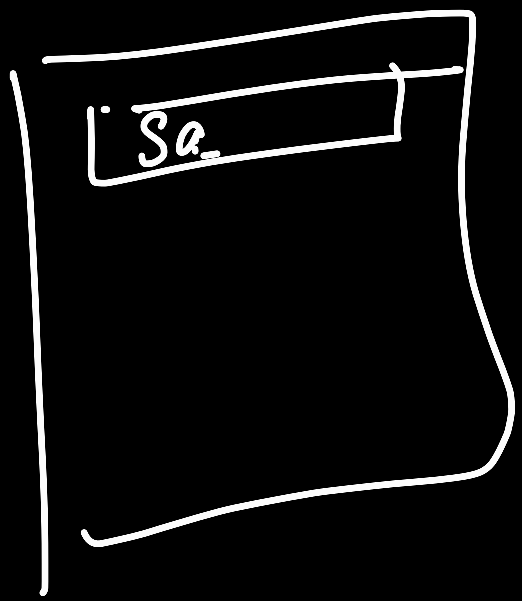
TRIE

Sam
Sameer
Sanket
Sarthak

Linux

Windows

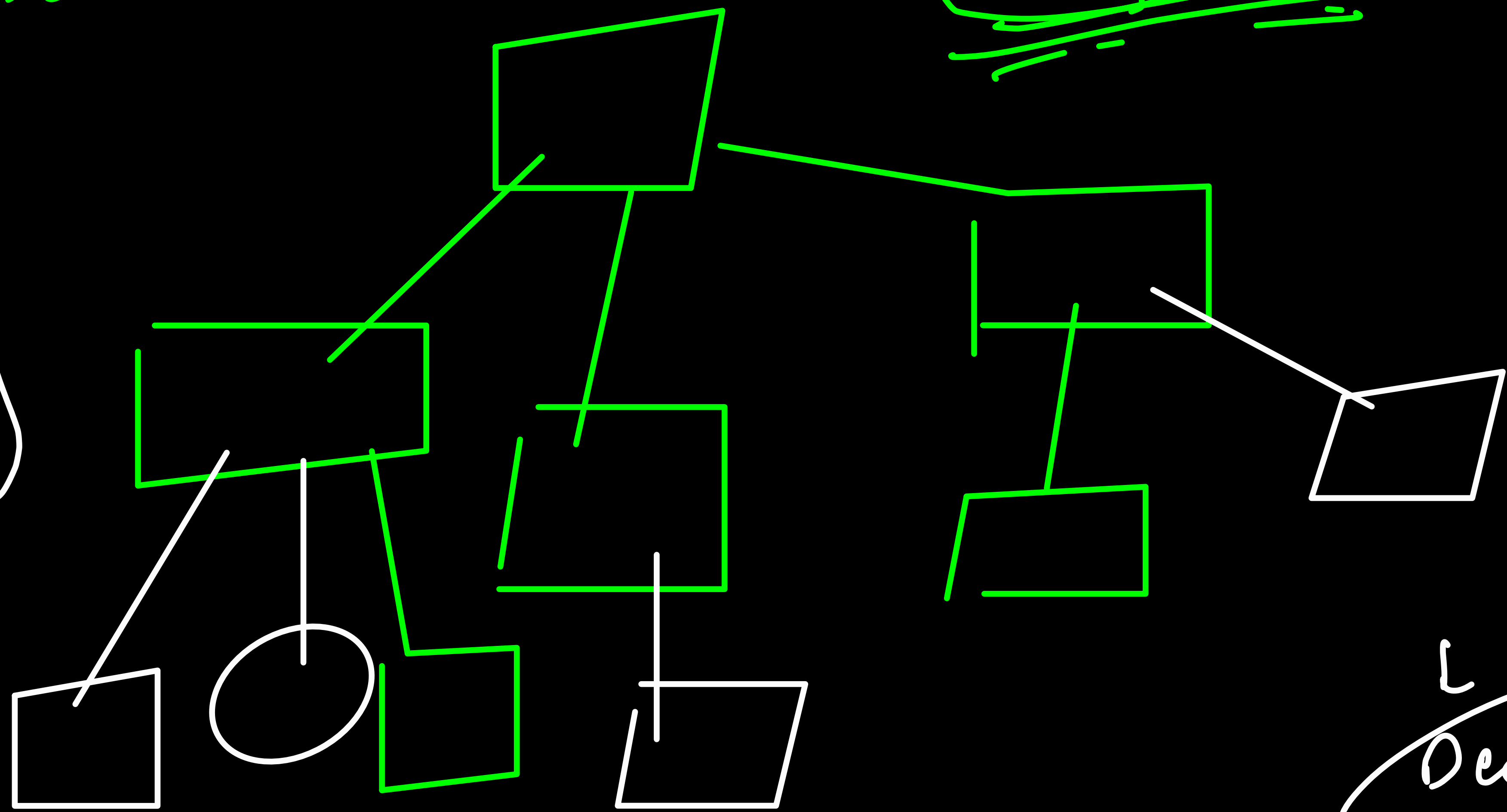


TREES

Hierarchical

Key query

Folders



L h ?

