top - down & left - right

⇒ 1 2 3 5 8 10 11 6 9 7

~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ ~~11~~ ~~6~~ ~~9~~ ~~7~~

1 2 3 5 8 10 11 6 9 7

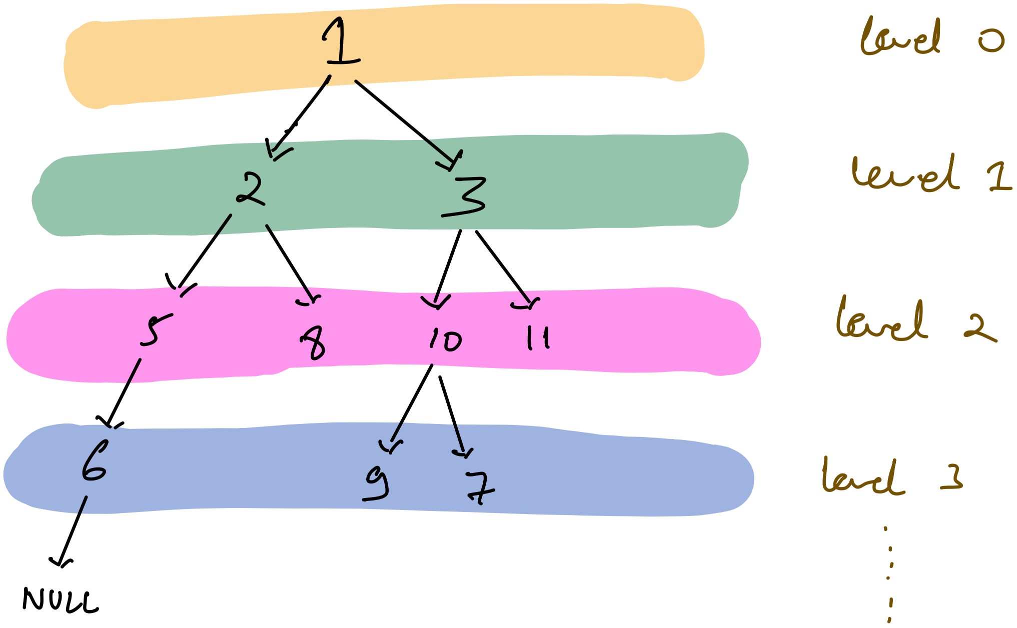
Code

```
q. enqueue (root)
while ( !q. isEmpty() )
{
    n = q. dequeue ()
    print ( n. data)
    if( n. left)   q. enqueue (n.left)
    if( n. right )   q. enqueue ( n.right )
}
```

T.C ⇒ O(N)

SC ⇒ O(N)

Q Print level by level in seperate line.



O/P

1
2 3
5 8 10 11
6 9 7

last → ~~1~~ ~~3~~ ~~11~~ 7

~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ ~~11~~ ~~6~~ 9 7

O/P

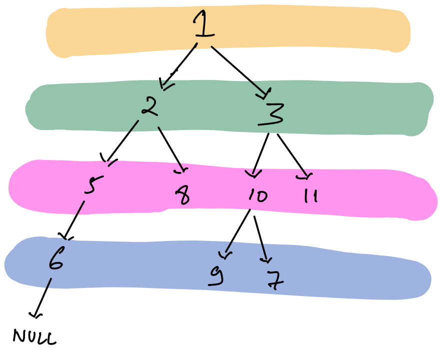
1
2 3
5 8 10 11
6 9 7

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

swap for left view.

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

Q. Print right view of binary tree.



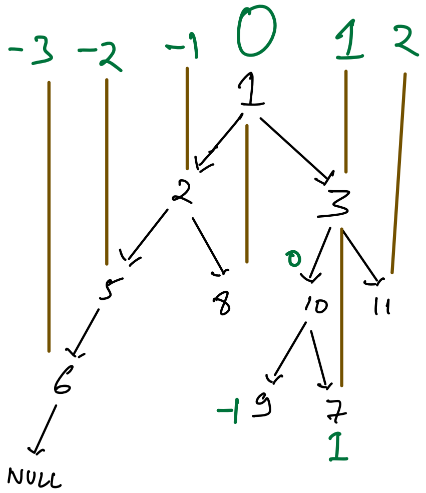
O/p => 1 3 11 7

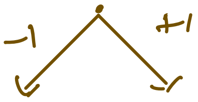
Soln -> Print last node of each level.

```
q. enqueue(root)
last = root
while (!q.isEmpty())
{
    n = q.dequeue()
    if( n.left)  q.enqueue(n.left)
    if(n.right)  q.enqueue(n.right)
    if( n == last )
    {
        print( n.data)
        if(!q.isEmpty())
            last = q.rear()
    }
}
```

T.C ⇒ O(N)
S.C ⇒ O(N)

Print Vertical Order Traversal.

