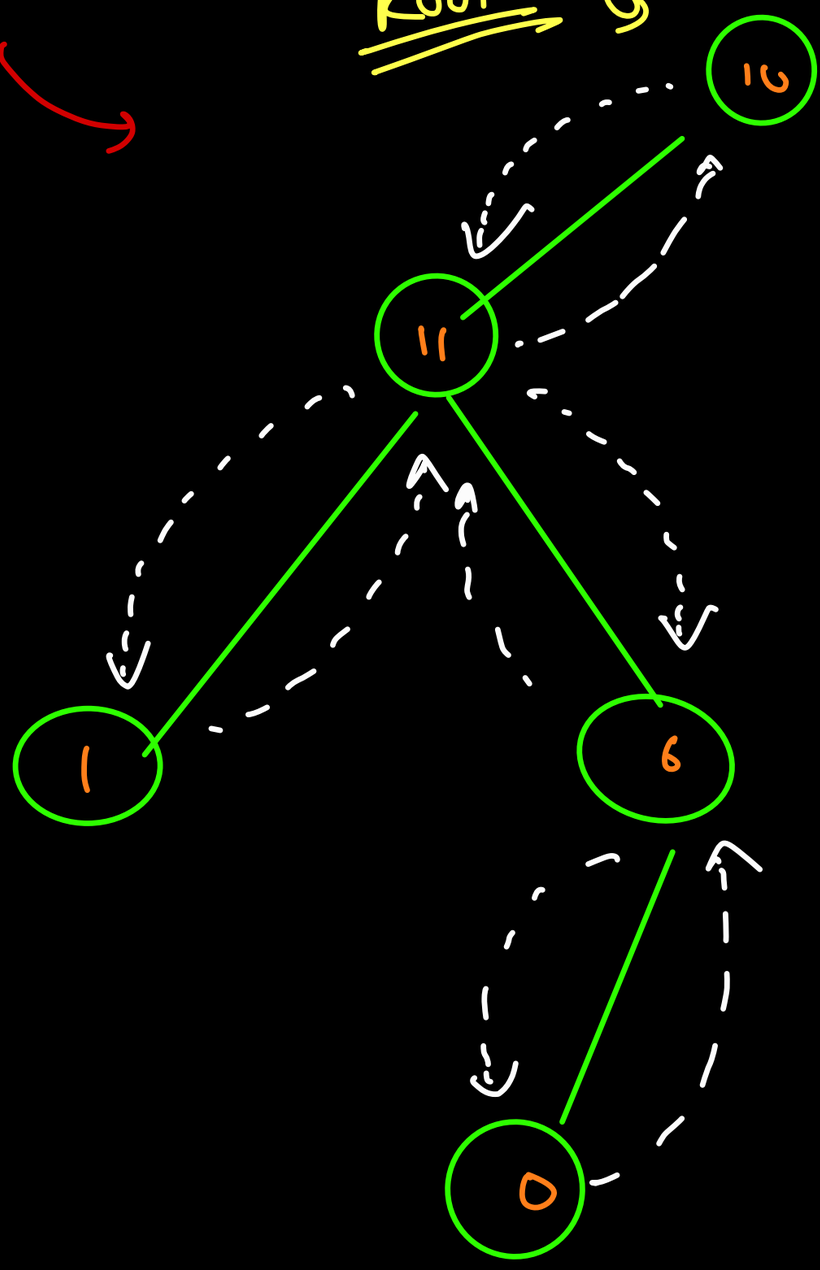
Given Binary Tree

Root

Recursion

DFS
depth first search



Postorder → 1, 0, 6, 11, 5, 7, 4, 3, 10

Inorder → 1, 11, 0, 6, 10, 5, 3, 7, 4

in depth first approach, we choose any one child and explore it's complete subtree & make other children

Wait:

LST → left Subtree

RST → right Subtree

Binary Tree

The 3 dfs types has only one diff i.e. the order to read elements.