Decision Tree

Database → RDBMS → MySQL

Indexing

Trees

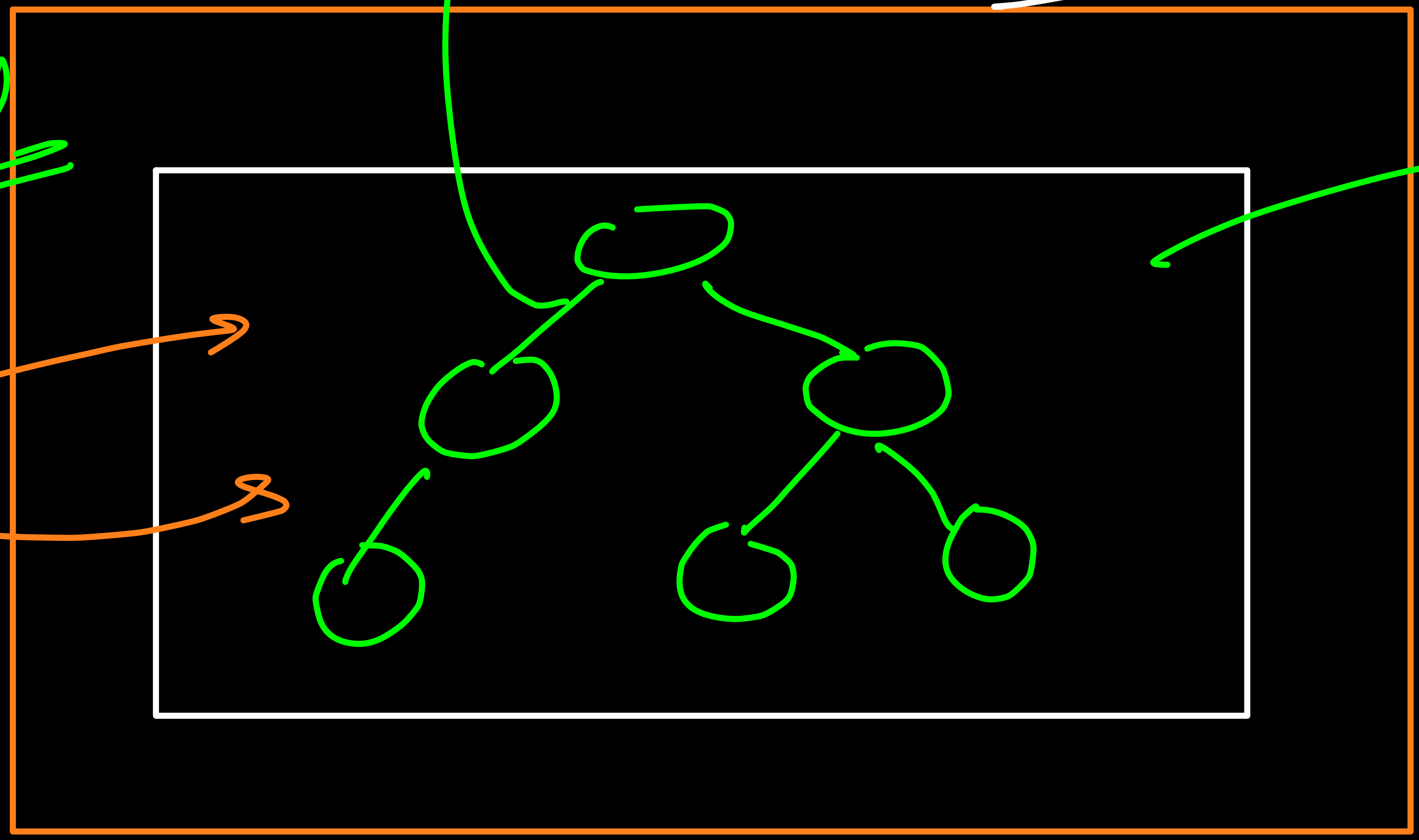
Document object Model

Dom Tree

Browser

HTML

React



<body>
<p> </p>
<div>
<p> </p>
</div>
</body>

body

p div

p

2 DOM

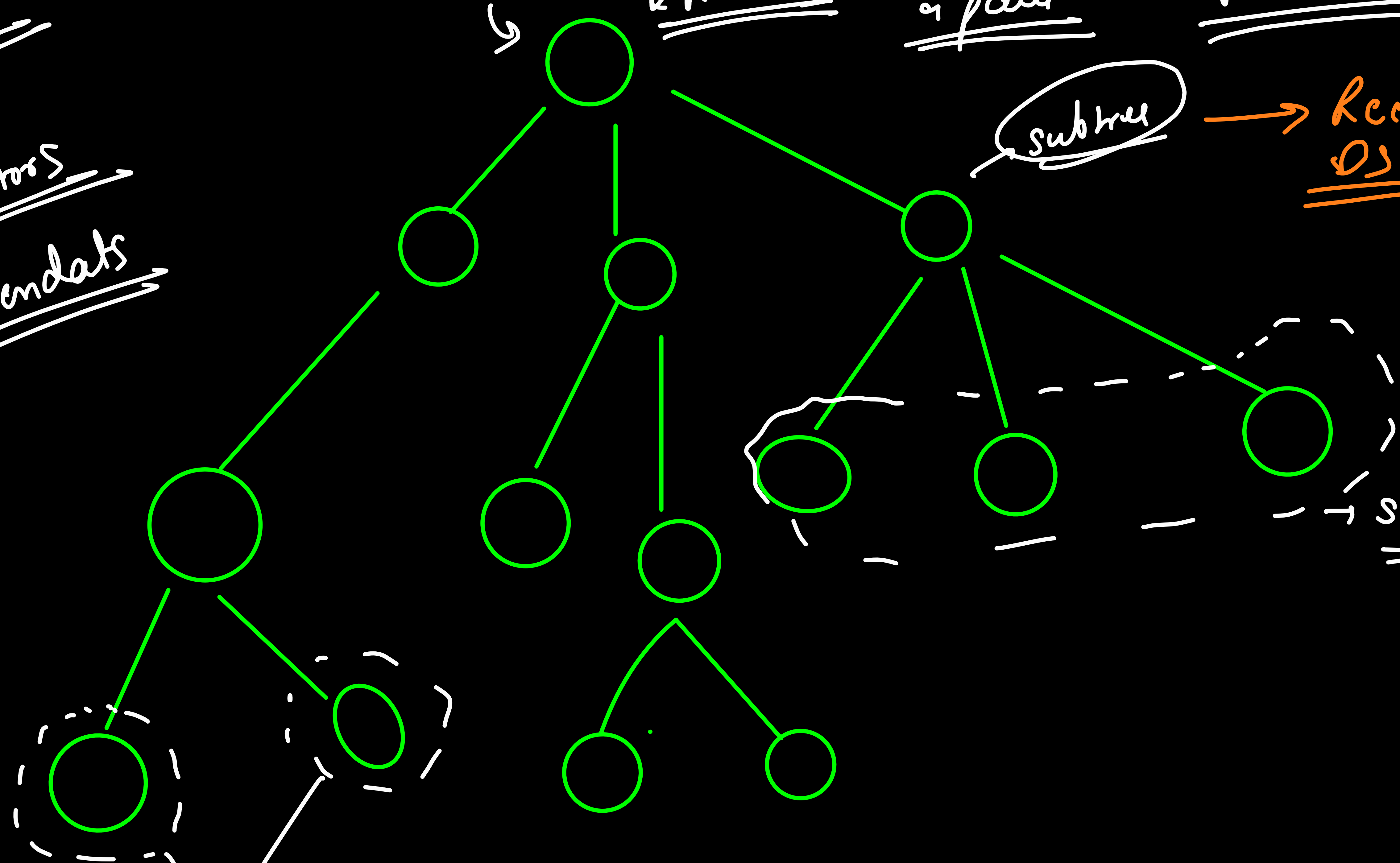
Rooted

Ancestors

descendants

Root → do not have a parent