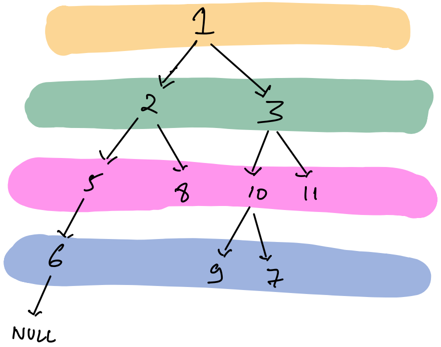




left to Right & Top to Bottom

o/p ⇒ -3 ⇒ 6
-2 ⇒ 5
-1 ⇒ 2 9
0 ⇒ 1 8 10
1 ⇒ 3 7
2 ⇒ 11

1) store vertical level of each node. hashmap.
2) level order traversal.
3) Sort & Iterate hashmap.



Hashmap <level, array<node>>

0 →	1, 8, 10
-1 →	2, 9
1 →	3, 7
-2 →	5
-3 →	6
2 →	11

(node, ds)

~~(1,0)~~ ~~(2,-1)~~ ~~(3,1)~~ ~~(5,-2)~~ ~~(8,0)~~ ~~(10,0)~~ ~~(11,2)~~

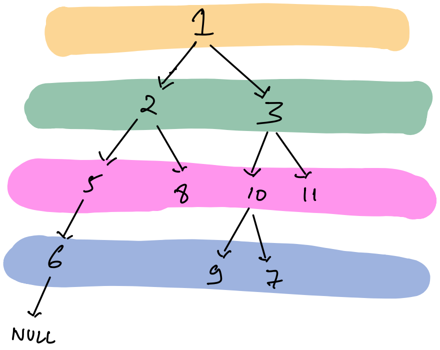
~~(6,-3)~~ ~~(9,-1)~~ ~~(7,1)~~

⇨ maintain min & max level

Code

```
hashmap < int , list < Node > >  hm
  queue < Node* , int > q.
  minD = 0        maxD = 0
  q.enqueue ( root, 0)
  while ( ! q.isEmpty () )
  {
      x = q.dequeue ()
      currDis = x.second
      minD = min ( minD , currDis)
      maxD = max ( maxD, currDis)
      hm.insert ( currDis, x.first) ← appending into the List.
      if ( x.left )   q.enqueue ( x.first.left , currDis -1)
      if( x.right )   q.enqueue ( x.first.right, currDis +1)
  }
  // Traverse hashmap from minD to maxD . & print
```

Q Print Top View



o/p ⇒ 6 5 2 1 3 11

Hashmap <level, ~~array~~ <node>>

→ Print first node of every vertical distance.

Types of Binary Tree.

1) Proper Binary Tree

→ every node has either 2 or 0 children
(never 1 child)

2) Complete Binary Tree.

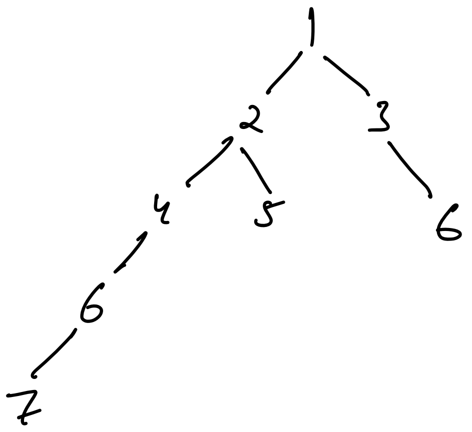
→ all level are completely filled except the last, which is filled from left to right

3) Perfect Binary Tree

→ All levels are completely filled including the last level

Q Check if a tree is height balanced tree.

$\forall_{nodes} \; |height\_leftChild - height\_rightchild| \le 1$ ✓



1) ∀node check if it is height balanced.
2) Recurse for left subtree.
3) Recurse for right subtree.

```
int    height( Node)                     O(N)
{
    if( node == NULL)
            return -1
    return 1+ man (height (node.left) , height(node.right))
}
```

```
bool    isHeightBalanced ( Node)
{
    if( node == NULL)                          T.C => O(N^2)
            return True.                       S.C => O(N)
    heightLeft   =  height ( Node.left)
    heightRight  =  height ( Node.right)
    if( abs( heightLeft - heightRight ) > 1)
            return false.

    return isHeightBalanced(Node.left) && isHeightBalanced(Node.left)
}
```

Code

```
isHB = True.
int    height ( Node )
{
    if( node = = NULL)
            return -1
    hl = height (node. left )
    hr = height (node. right)
    if( abs( hl-hr) > 1)    isHB = False.
    return 1+ max ( hl, hr)
}
return isHB
```

T.C   O(N)

S.C   O(N)