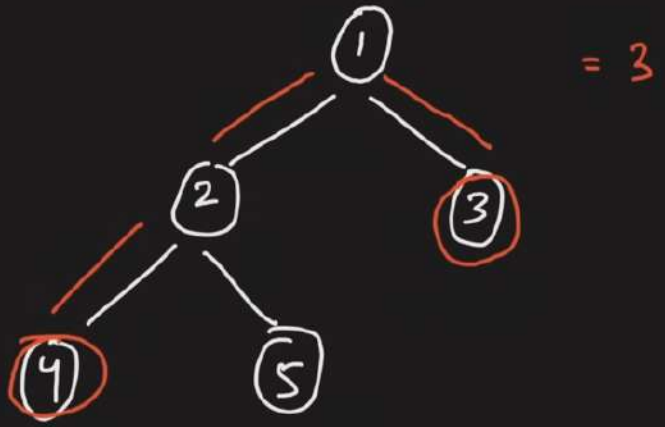
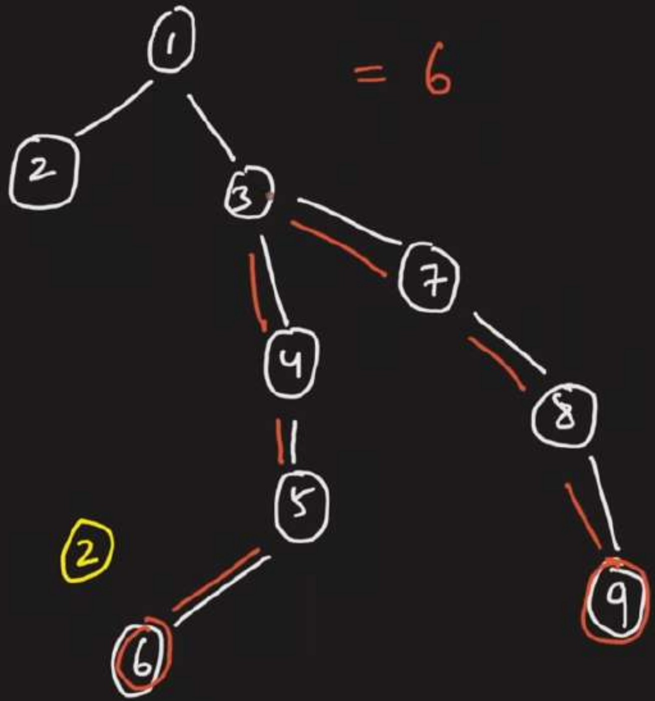
Diameter of a Binary Tree



