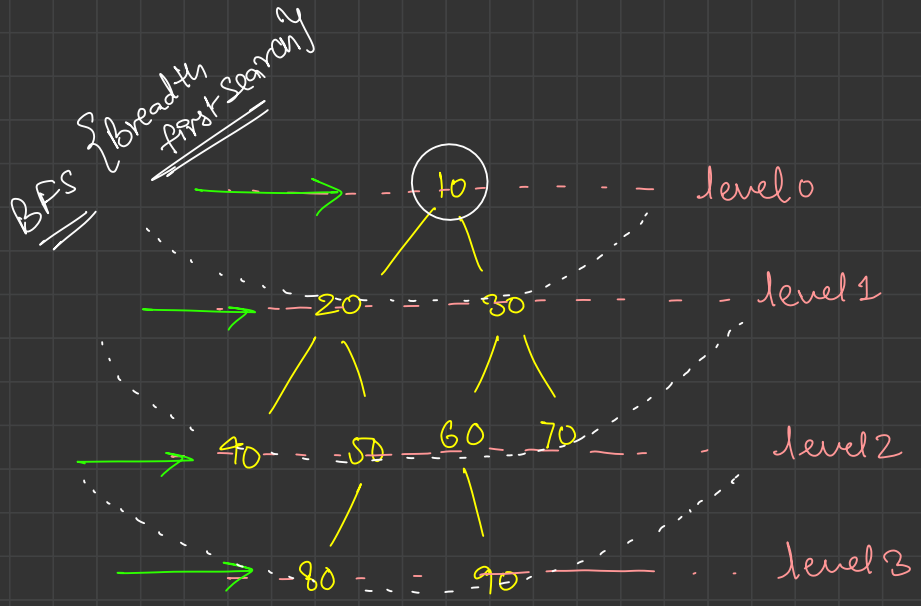
# Traversal on Binary Trees

1. Pre
2. In
3. Post

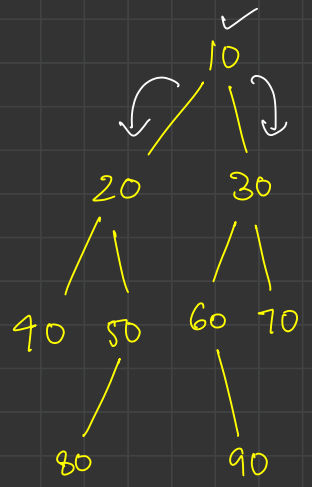
4. level Order traversal

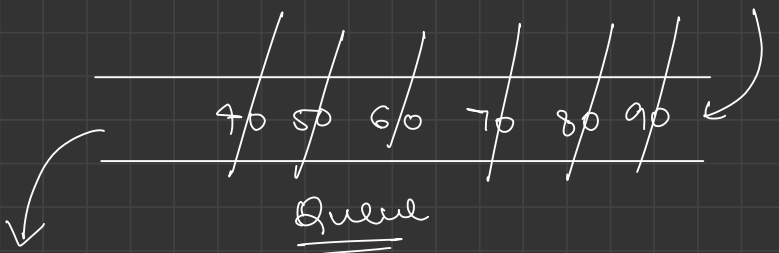
# Level Order Traversal




o/p

10
20 30
40 50 60 70
80 90




size = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

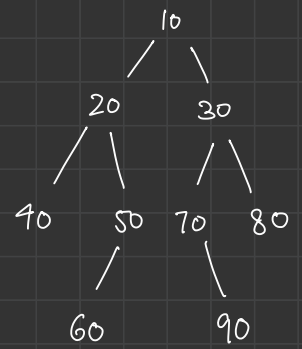
level = 0 1 2 3

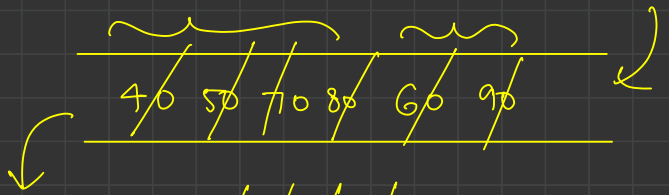
10
20 30
40 50 60 70
80 90

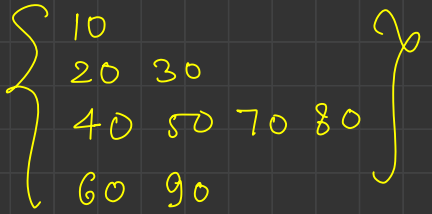
```
Queue<TreeNode> que = new ArrayDeque();
que.add(root);
level = 0;
while (que.size() >= 0)
{
    int size = que.size();
    while (size-- > 0)
    {
        TreeNode node = que.remove();
        print(node.val);
        if (node.left != null)
            que.add(node.left)
        if (node.right != null)
            que.add(node.right);
```

```
        }
        level++;
    }
```

