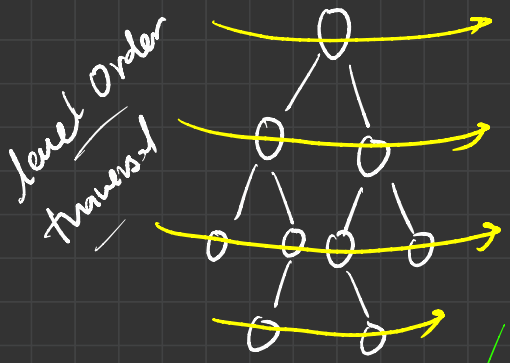
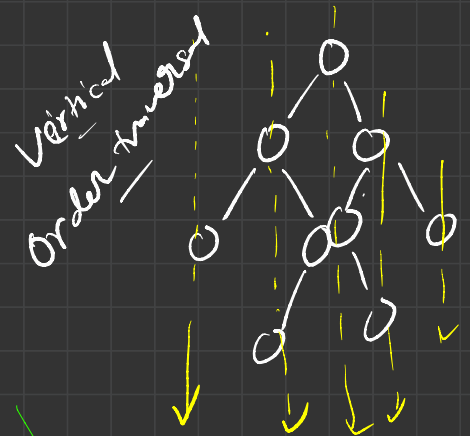
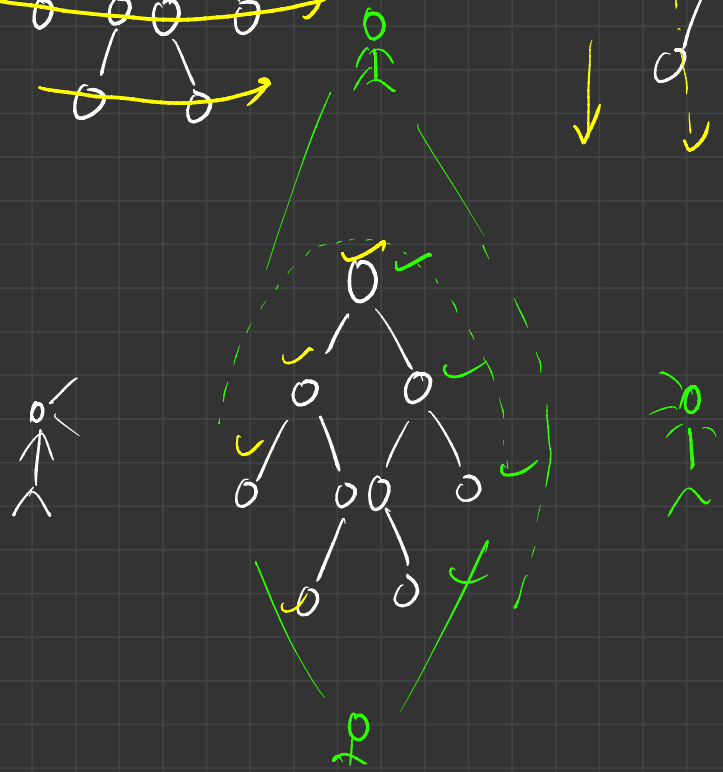
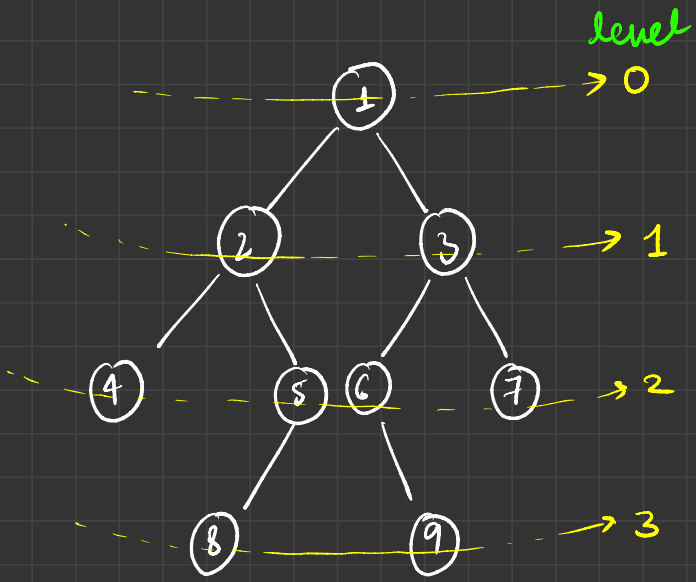

