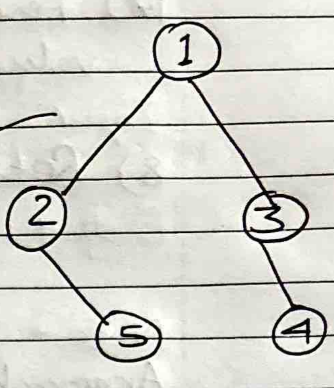
DATE □□□□□□□□

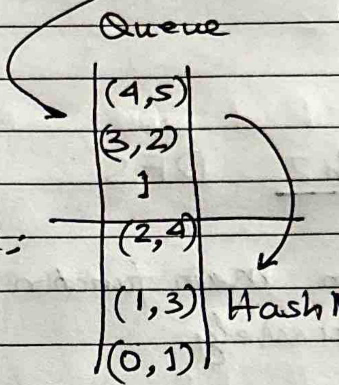
LEETCODE | 199 |

Binary Tree Right Side View



Input : root = [1, 2, 3, null, 5, null, 4]
Output : [1, 3, 4]

Approach 1 BFS



1) class Pair {
        int Level;
        TreeNode node;
    }

2) Queue < Pair> queue = new LinkedList<Pair>();
        queue.offer(new Pair(root, 0));

3) While Loop
        till queue is empty

Also create a HashMap to store levels & right element of the view

4) Pair current = queue.poll();
    TreeNode node = current.node;
    Int level = current.level;

5) If Hashmap doesn't have key then only creat it
    if (!hm.containskey (Level)) {
        hm.put (level, ~~current~~ node.val);
    }

6) Add right of node & then left of node to queue
        if (node.right != null)
            queue.offer(node.right);
        if(node.left != null)
            queue.offer (node.left)

classmate

PAGE □□□

7) Finally iterate over hashmap & add all values to Arraylist

8) Return the Arraylist

Approach 2 DFS

1) From main function call another function recursively.

Recursive fn looks like

```
void dFSTree(TreeNode node, int level, List<Integer> view)
  {
        if (node==null)
                return;

        if (view.size() == level) { // mean array's
              view.add(node.val)}      level'th element is
                                       not present
        }
```

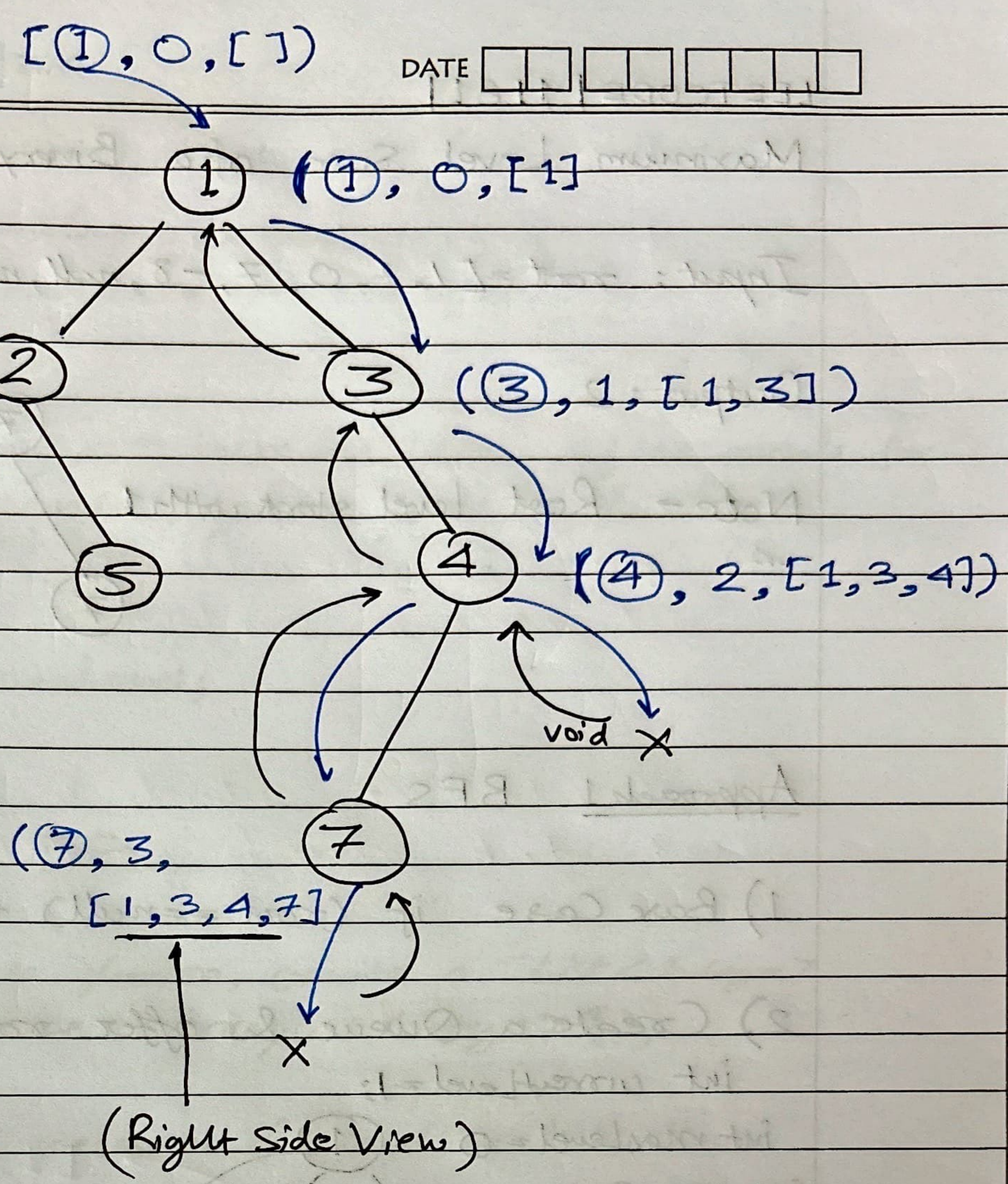
2) This is the primary responsibility of Recursive function.
Leave rest to the recursive calls
(Recursive Leap of FAITH)

```
      dFSTree(node.right, level+1, view);
      dFs Tree ( node.left, level+1, view);
```

Right View is what we want so call node.right first Recursion call & then node.left

3) Return view Arraylist from Step 1 above

Recursion Tree


[(①, 0, [ ])
1
(①, 0, [1]
2
3
(③, 1, [1, 3])
5
4
(④, 2, [1, 3, 4])
void
X
7
(⑦, 3,
[1, 3, 4, 7]
X
(Right Side View)