Dfs

Preorder

Root

Left child → LST

Right-child → RST

Inorder

Left Child → LST

Root

Right Child → RST

PostOrder

Left child → LST

Right Child → RST

Root

PreOrder

↓

Root

LST

RST

Root of LST

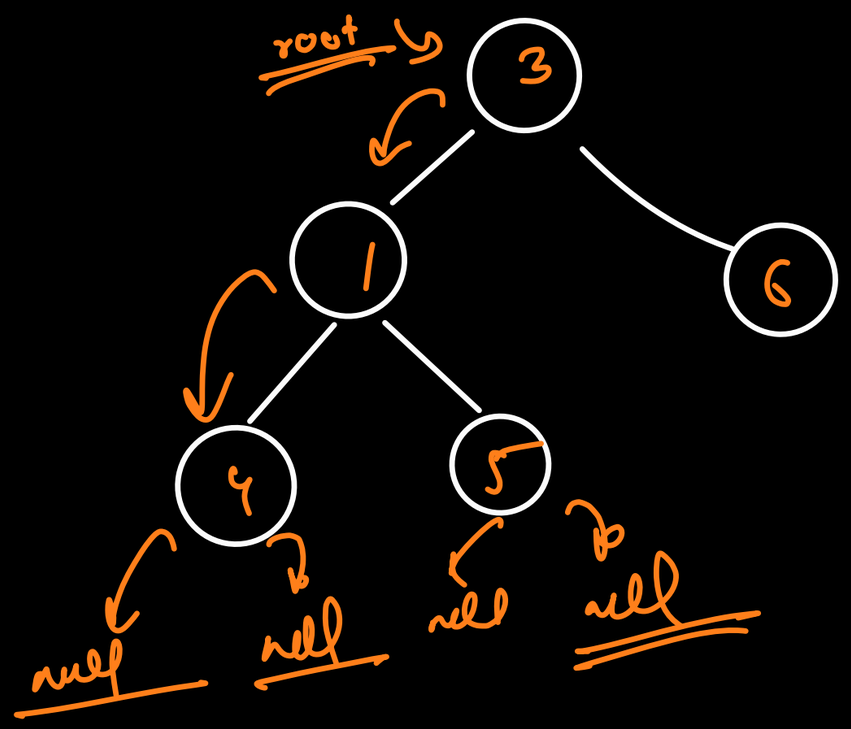
Root

Root of RST

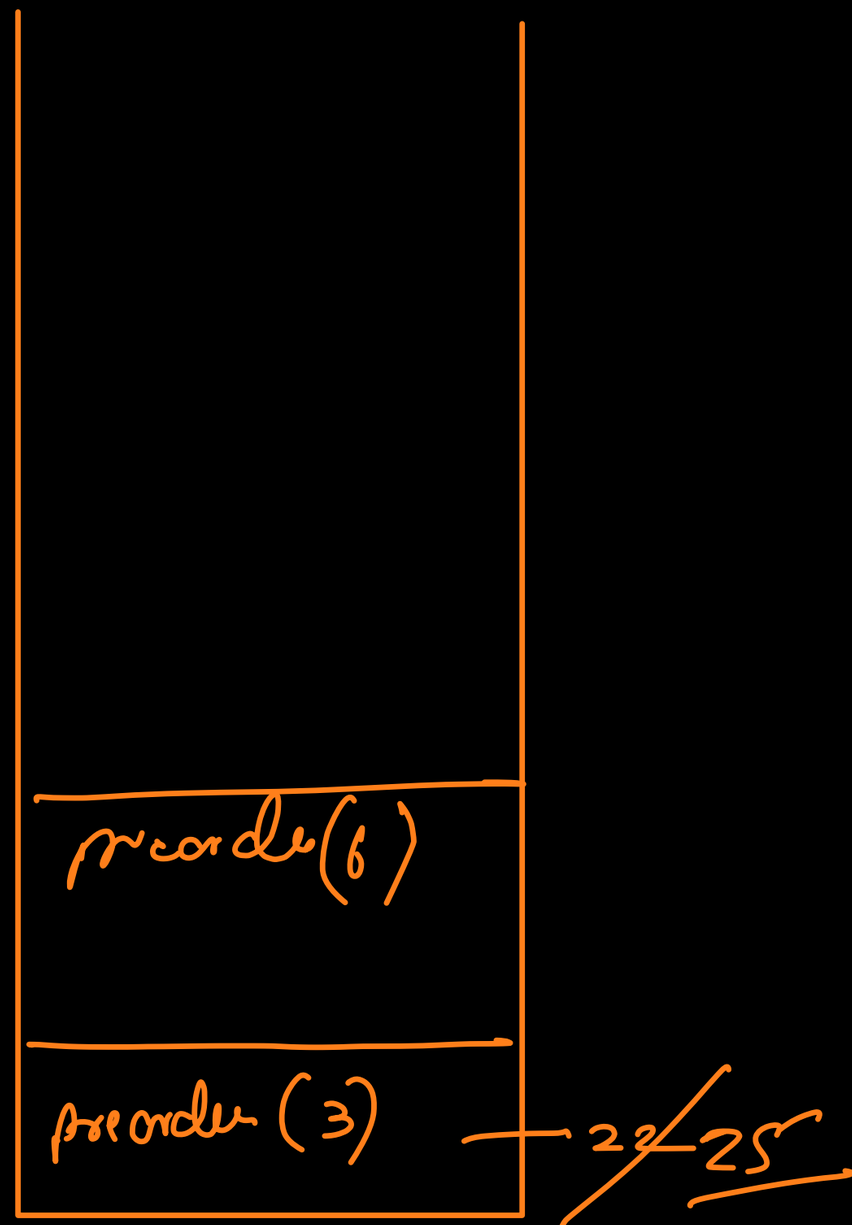
Leaf Node

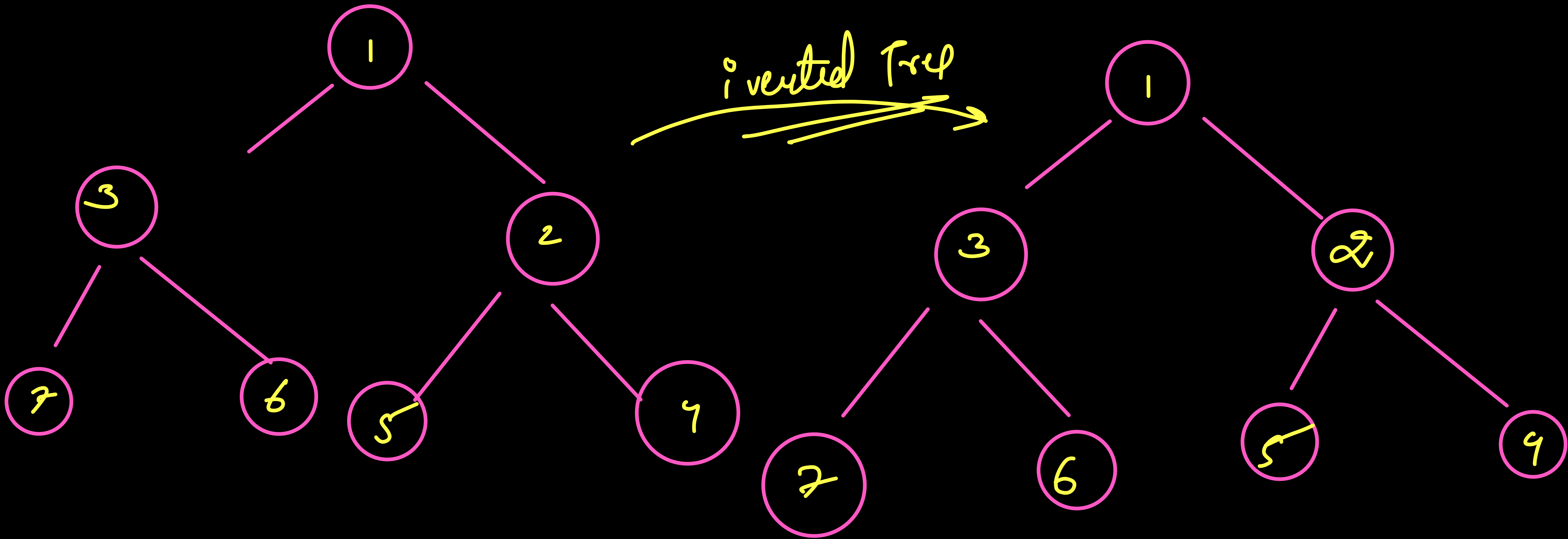
10, 11, 1, 6, 0, 3, 5, 4

```
let result;
function preorder(root) {
    if(root == null) return null;
    // If th›e root is not null, that means it has some data
    // process the root
    result.push(root.val);

    // go to the left sub tree recursively
    preorder(root.left);

    // go to the right sub tree recursively
    preorder(root.right);
}

var preorderTraversal = function(root) {

    result = [];
    preorder(root);
    return result;
};
```