level Order
Traversal

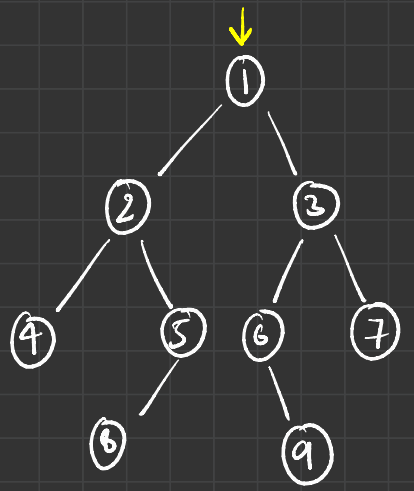

Vertical
Order Traversal

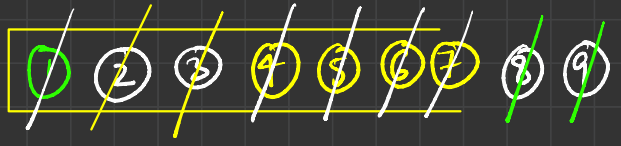


Level Order traversal



1
2 3
4 5 6 7
8 9





que

size : 1 0 1 1 0 4 3 2 1 0 2 1 0

1

2 3

4 5 6 7

8 9

step1: add root in que.

step2: Check is que. size ?0

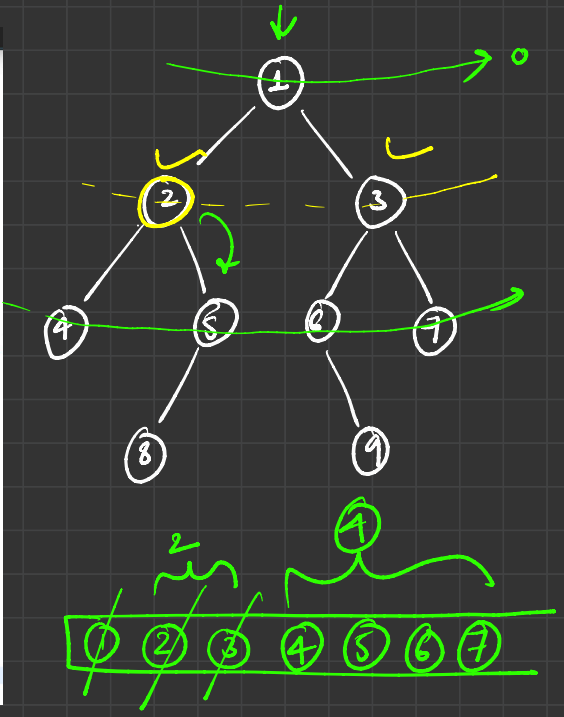
step3: get size of the que

step4: remove front of the que and add childrens.