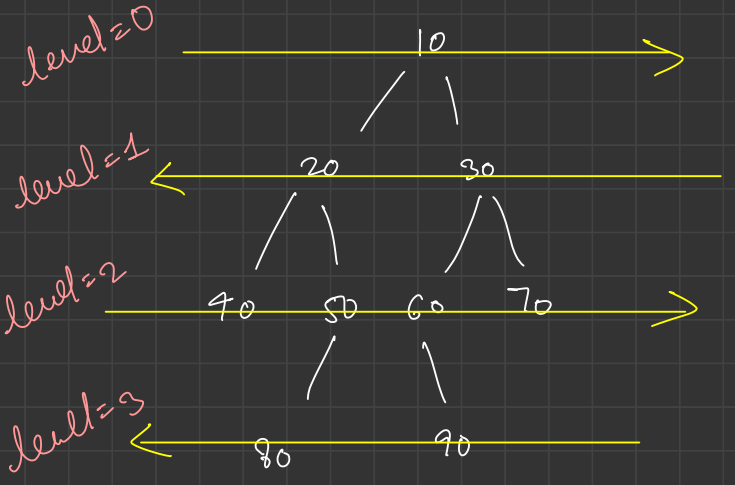
```java
public List<List<Integer>> levelOrder(TreeNode root) {
    // step 1: create a que
    Queue<TreeNode> que = new ArrayDeque<>();

    // step 2: add root to que
    que.add(root);

    // step 3: level of the root is zero
    int level = 0;

    // step 4: run till you have people in que
    while (que.size() > 0) {

        // step 5: get number of people at same level
        int size = que.size();

        System.out.print("level " + level + " -> ");

        // step 6: do. work for same level people together, and add there childs also
        while (size-->0) {

            // step 7: remove the front node of the que
            TreeNode rnode = que.remove();

            // print it
            System.out.print(rnode.val + " ");

            // step 8: check for left child
            if (rnode.left != null) {
                // add left child to que
                que.add(rnode.left);
            }

            // step 9: chech for right child
            if (rnode.right != null) {
                // add right child to que
                que.add(rnode.right);
            }
        }

        // step 10: increase level
        level++;

        System.out.println();
    }

    return new ArrayList<>();
}
```



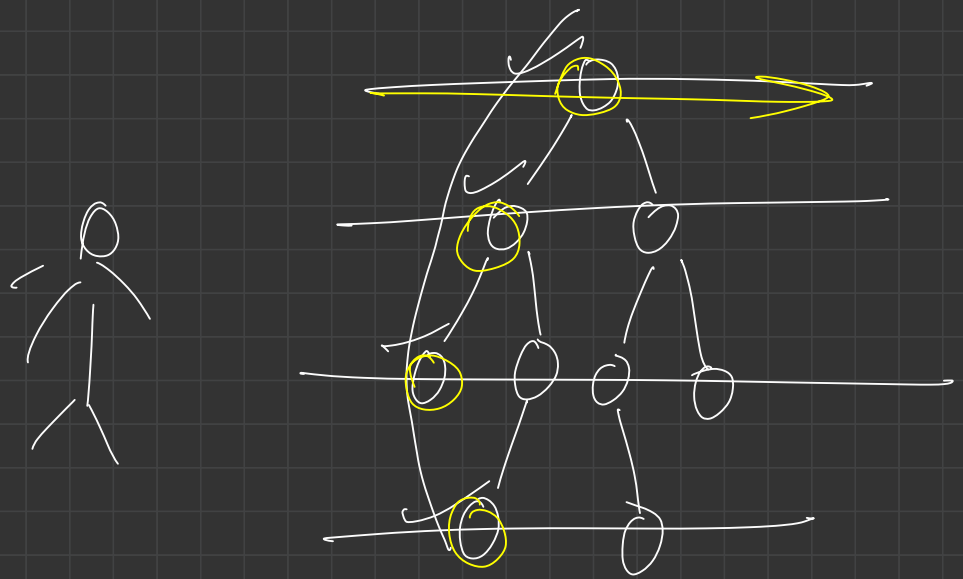
# Zig Zag Traversal




o/p

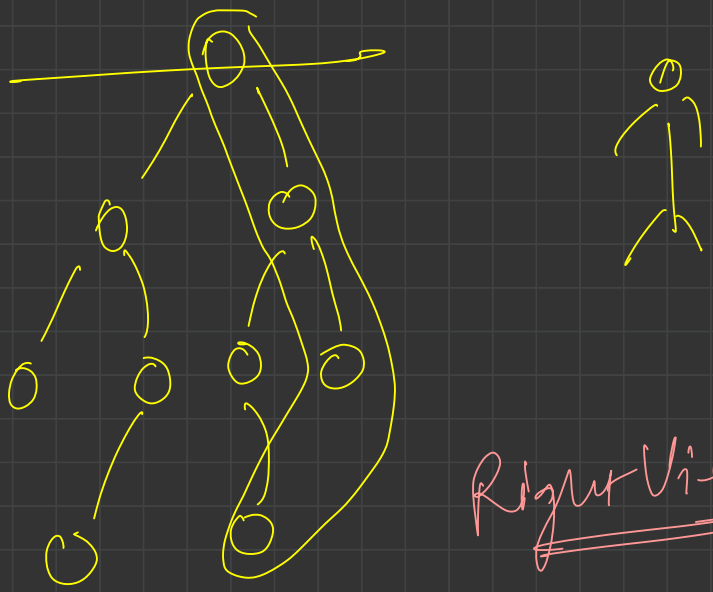
10
30 20
40 50 60 70
90 80

Odd level → reverse



Left View

↳ first Node of Each level !



Right View

last Node of each level