[3, 1, 4, 5]
g
result





Post Order → Preorder

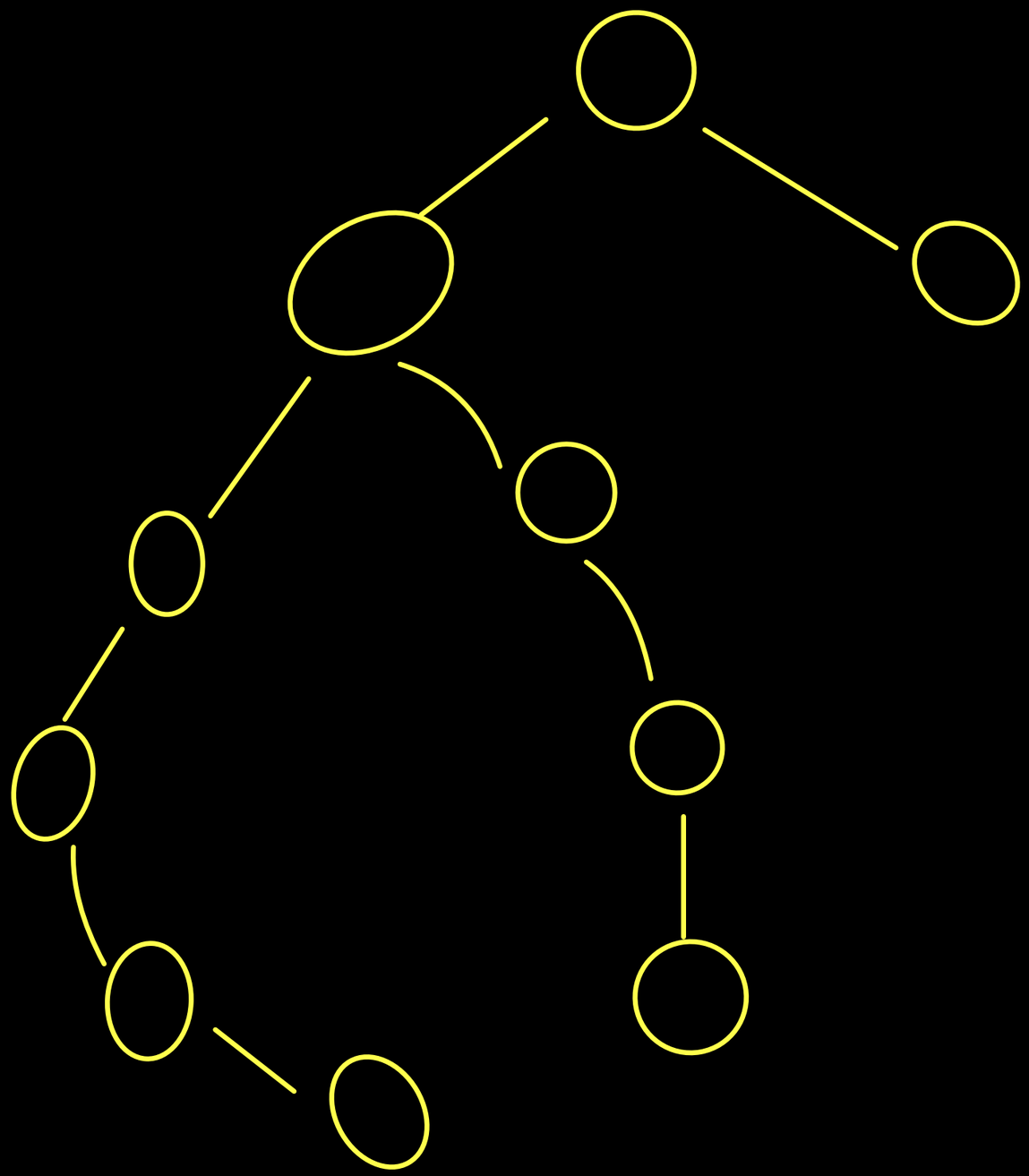
$f(root) = f(root.left)\ f(root.right)$

this func^n invents the

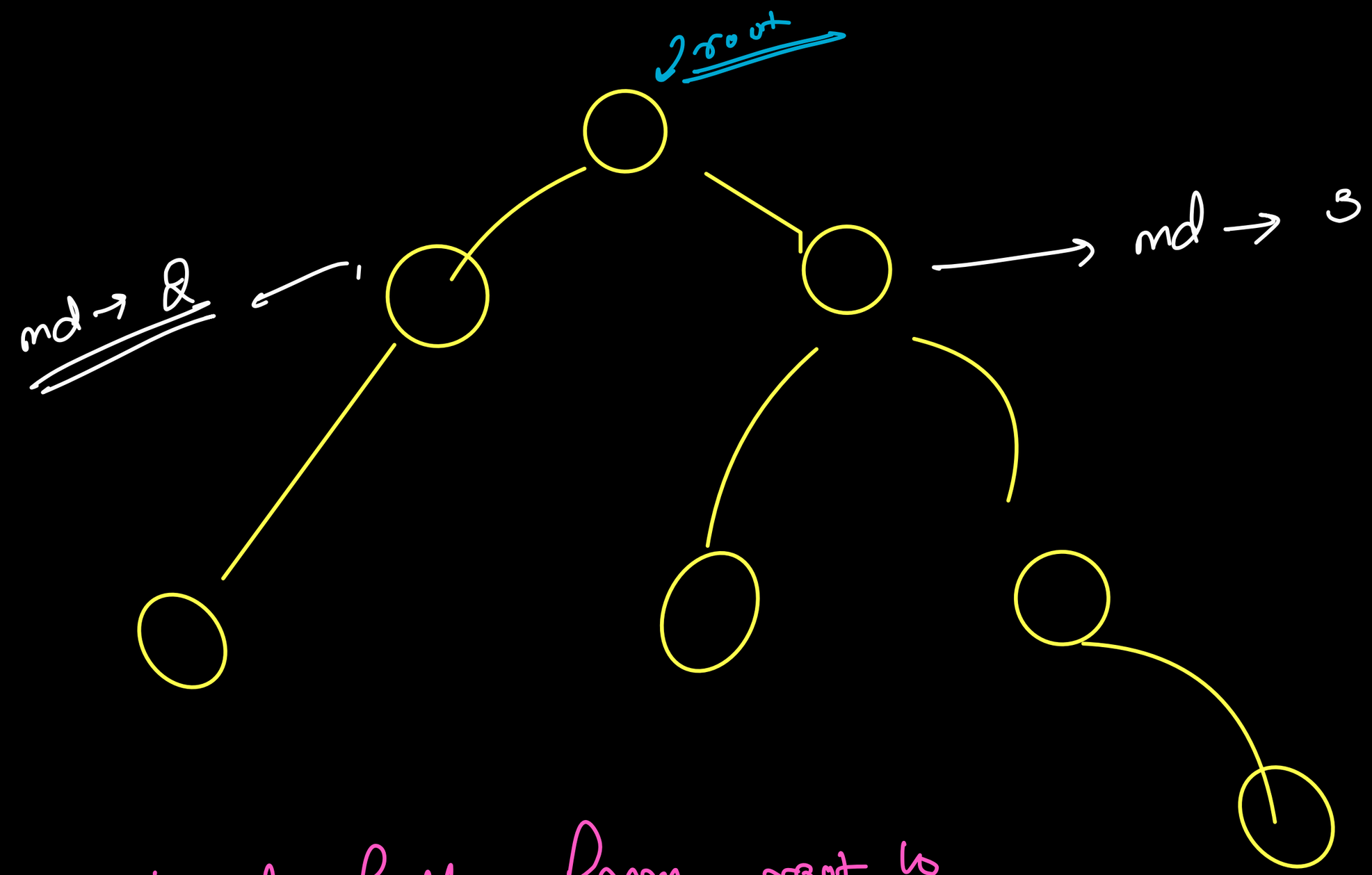
gen 6.7

// invent the callable req

sup(lst, ks)



JOIN THE DARKSIDE

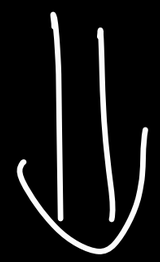


max depth → longest path from root to any leaf

$$f(root) = 1 + \max\Big(f(root.left),\ f(root.right)\Big)$$

2

maximum depth of a B.T



= lmd → f(root.left)