Parent - child

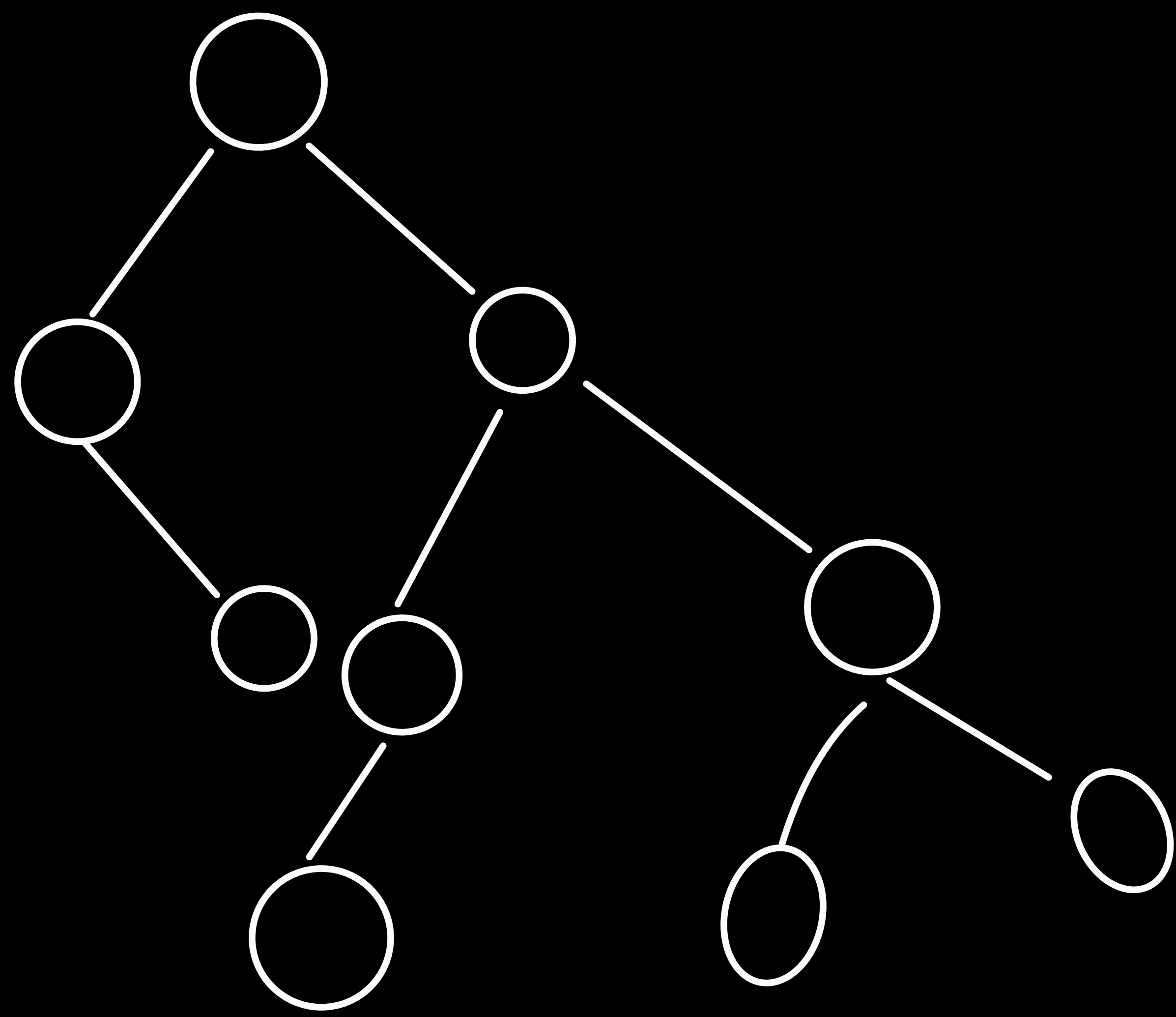


subtree → Recursive DS

sibling

Terminal / leaf nodes

a tree in which every node can get max two children are called as Binary trees



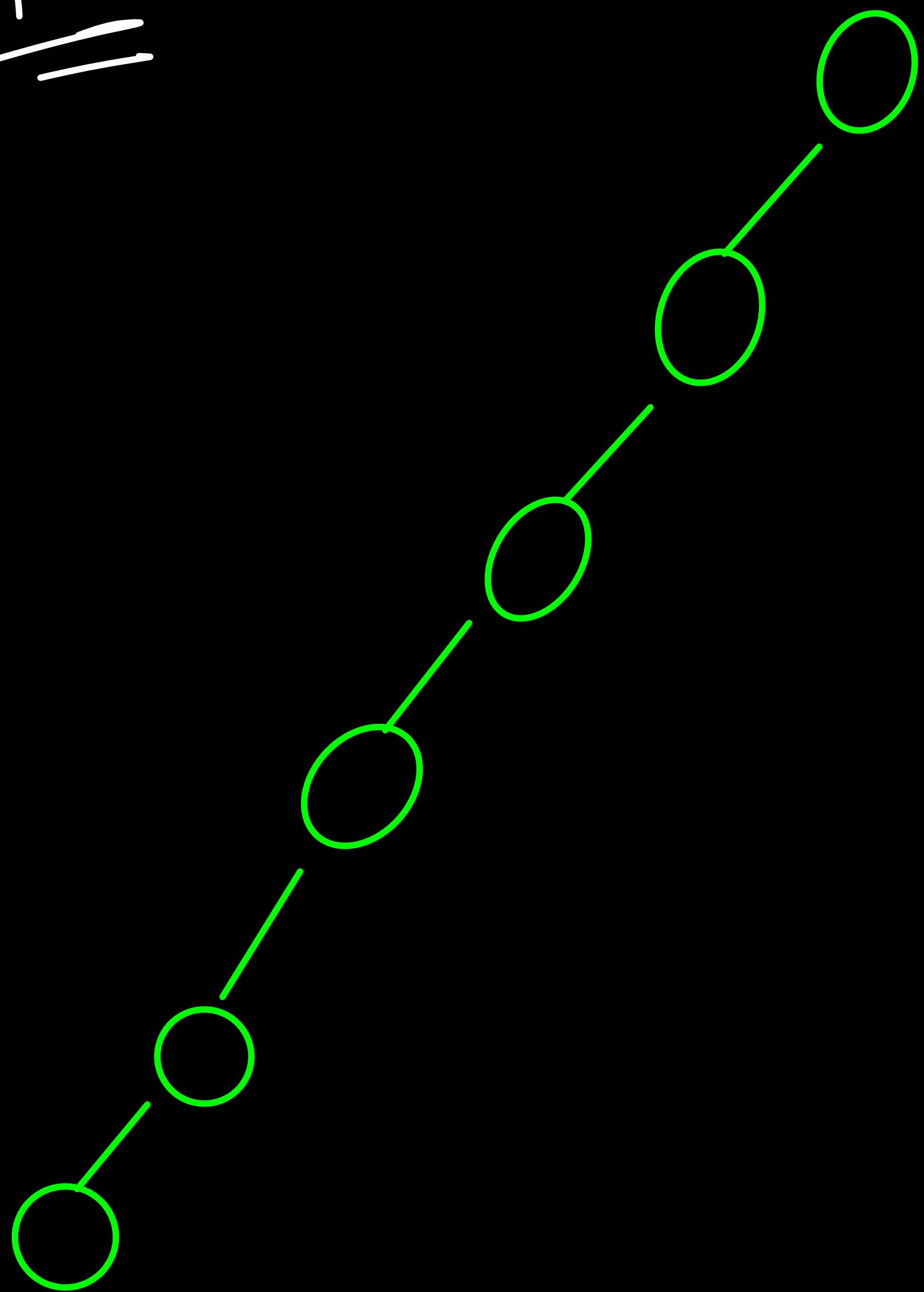
if in a tree a node can get max 3 children, that's a ternary tree

