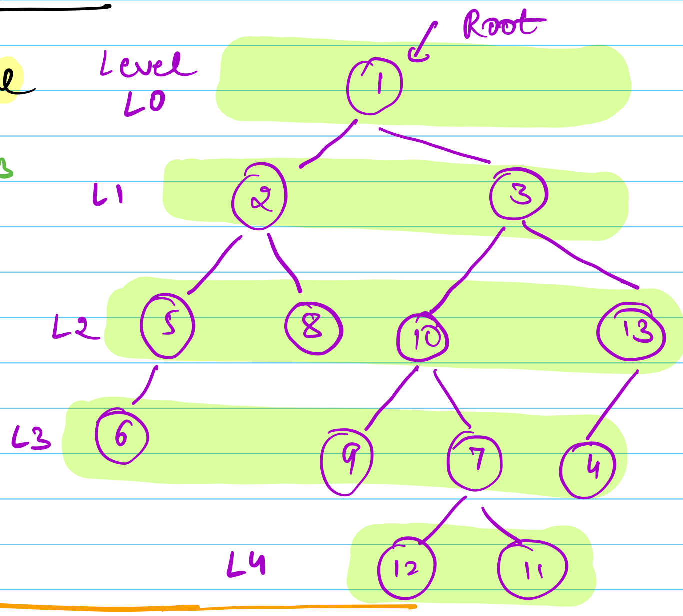
1/8/2023

TREES - 2

Level order traversal

1 2 3 5 8 10 13

6 9 7 4 12 11



Level - by level →
Queue

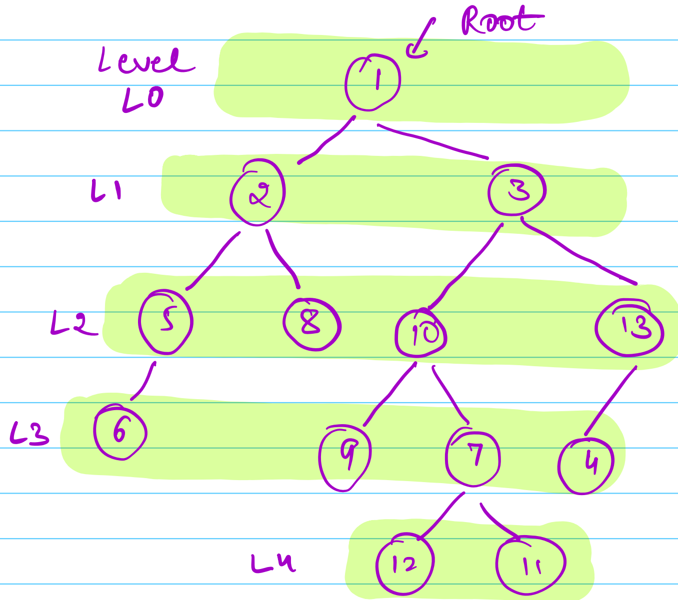
~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ ~~13~~ ~~6~~ ~~9~~ ~~7~~ ~~4~~ ~~12~~ ~~11~~

```
q.enqueue(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);
}
```

TC: O(N)
SC: O(N)

Print levels in separate line

1
2 3
5 8 10 13
6 9 7 4
12 11



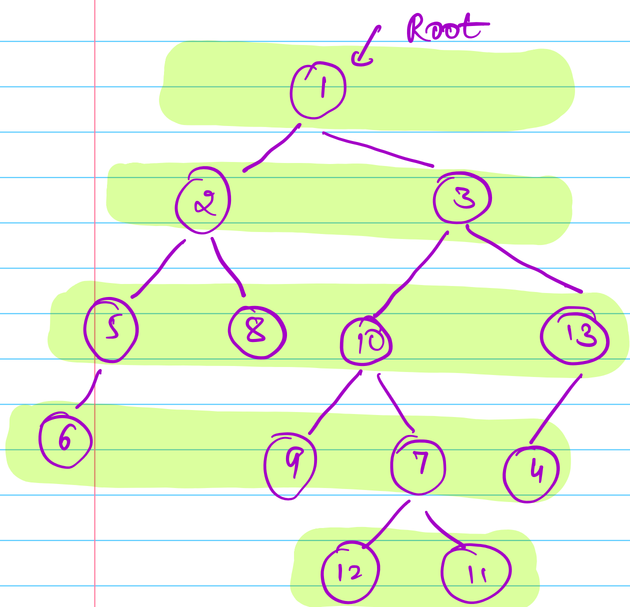
~~1~~ ~~2~~ ~~3~~ ~~5~~ ~~8~~ ~~10~~ ~~13~~ ~~6~~ ~~9~~ ~~7~~ ~~4~~ ~~12~~ ~~11~~