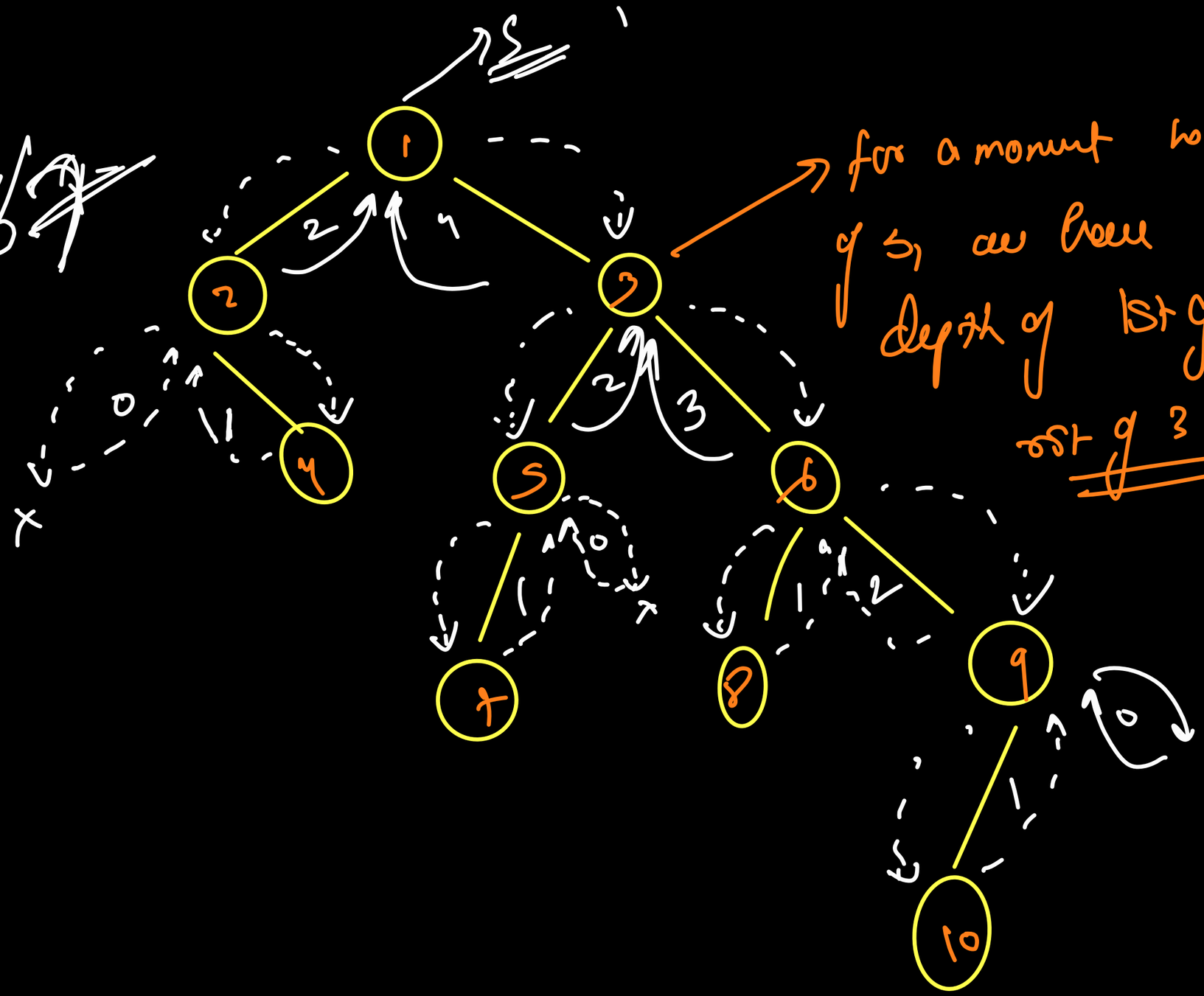
rmd → f(root.right)

ans → 1 + max(lmd, rmd)

diameter

ans = -∞

global



for a moment to calculate depth of 5, we know ans of depth of 1st of 3 & 2nd of 3.

