"TREES-2"

Level Order Traversal

↳ ① Build a Tree if the input is in Level order

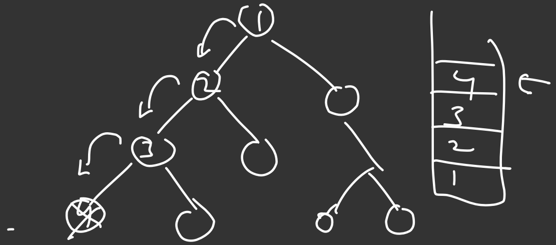
② Print the tree Level By Level.

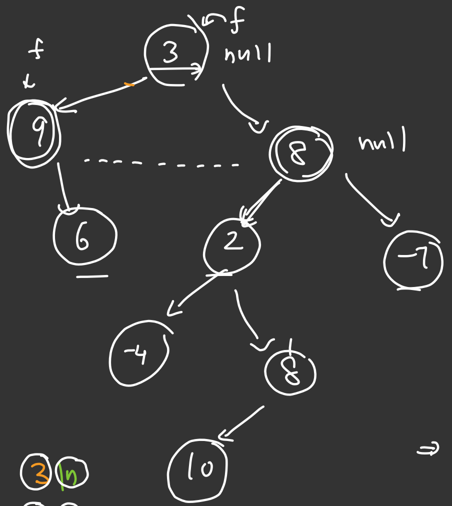
Traversal question

↓

a,ihm ⇑ Recursion

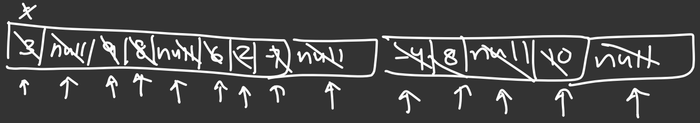
(Code is easy)

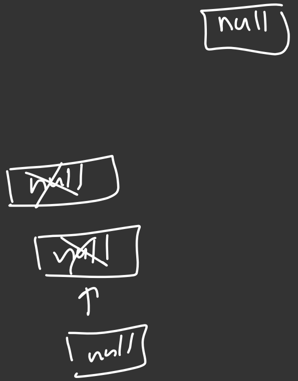


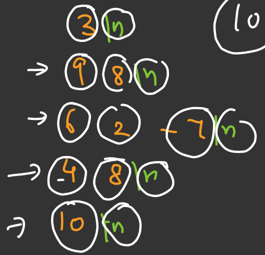


```
Queue <Node> q
q.add(root)
q add(NULL)
while (!q empty())
    [f] = q poll();
    null => if (f == null) {
            print('\n'),
        => if (!q empty())
                { q add(null) };
=> else {
    { print (f data); <-
    if (f left != null)
            q push (f.left)
    if (f.right != null)
            q push (f.right)
    }
}
```