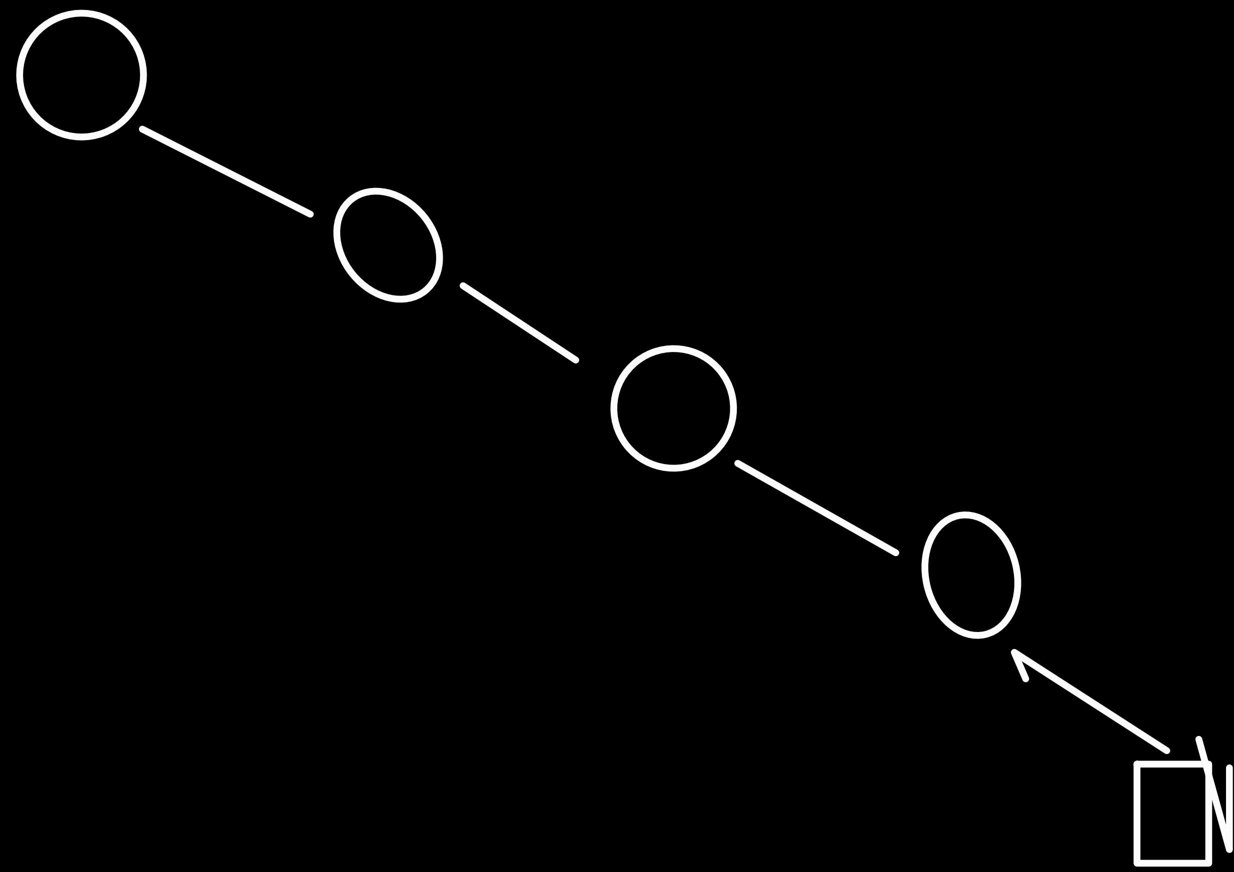
if every node can get max n children → n-ary tree