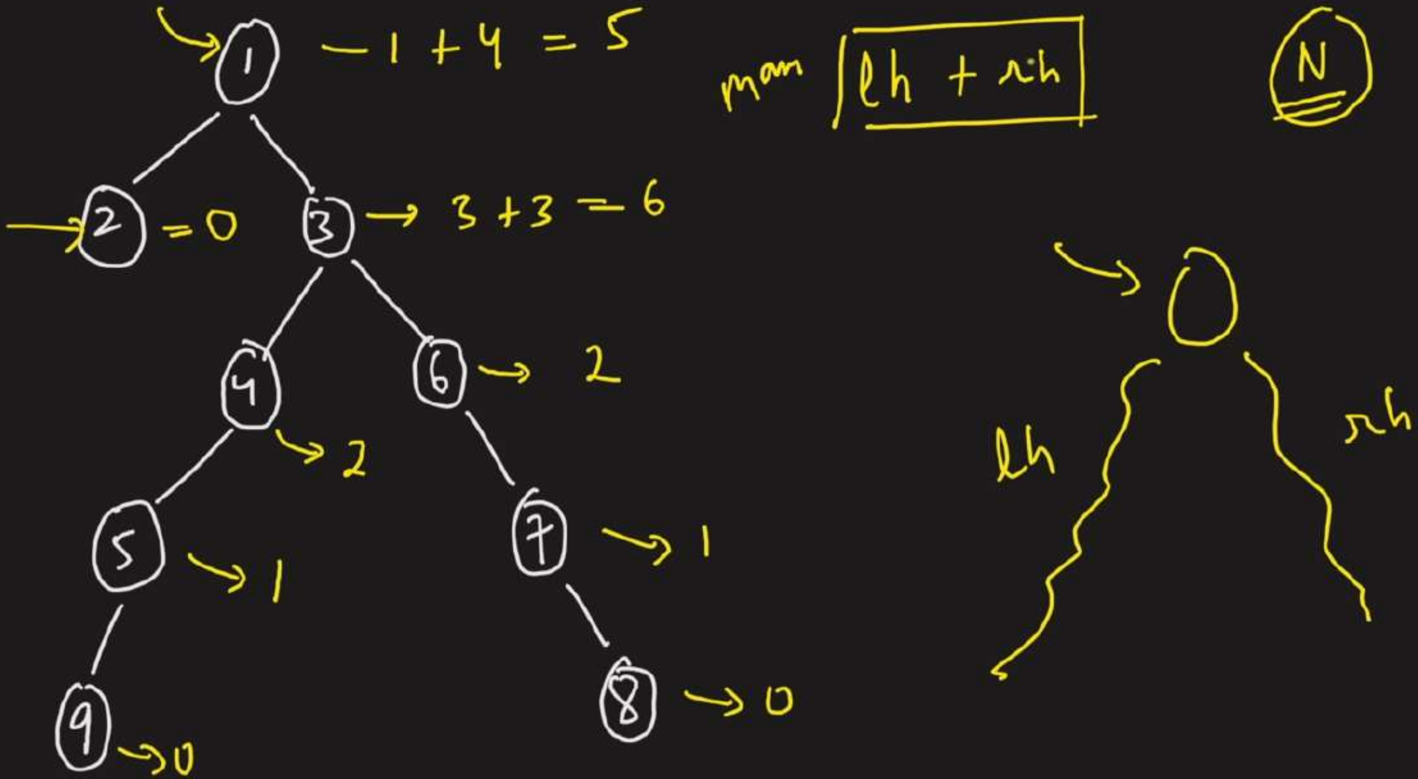
1

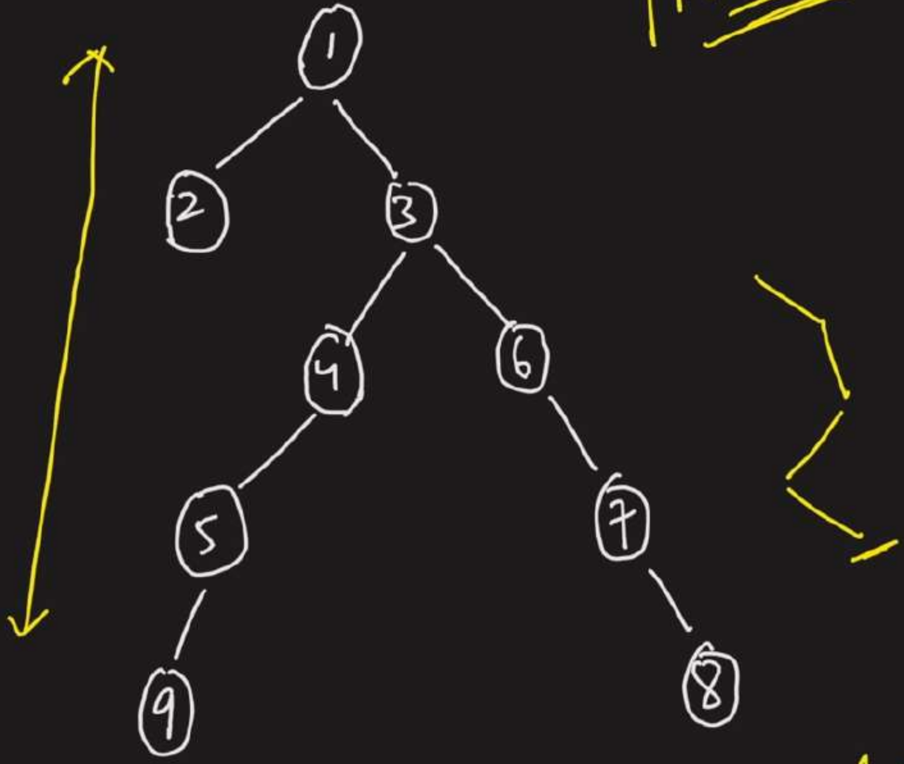
→ longest path bet^n 2 nodes

→ path does not need to pass via root



2


1
−1 + 4 = 5
2
= 0
3
→ 3 + 3 = 6
4
→ 2
6
→ 2
5
→ 1
7
→ 1
9
→ 0
8
→ 0
max
lh + rh
N
lh
rh