step5: repeat from step2

this will be done size times, and people removed here will be in same level

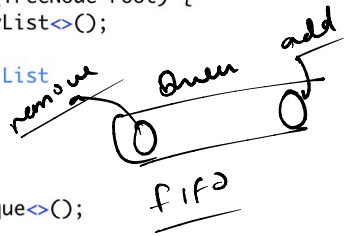
```java
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> lo = new ArrayList<>();

//      if no root, hence return empty List
    if (root == null) {
        return lo;
    }

    Queue<TreeNode> que = new ArrayDeque<>();
//      step 1: add root
    que.add(root);

//      step 2: do your work till que size is greater than zero
    while (que.size() > 0) {
//          step 3: get curr size of the que
        int size = que.size();
        List<Integer> currLevel = new ArrayList<>();

//          step 4: remove size amt of fronts from the que and add there chilren, these size amt of people will lie on same level
        while (size > 0) {
            TreeNode rnode = que.remove();
            currLevel.add(rnode.val);

//              step 5: see if children are there, add them
            if (rnode.left != null) {
                que.add(rnode.left);
            }
            if (rnode.right != null) {
                que.add(rnode.right);
            }

            size--;
        }

        lo.add(currLevel);
    }

    return lo;
}
```

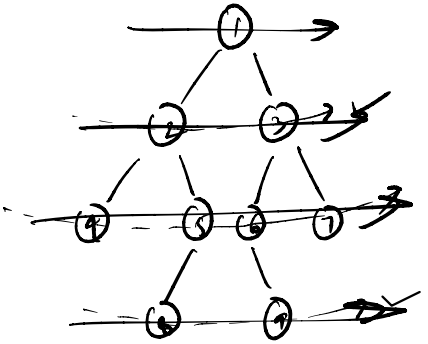


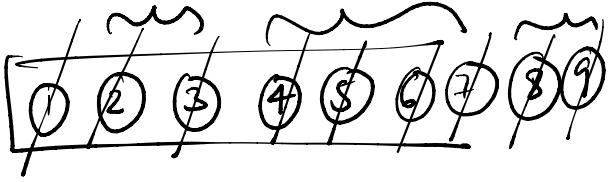
size : 7 10 node - 7 3

{{ 1 }, { 2, 3 }

```java
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> lo = new ArrayList<>();

//          if no root, hence return empty List
        if (root == null) {
            return lo;
        }

        Queue<TreeNode> que = new ArrayDeque<>();
//          step 1: add root
        que.add(root);

//          step 2: do your work till que size is greater than zero
        while (que.size() > 0) {
//              step 3: get curr size of the que
            int size = que.size();
            List<Integer> currLevel = new ArrayList<>();

//              step 4: remove size amt of fronts from the que and add there chilren, these size amt of people will lie on same
            while (size > 0) {
                TreeNode rnode = que.remove();
                currLevel.add(rnode.val);

//                  step 5: see if children are there, add them
                if (rnode.left != null) {
                    que.add(rnode.left);
                }
                if (rnode.right != null) {
                    que.add(rnode.right);
                }

                size--;
            }

            lo.add(currLevel);
        }

        return lo;
    }
```