generic tree

Types of BT

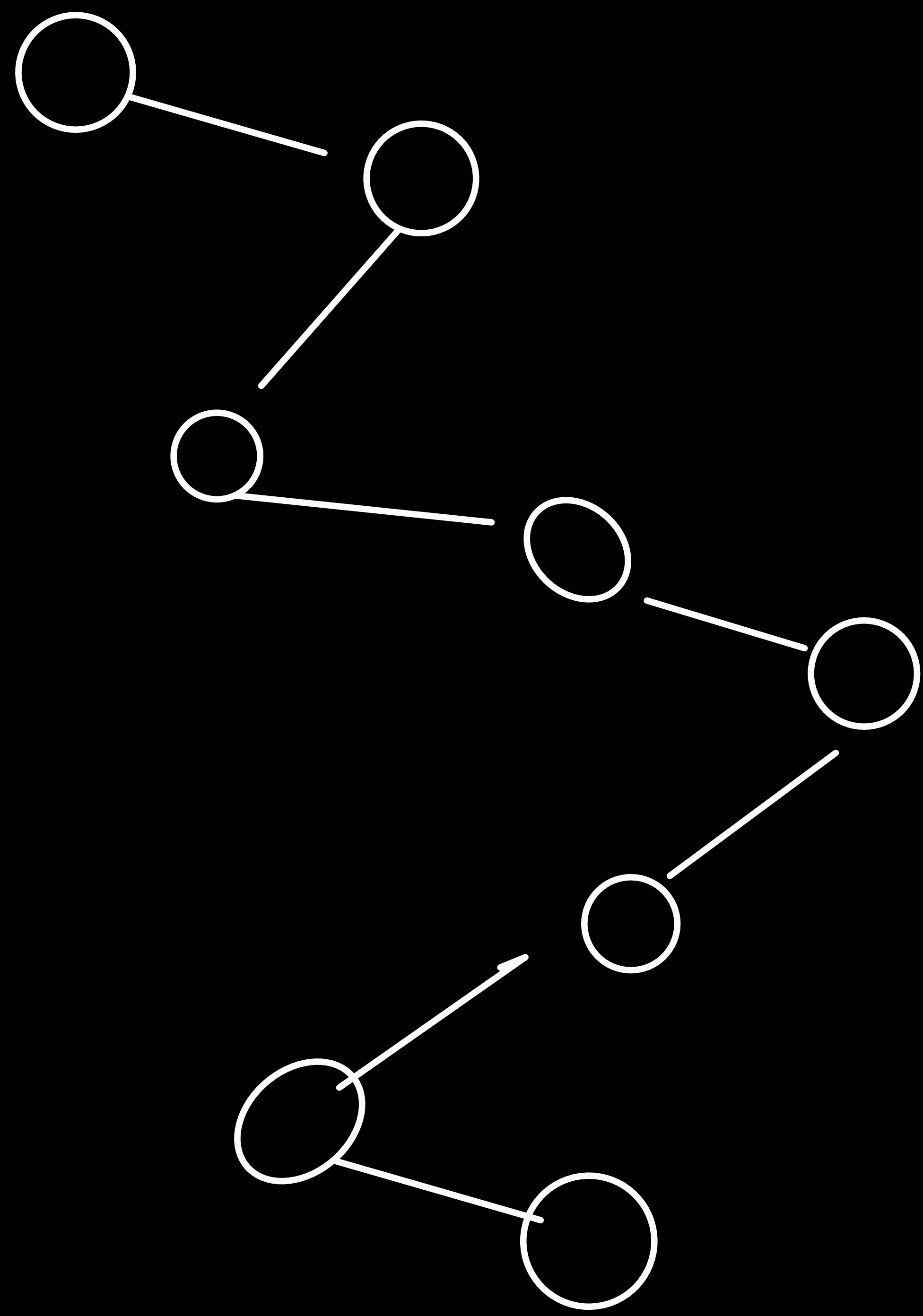
1) Skewed Binary Tree





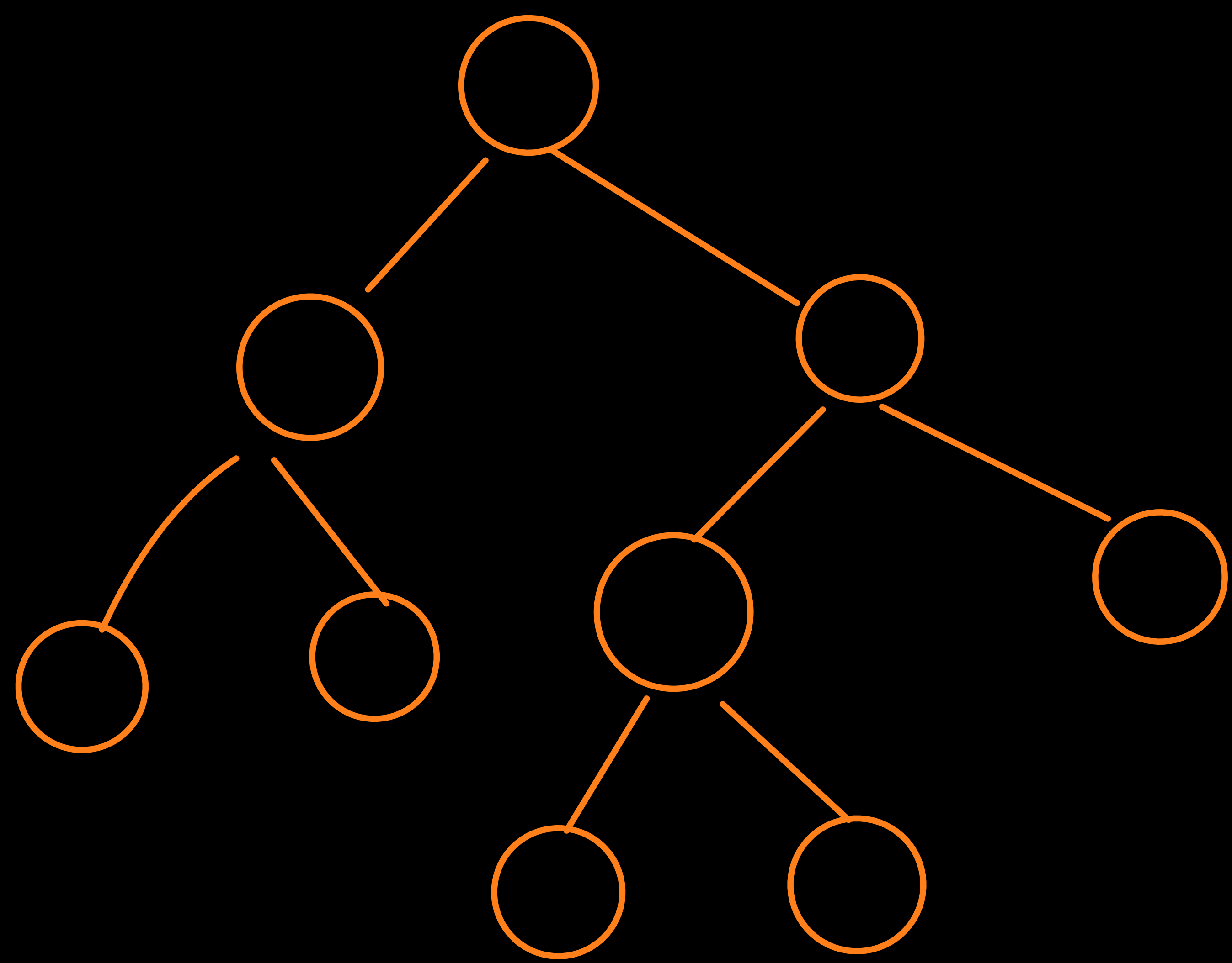
(2) Degenerate Tree

every node has max 1 child



③ full binary tree → In a full binary tree, every node has either 2 children or no children.