order







Level order

↳ Left to Right

↳ Right to left

↓

Same algo mer order of stmts swapped

```
q.add(f.right)
q.add(f.left)
```

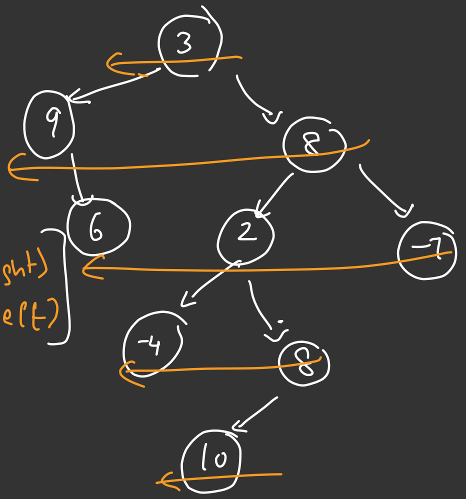
3

8 9

-7 2 6

8 -4

10



3	8	9	-7	2	6	8	-4	10

↑ ↑ ↑ ↑ ↑ ↑ ↑

3

8 9

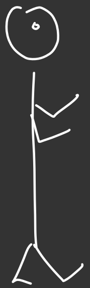
-7 2 6

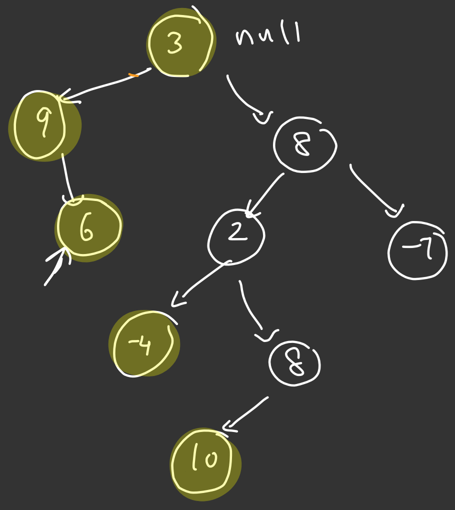
8 -4

10

Views of Tree

① Left View





First Node of every level.

⇓

3
9 8
6 2 -7
-4 8
10

⇒ 3, 9, 6, -4, 10

⇒ Array

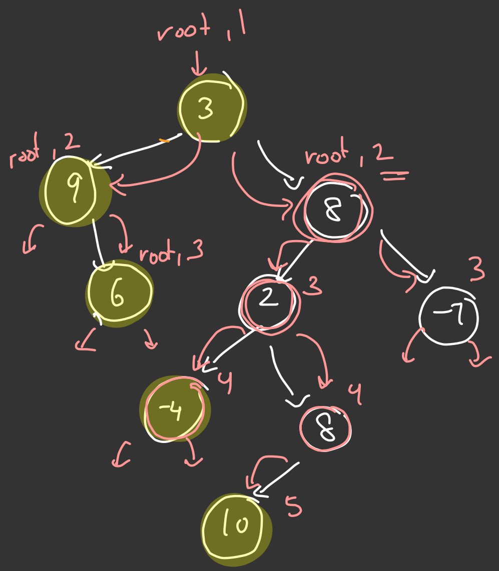
3, null, 9, 8, null, 6, 2, -7, null, -4, 8 null, 10, null

① Print the first element of each levell

② Level order Traversal

↳ if null is before a node, then it will be part of ans.

max level = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5



Algo-2 Rec way

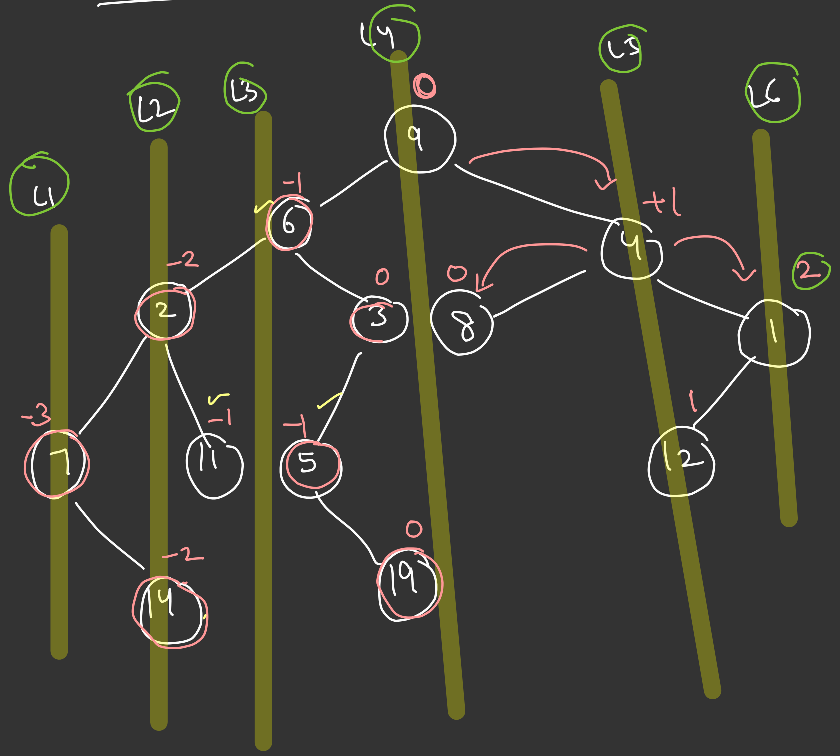
```
class logic {
    static int maxLvl = 0

        printLeftView(Node root, int lvl) {
            if (root == null)
                    return
            if (lvl > maxLvl) {
                    print(root data)
                    maxLvl = lvl
            }
            printLeftView(root left, lvl + 1)
            printLeftView(root.right, lvl + 1)
        }
}
```