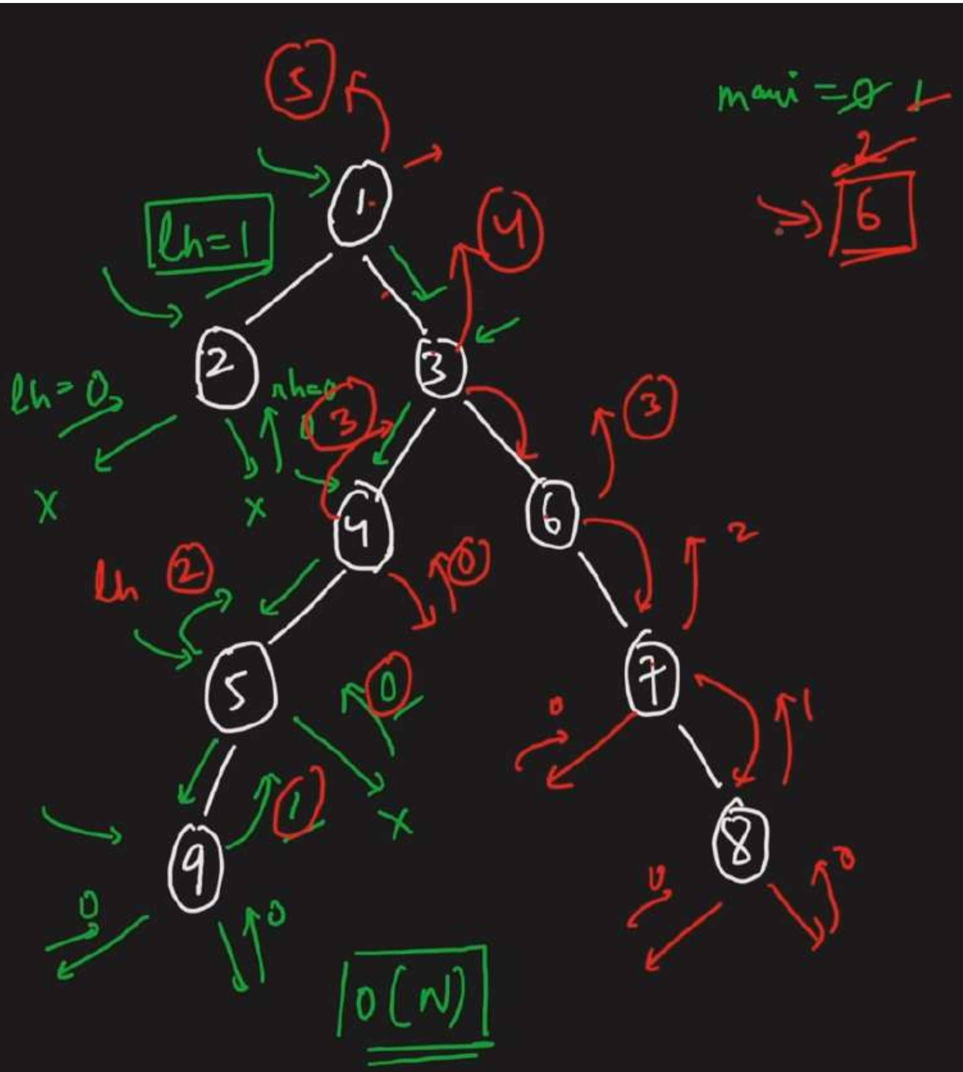


maxi = 0

```
findMax(node)      root
{
    if (root == null)
        return;

    lh = findLeftH(node -> left)
    rh = findRightH(node -> right)
    maxi = max(maxi, lh + rh);
    findMax(node -> left)
    findMax(node -> right)
}
```