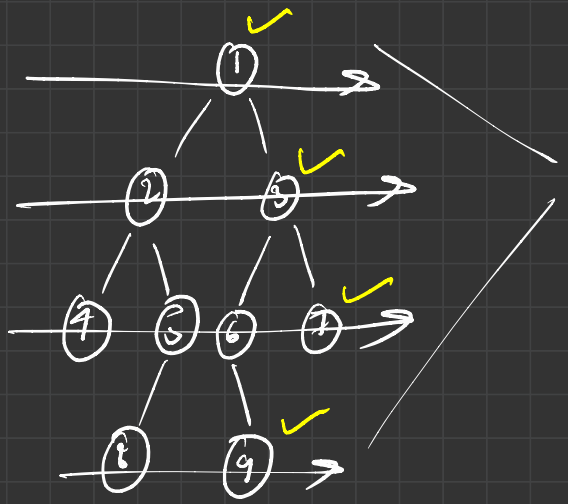






Size = 2 4 0

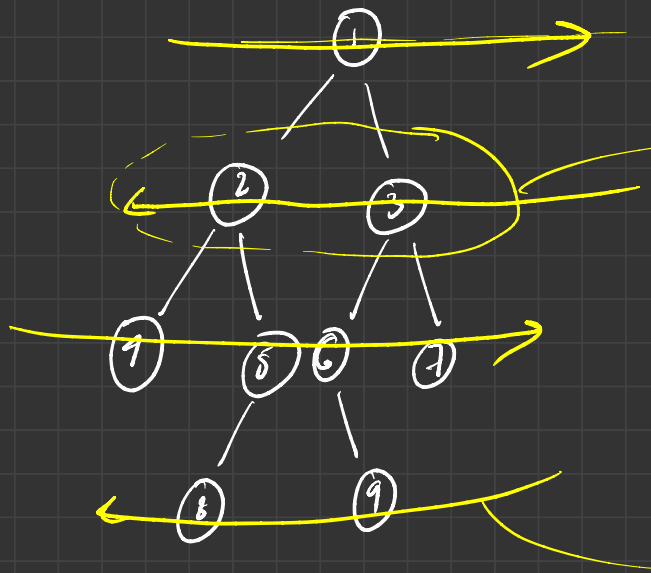
{1} {2,3}
{4,5,6,7}
{8,9}



{ 1, 3, 7, 9 }

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

Zigzag traversal



0

1 → {2, 3}

↳ Reverse {3, 2}

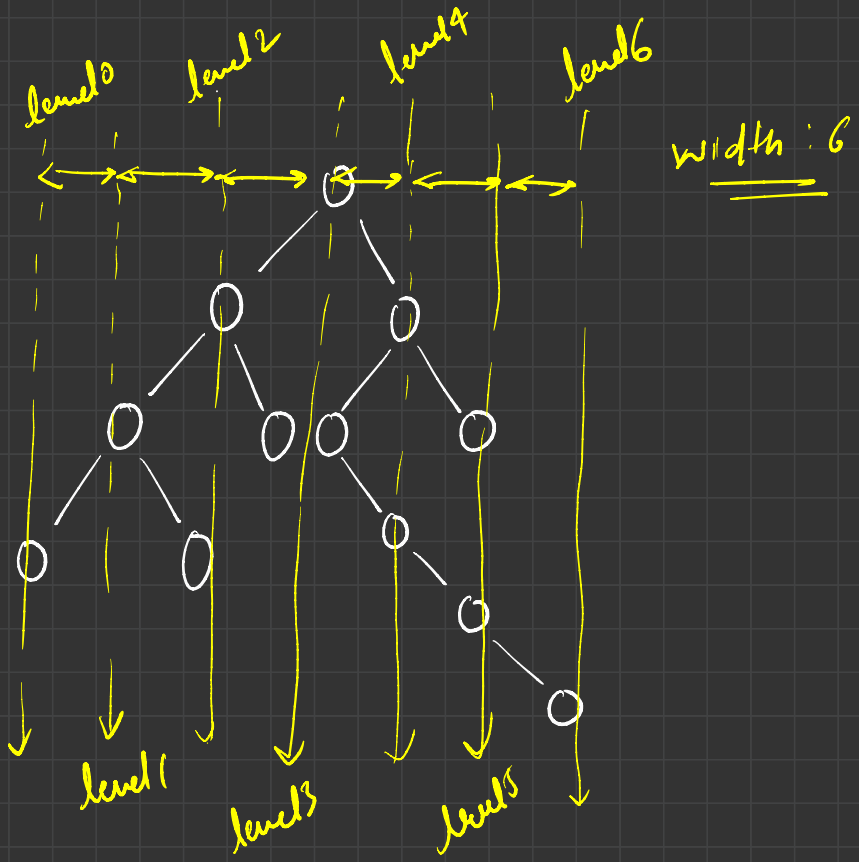
2

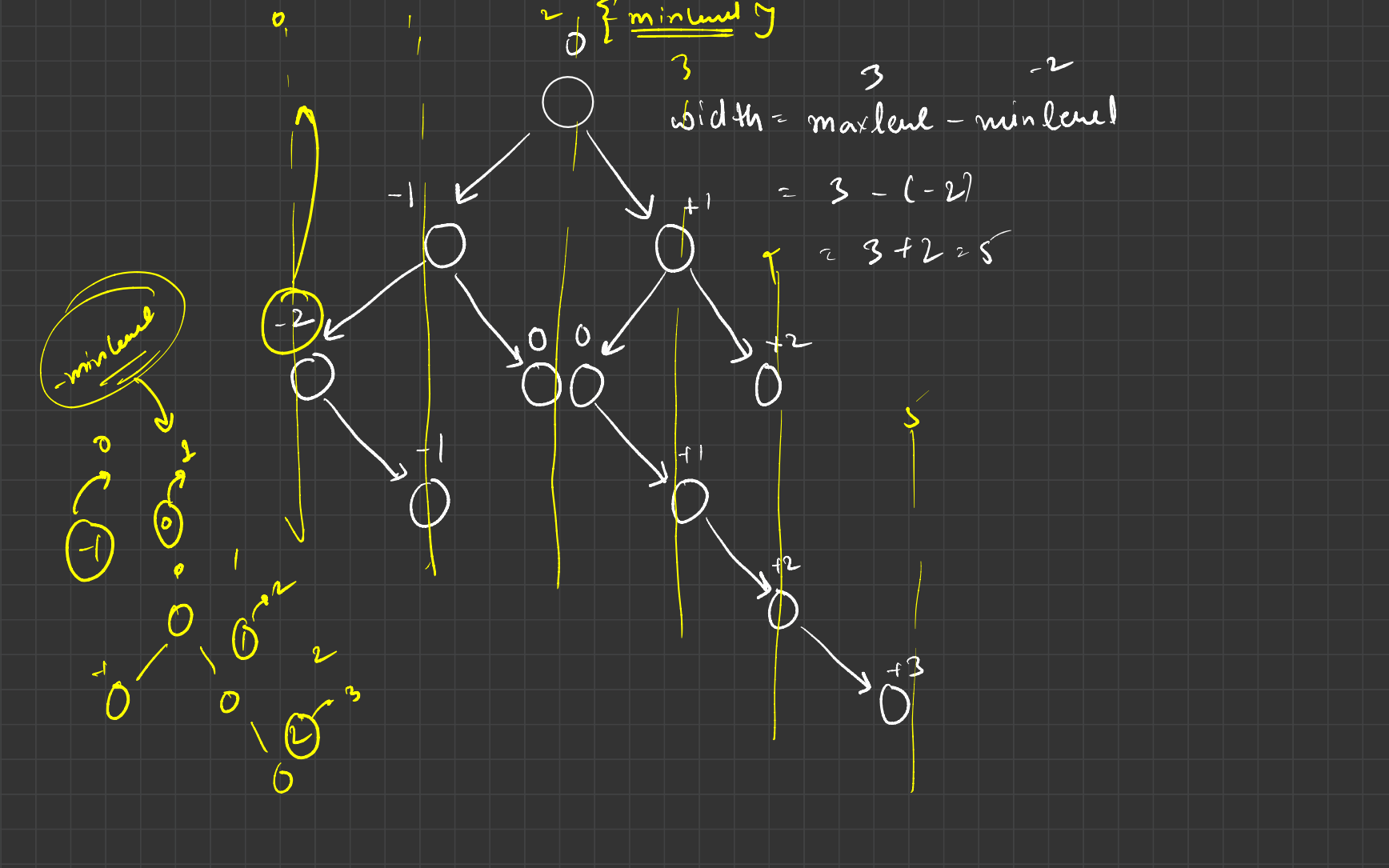