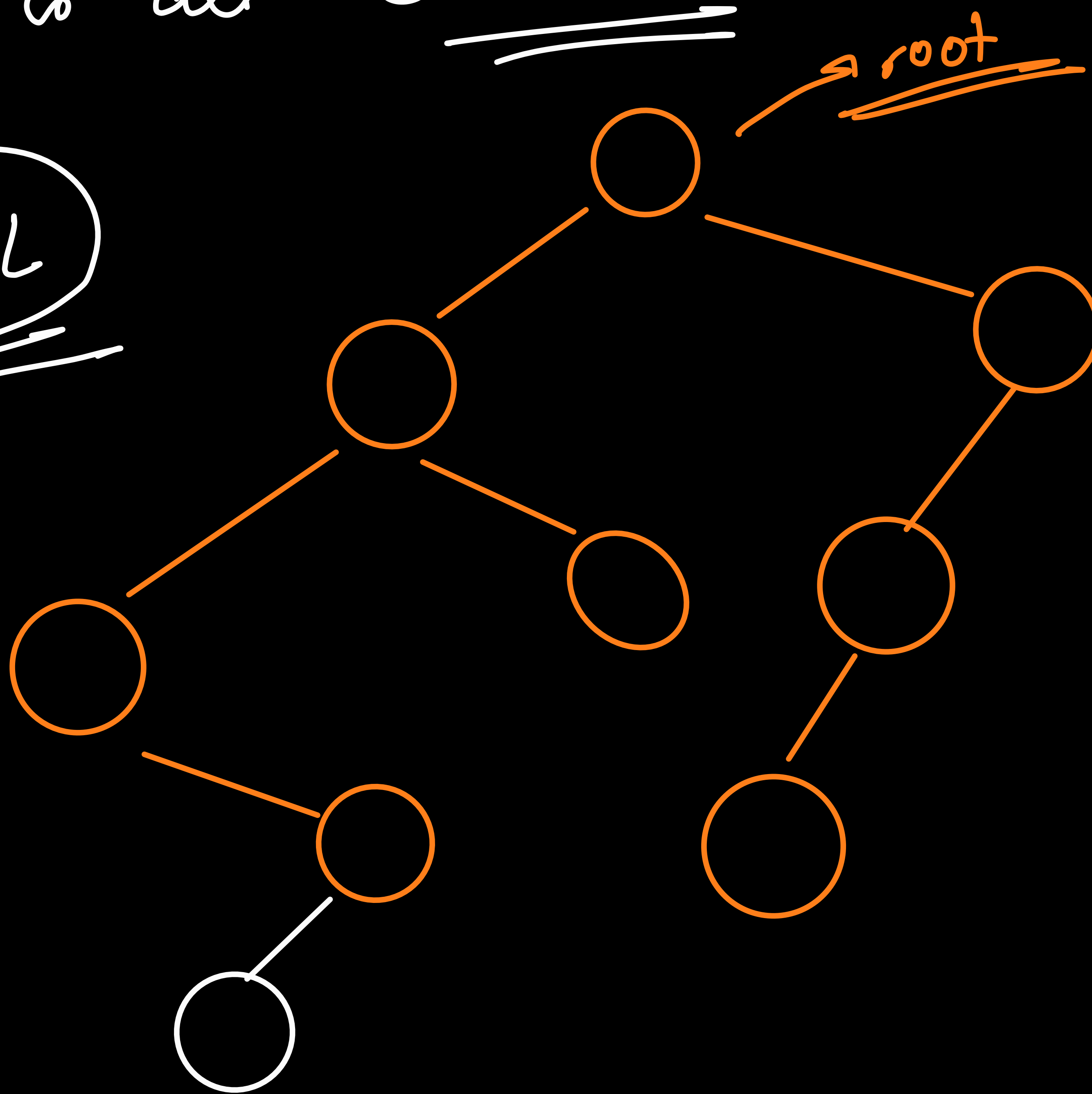
A segment tree (RMQ) is an example of full binary trees.



④ Balanced binary trees → In a balanced binary tree, the absolute diff btw height of the left subtree & right subtree is at max 1 and this is recursively true for all subtree.