$f(root)$ $\Rightarrow$ $lst = f(root.left)$

$rst = f(root.right)$

$ans = max(ans, lst + rst + 1)$

return $1 + max(lst, rst)$

↓

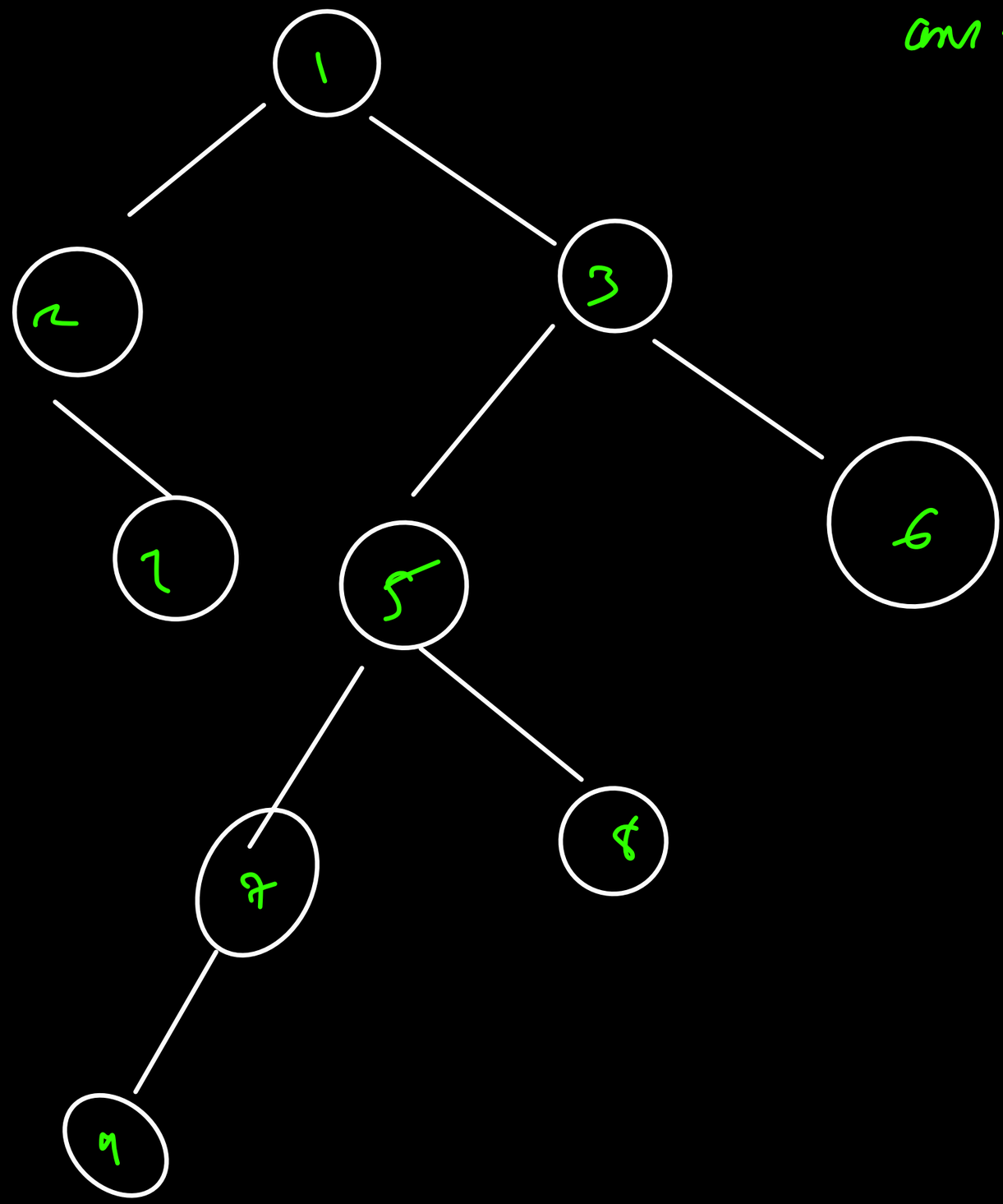
returns the max depth of the tree rooted at root

↙

Calc the diameter passing from the current root node.

$|h_l - h_r| \leq 1$

ans = T



ans = true

f(root) =>

hl = f(root.left)

hr = f(root.right)

ans = ans && (abs(hr-hl) <= 1)

return if max(hr, hl)