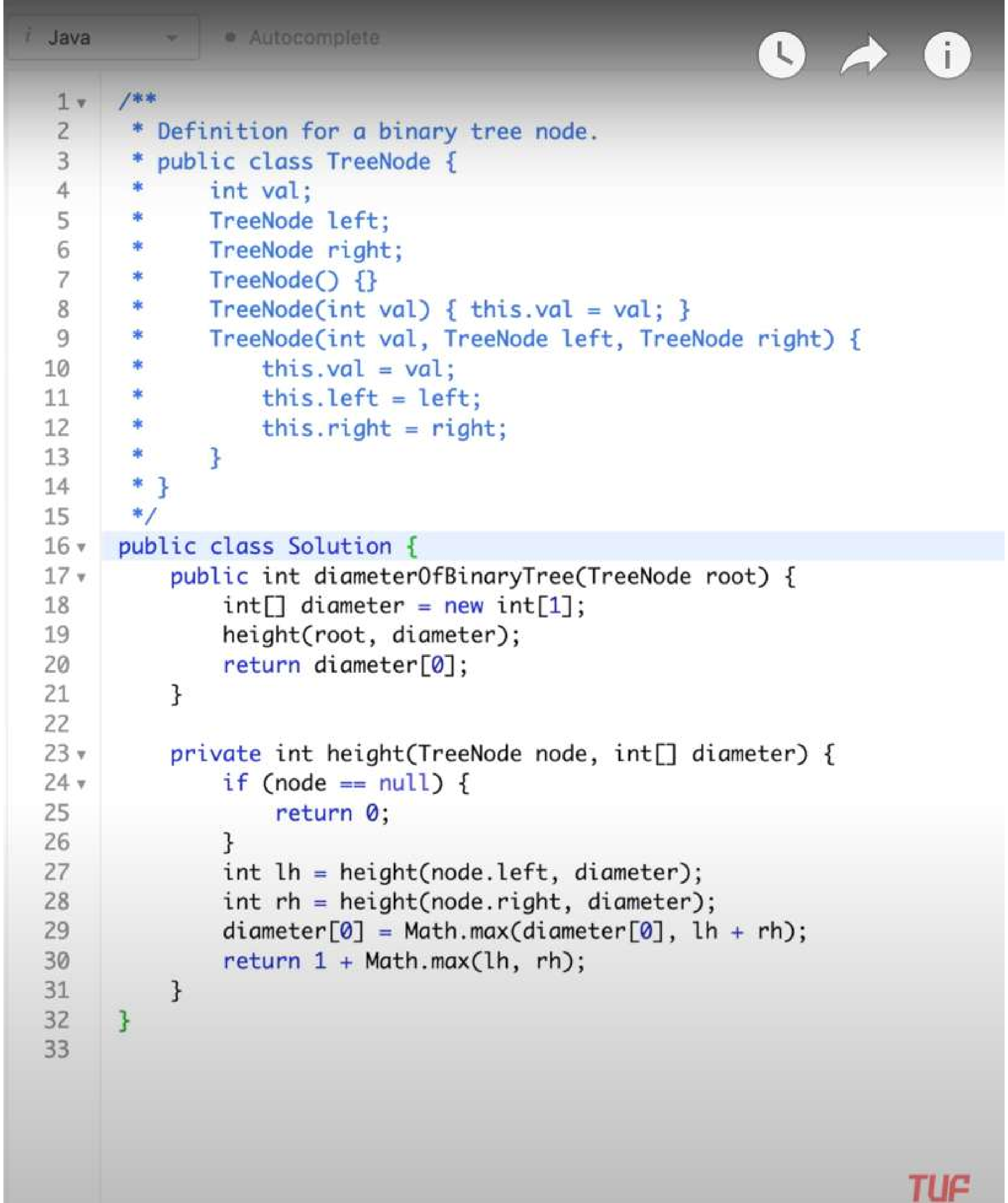
$O(N) \times O(N) = O(N^2)$



```
int findMax(node, maxi)
{
    if (node == null)
        return; 0;

    lh = findMax(node->left, maxi);
    rh = findMax(node->right, maxi);
    maxi = max(maxi, lh+rh);

    return 1 + max(lh, rh);
                   3    3
}
```



```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public int diameterOfBinaryTree(TreeNode root) {
        int[] diameter = new int[1];
        height(root, diameter);
        return diameter[0];
    }

    private int height(TreeNode node, int[] diameter) {
        if (node == null) {
            return 0;
        }
        int lh = height(node.left, diameter);
        int rh = height(node.right, diameter);
        diameter[0] = Math.max(diameter[0], lh + rh);
        return 1 + Math.max(lh, rh);
    }
}
```