3 → {6, 9} → Reverse {9, 6}

Even level: Left to Right

Odd level: Right to Left

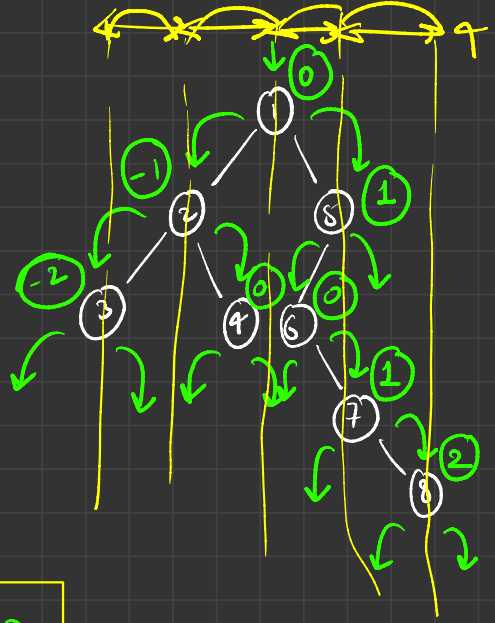
Width of the BT



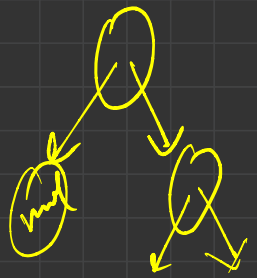

minLevel
3
-2
width = maxLevel - minLevel
= 3 - (-2)
= 3 + 2 = 5
-minLevel

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

```
int minLevel = 0;
int maxLevel = 0;

public void verticalLevels (TreeNode root, int level) {
  if (root == null) {
        return;
    }

   verticalLevels(root.left, level - 1);
   verticalLevels(root.right, level + 1);

   minLevel = Math.min(level, minLevel);
    maxLevel = Math.max(level, maxLevel);
}
```