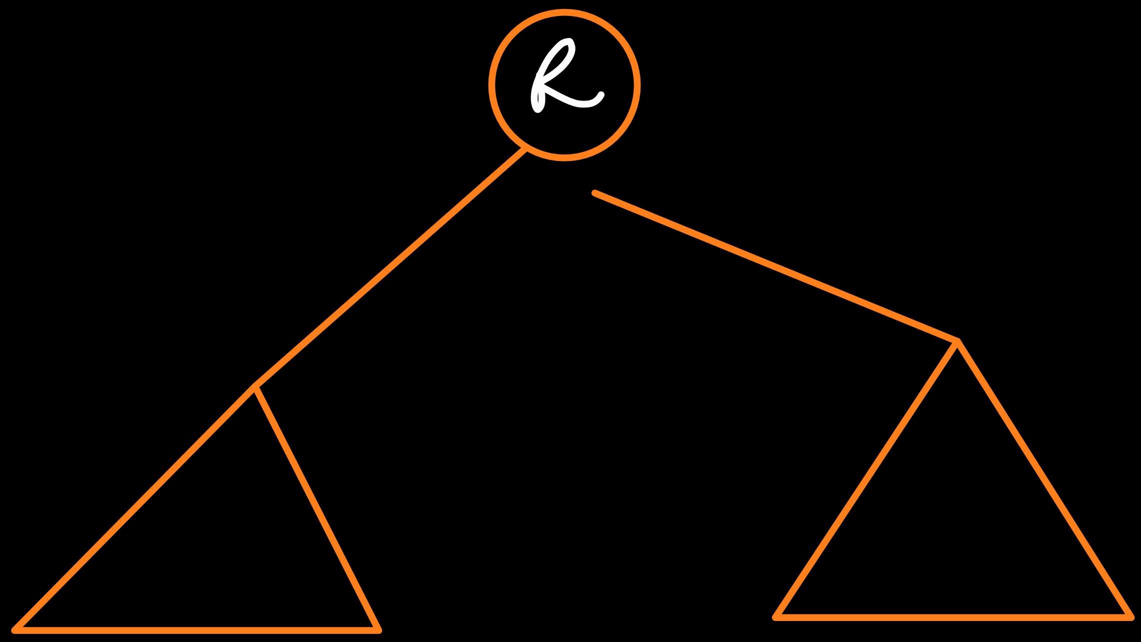
$|h_{LST} - h_{RST}| \le 1$

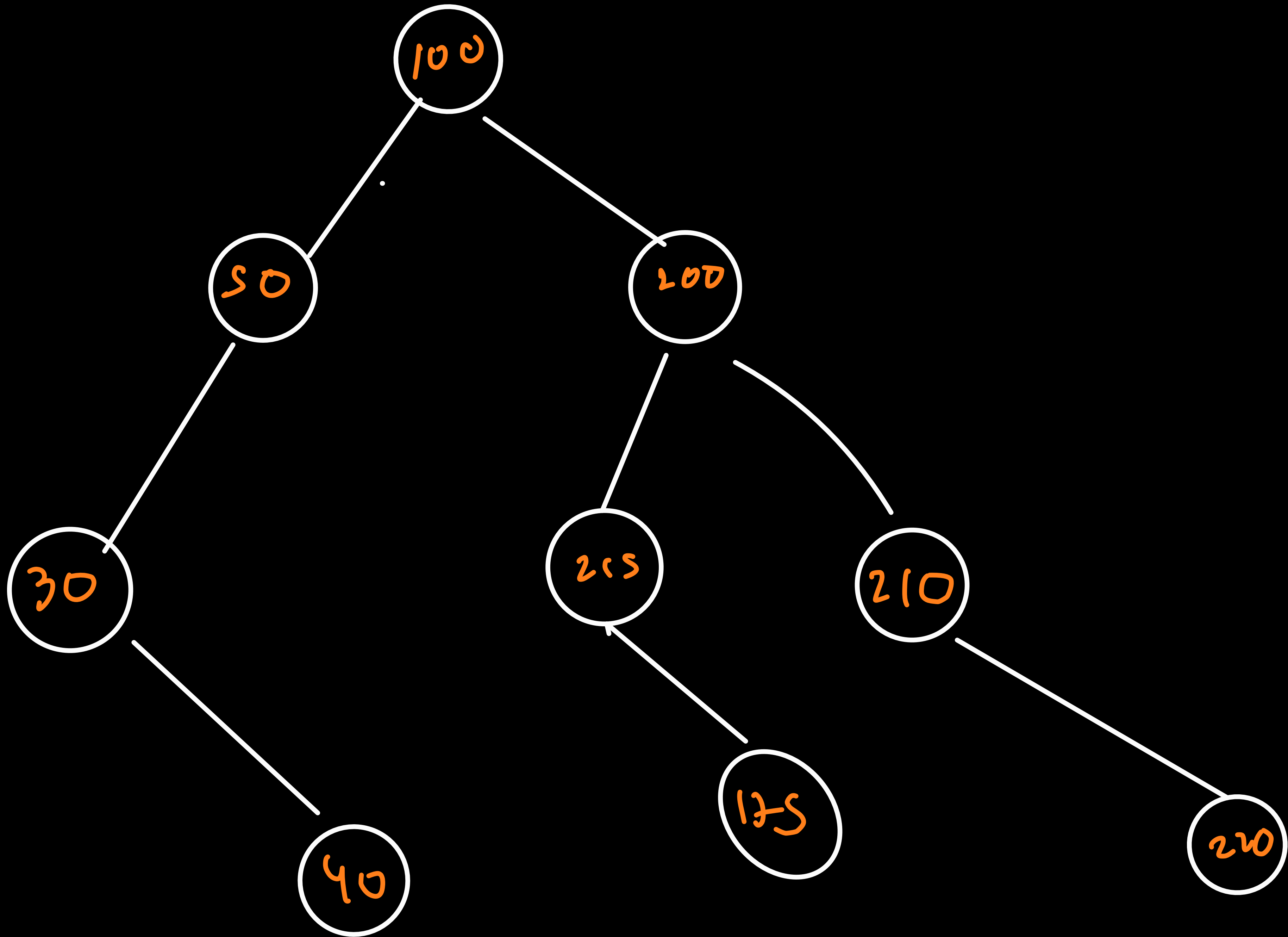
Ex → AVL



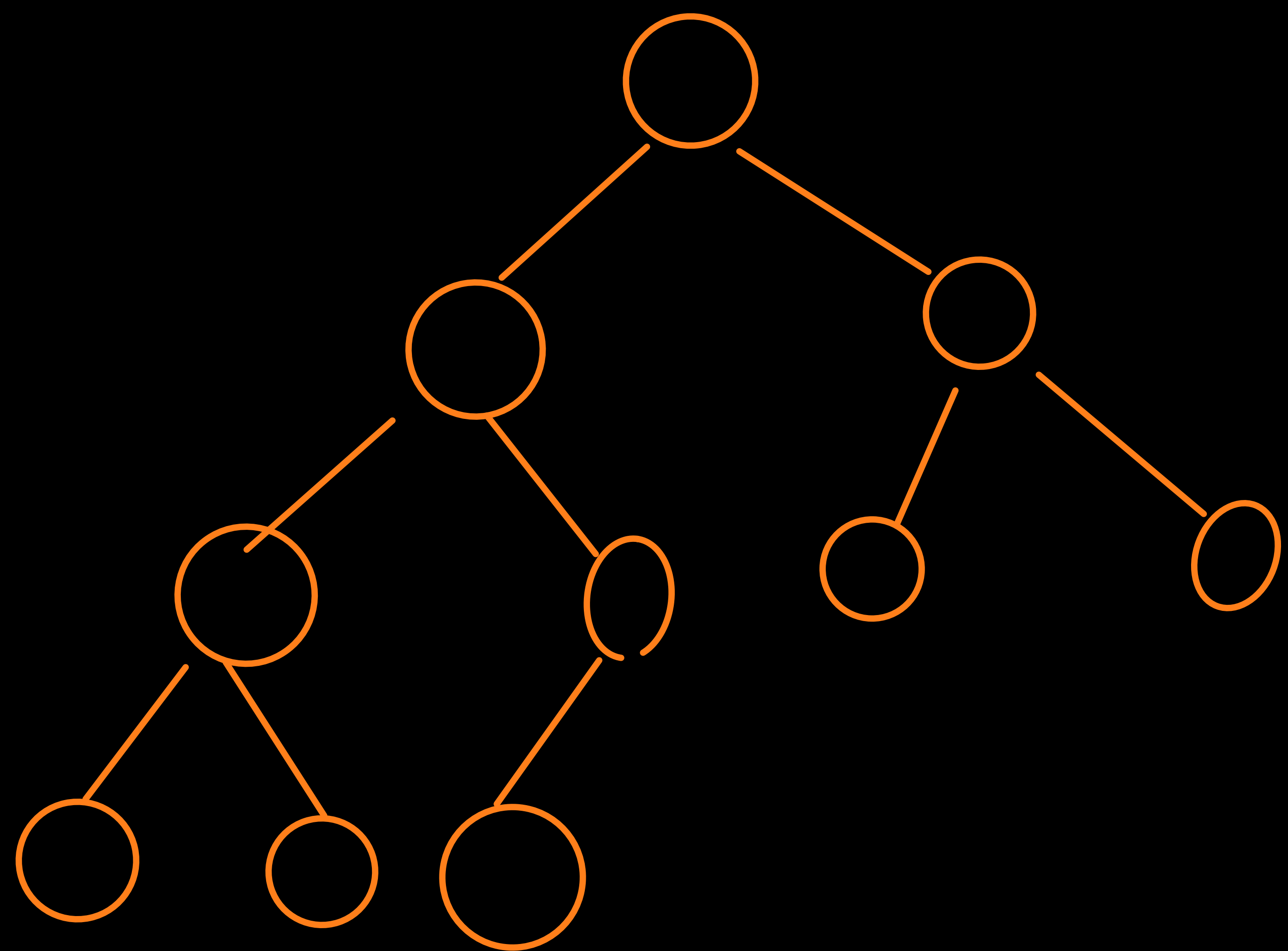
Binary Search tree

every node in the left subtree should be less than the root & every node in the right subtree should be greater than root. & this should be true recursively for all subtree