3, 9, 6, -4, 10

logic l;

l. printLeftTree(tree.root, 1);

⇒ Vertical Print OF BINARY TREE



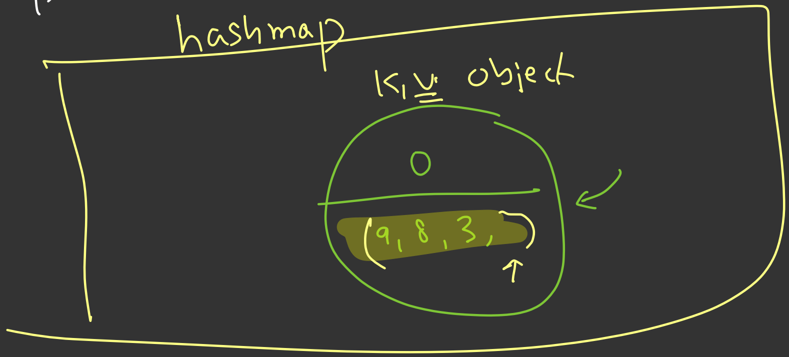
Key dist	value output list ~~of Nodes~~
-3	7
-2	2 14
-1	6 11 5
0	9 3 8 19
1	4 12
2	1

```
hashmap<int, list<int>> hm;
         dist
```

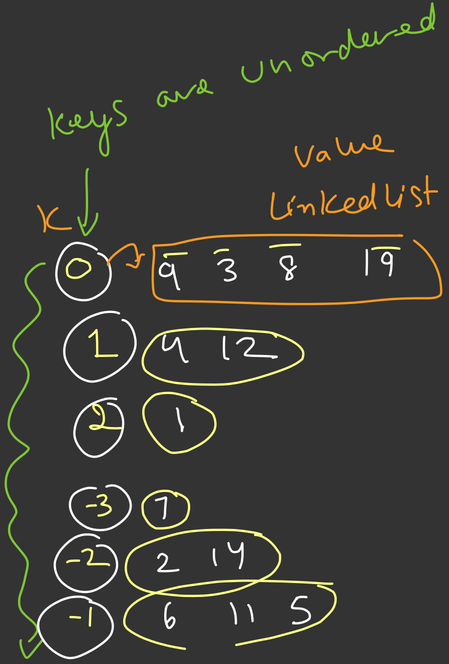
①

O(N)

```
printTree Dist ( Node root, int dist) {
        if(root == null)
              return;
    => hm getORDefault(dist, new LinkedList()).add(root data),

        printTree Dist (root.left, dist - 1);
        printTree Dist (root.right, dist + 1),
}
```



1 → 0
2 → -1
4 → -2
5 → +1



Min Key

Max Key

-3, -2, -1, 0, 1, 2

O(N)

```
② for( every key : hm) {
      minkey = min( — — —
      max key = max( - - - )
```