last last last last last

```
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);

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

TC: O(N)
SC: O(N)

Q Print right view of binary tree



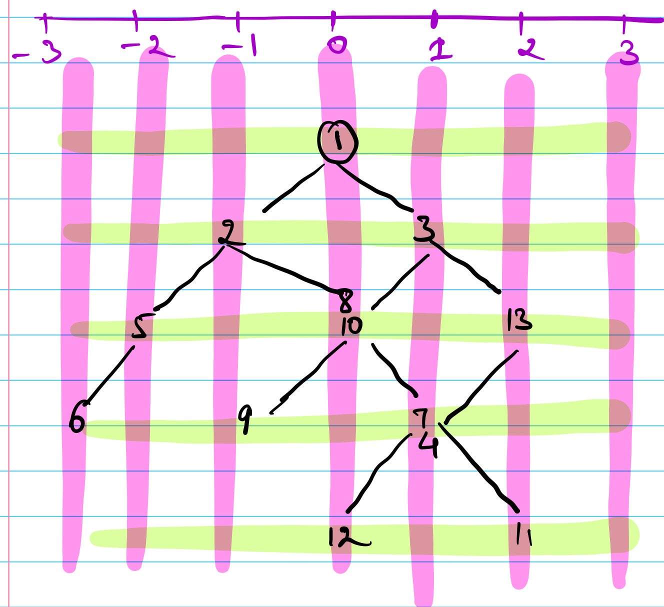
O/P ⇒ 1, 3, 13, 4, 11

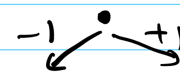
Sol^n ⇒ print last node of every level.

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

HW → Print left view.

Q. Print vertical order traversal.





1.
2. Print top to bottom
3. Overlap → first print data from left & then data from right.

O/P →
-3 → 6
-2 → 5
-1 → 2 9
0 → 1 8 10 12
1 → 3 7 4
2 → 13 11

1> node → vertical distance from root
use → hashmap.

2> Level order traversal.

(node, dist)

~~(1,0)~~ ~~(2,-1)~~, ~~(3,1)~~, ~~(5,-2)~~, ~~(8,0)~~ ~~(10,0)~~ ~~(13,2)~~ ~~(6,-3)~~ ~~(9,-1)~~ ~~(7,1)~~

~~(4,1)~~
~~(12,0)~~
~~(11,2)~~

Key (int) (Value) list
Vertical ⟶ list of values

0 → 1, 8, 10, 12
-1 → 2, 9
1 → 3, 7, 4
-2 → 5
2 → 13, 11

-3 → 6

Top view

```
for i -> minDist to maxDist
    for x in hm.get(i) {
        print(x);
    }
    print("\n");
}
```

TC: O(N)
SC: O(N)

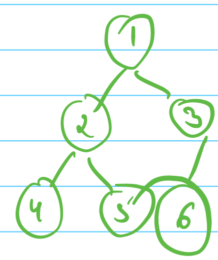
```
x = q.dequeue()   x -> {node, dist}

if (node.left != null) { q.enqueue(node.left, x.dist-1); }
if (node.right != null) { - - - - }

