1. **Minimum time taken to BURN the Binary Tree from a Node**

class Solution {

    public int amountOfTime(TreeNode root, int start) {

        Map<TreeNode, TreeNode> parentMap = new HashMap<>();

        buildParentMap(root, null, parentMap);

        TreeNode startNode = findNode(root, start);

        Queue<TreeNode> q = new LinkedList<>();

        Set<TreeNode> visited = new HashSet<>();

        q.offer(startNode);

        visited.add(startNode);

        int time = -1;

        while (!q.isEmpty()) {

            int size = q.size();

            time++;

            for (int i = 0; i < size; i++) {

                TreeNode node = q.poll();

                if (node.left != null && !visited.contains(node.left)) {

                    visited.add(node.left);

                    q.offer(node.left);

                }

                if (node.right != null && !visited.contains(node.right)) {

                    visited.add(node.right);

                    q.offer(node.right);

                }

                if (parentMap.containsKey(node) && !visited.contains(parentMap.get(node))) {

                    visited.add(parentMap.get(node));

                    q.offer(parentMap.get(node));

                }

            }

        }

        return time;

    }

    private void buildParentMap(TreeNode node, TreeNode parent, Map<TreeNode, TreeNode> parentMap) {

        if (node == null) return;

        if (parent != null) parentMap.put(node, parent);

        buildParentMap(node.left, node, parentMap);

        buildParentMap(node.right, node, parentMap);

    }

    private TreeNode findNode(TreeNode root, int val) {

        if (root == null) return null;

        if (root.val == val) return root;

        TreeNode left = findNode(root.left, val);

        if (left != null) return left;

        return findNode(root.right, val);

    }

}