3

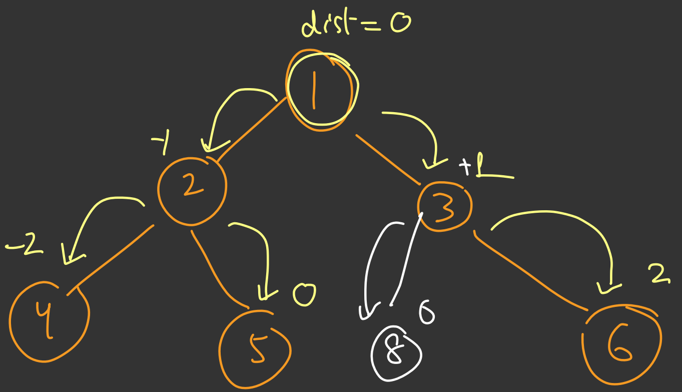
max Key —— min Key

O(N)

```
③ for( Key = minKey —— max key ) {
      LinkedList l = hm.get(key)
      ⇒ for( Int x : list ) {
             print(x);
```

3

-3	7
-2	2 14
-1	6 11 5
0	9 3 8 19
1	4 12
2	1



(int)
Key   Value (list)

0 → (1, 5, 8)
-1 → (2,)
-2 → (4,)
1 → (3)
2 → (6)

Min → Max

-2, -1, 0, 1, 2

4   2   1   3   6
        5
        8

TC → O(N)
SC

[vertical order Print

→ Left view

→

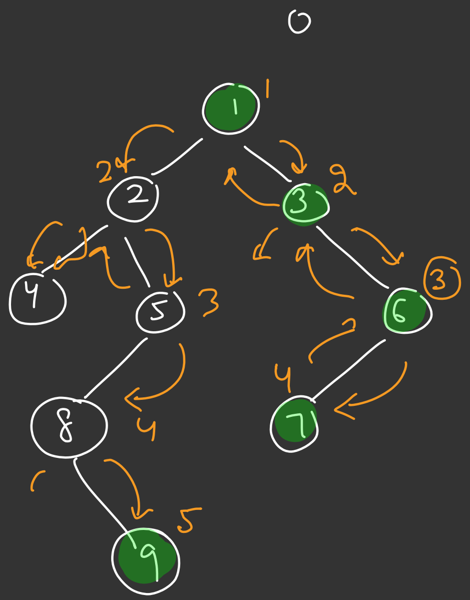
Right View

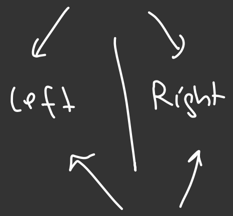
↳ Right

↳ Left

Break :)