ArrayList = hm.get(dist)
ArrayList.add(node);
```

8:55 am IST

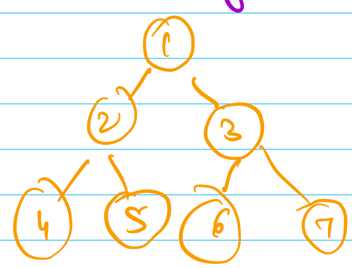
Types of Binary trees

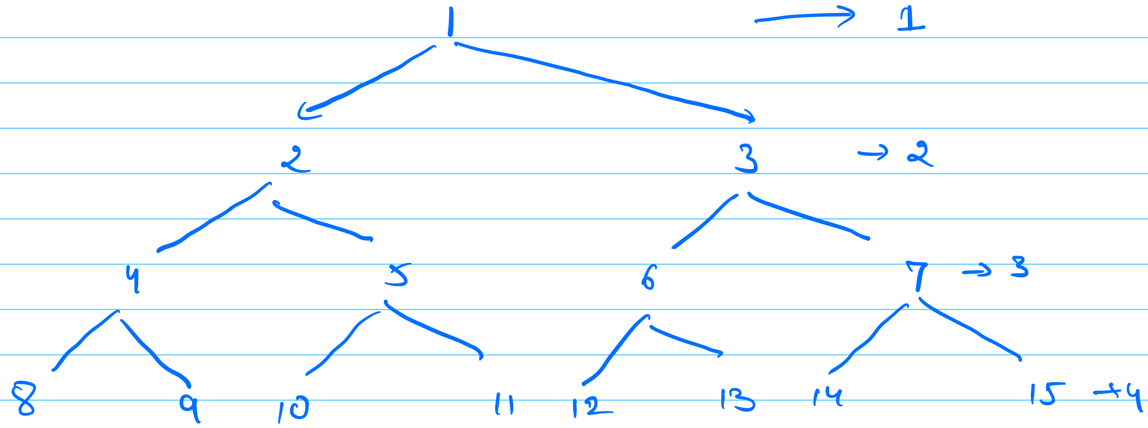
(1) Proper binary tree → Every node has either 0 or 2 children.

(2) Complete binary tree → Every level is complete except may be the last level. All nodes of the last level are as left as possible.



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





Height of perfect binary tree with N nodes

$$N = 1 + 2 + 4 + 8 \cdots\cdots + 2^H$$

$$= \frac{1(2^{H+1} - 1)}{2-1} \Rightarrow (2^{H+1} - 1)$$

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

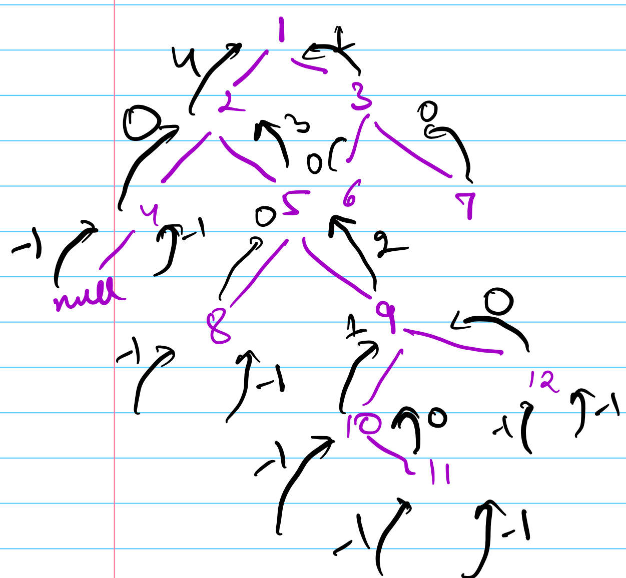
$$2^{H+1} = N+1$$

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

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

Q Check if the given tree is height balanced.

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



null → height = −1   Postorder

boolean isHB = true.

```
int height (root) {
    if (root == null) return -1;

    L = height (root.left)
    R = height (root.right)

    if ( abs|L-R| > 1) isHB = false;

    return max(L, R)+1;
}
```

TC: O(N)
SC: O(H)