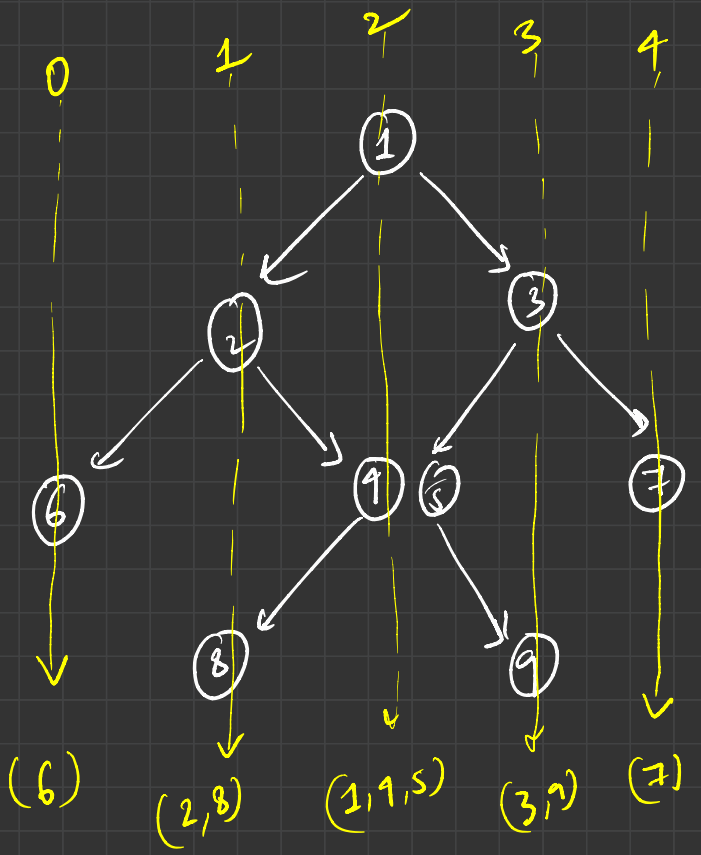
TreeNode
Left
Right



minLevel = 0 - 2
maxLevel = 0 2

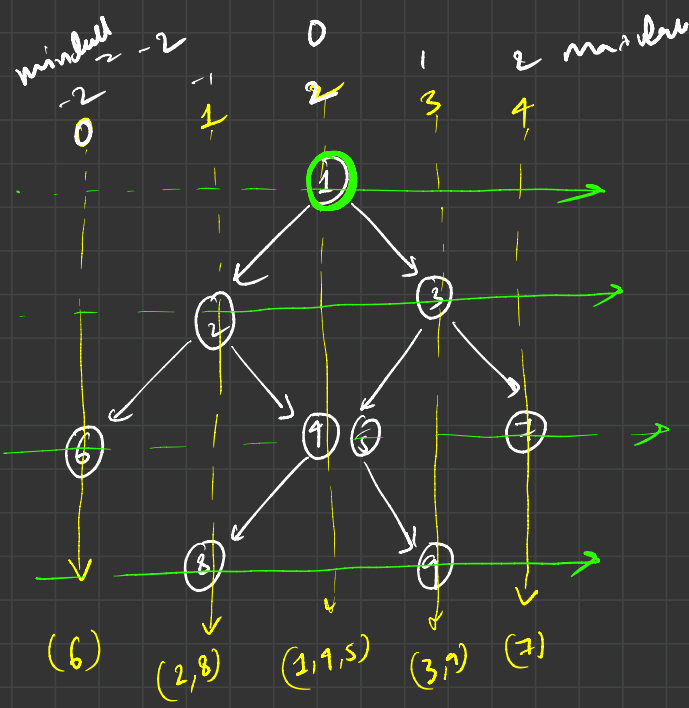
width = 2 - (-2) = 2 + 2 = 4

Vertical Order Traversal



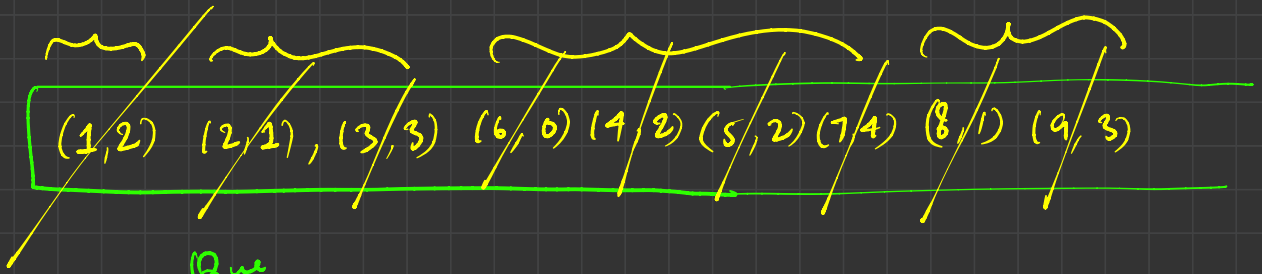
[No. of levels = width of the tree + 1]