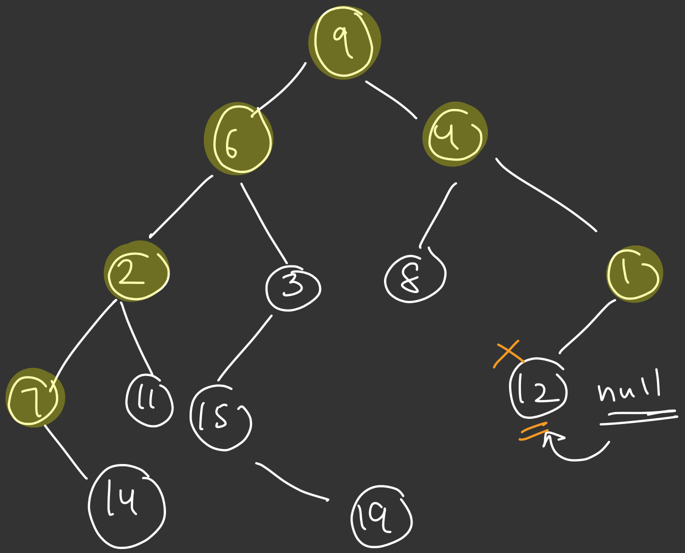
10.30





1, 3, 6, 7, 9

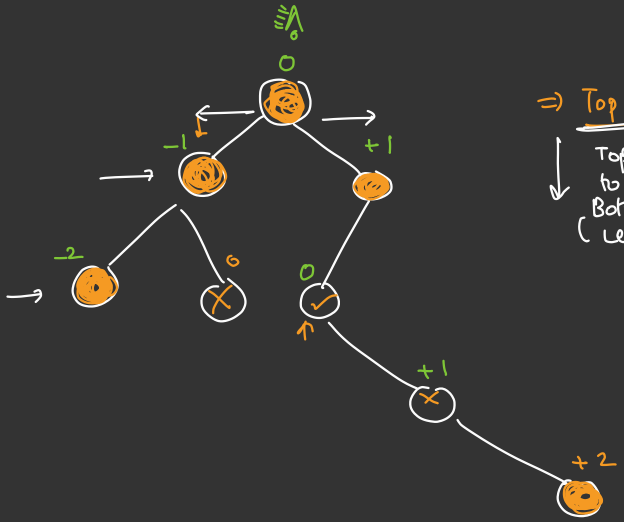
Top View



7 → 2 → 6 → 9 → 4 → 1.

5 mins

10·45



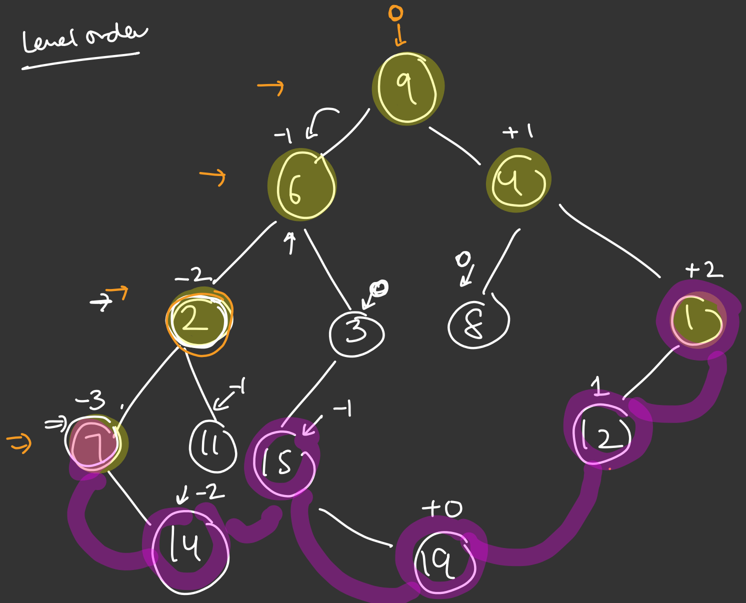
=> Top Most Node for each (vertical) horizontal distance from root

Top to Bottom (Level order Trav)

Hashmap<int, Node>

dist

Level order



Bottom View ⇒ TODO

hashmap — + Level order Traversal

N

0 — 9
-1 ⇒ 6
1 ⇒ 4
-2 ⇒ 2
+2 ⇒ 1
-3 ⇒ 7

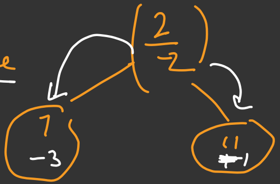
Preorder

~~0 — 9 —~~
~~-1 — 6~~
~~-2 → 2~~
~~-3 → .~~

1 ⇒ X

Level order usig queue

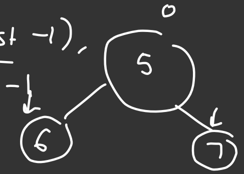
Pair <Node, Dist >



2, -2	- - -	7, -3	11, 1

```
Queue < Pair >  q  ;
Hashmap < int , Node >  hm ;

q add ( Pair (root, 0) ) ;

while ( !q empty() ){
    Pair f  =  q.poll () ;
    if ( hm not contains f dist ){
         hm[f dist] = f n,
    }

    if ( f.n. left != null ) {
         q.add ( pair ( f.n.left , f.dist -1 ),
    }
    if ( f.n.right != null ) {
         q.add ( pair ( f.n.right ,
                        f.dist +1 ),
    }
}

for ( every key . hm ) (
print ( hm get ( key ) . data ) ;
```

```
class Pair {
    Node n
    int dist
}
```



hm 0 → 5

L 3