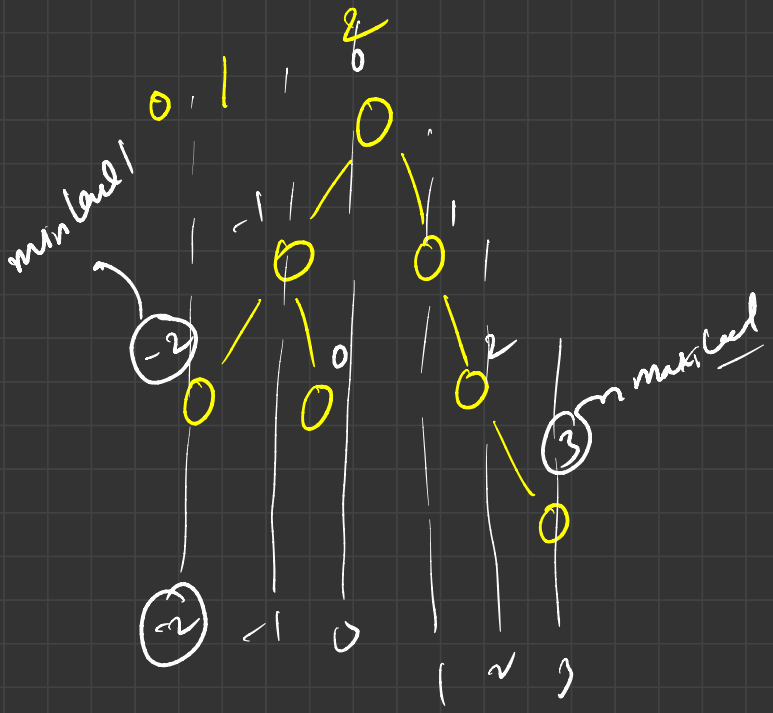
{ 6
2, 8
1, 9, 5
3, 9
7

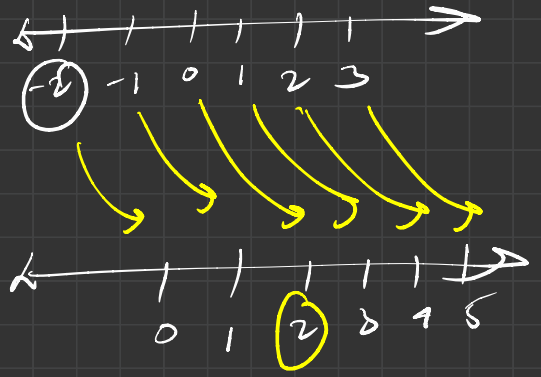


VL	
0	6
1	2, 8
2	1, 4, 5
3	3, 9
4	7

```
class pair {
    Node
    level
};
```







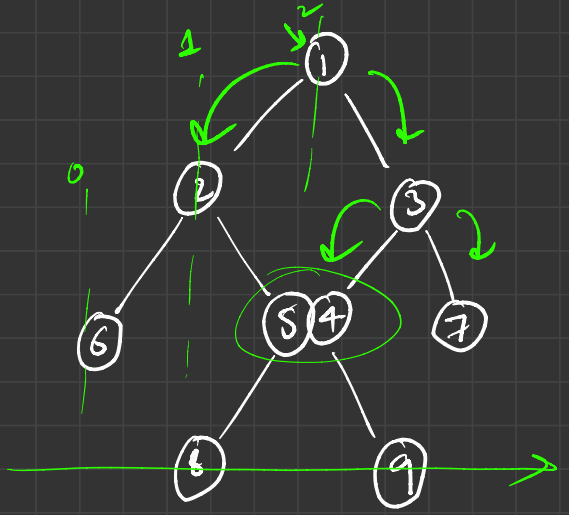
{- minLevel }

{- (-2) = 2 }

Priority Queue

- Acts as a normal queue only
- But it can manage it's value or importance, using a particular priority.

```java
public List<List<Integer>> verticalTraversal(TreeNode root) {
    verticalLevels (root, 0);
    int width = maxLevel - minLevel;
    int numberOfLevels = width + 1;

    List<List<Integer>> vo = new ArrayList<>();
    for (int i = 0; i < numberOfLevels; i++) {
        vo.add(new ArrayList<>());
    }

    PriorityQueue<Pair> Ppq = new PriorityQueue<>();
    pq.add(new Pair(root, -minLevel));

    while (Ppq.size() > 0) {
        int size = Ppq.size();

        PriorityQueue<Pair> Cpq = new PriorityQueue<>();
        while (size > 0) {
            Pair rpair = Ppq.remove();

            vo.get(rpair.vLevel).add(rpair.node.val);

            if (rpair.node.left != null) {
                Cpq.add(new Pair(rpair.node.left, rpair.vLevel - 1));
            }

            if (rpair.node.right != null) {
                Cpq.add(new Pair(rpair.node.right, rpair.vLevel + 1));
            }

            size--;
        }

        Ppq = Cpq;
    }

    return vo;
}
```



ppq

size =

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

Vo

Priority : Same level → give smaller val

diff level → give smaller level

Top View