100
50
200
30
215
210
40
175
220

Complete Binary tree → In a complete BT, all the level except the last are full & last is level is filled from left to right without skipping any child.



Perfect

even the last level is full.

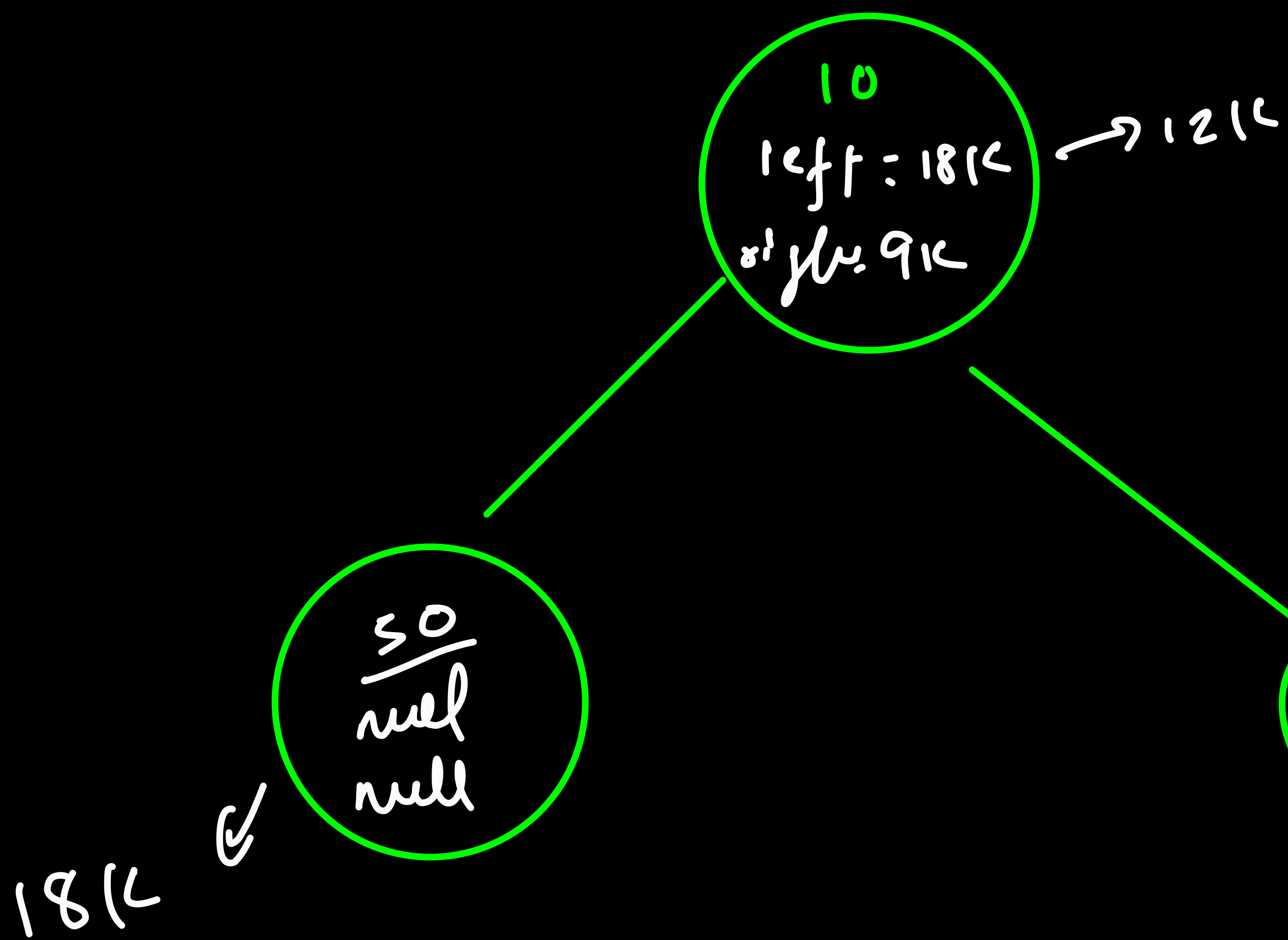
Node

[illegible]

data

left

right



Arrays

TRAVERSALS

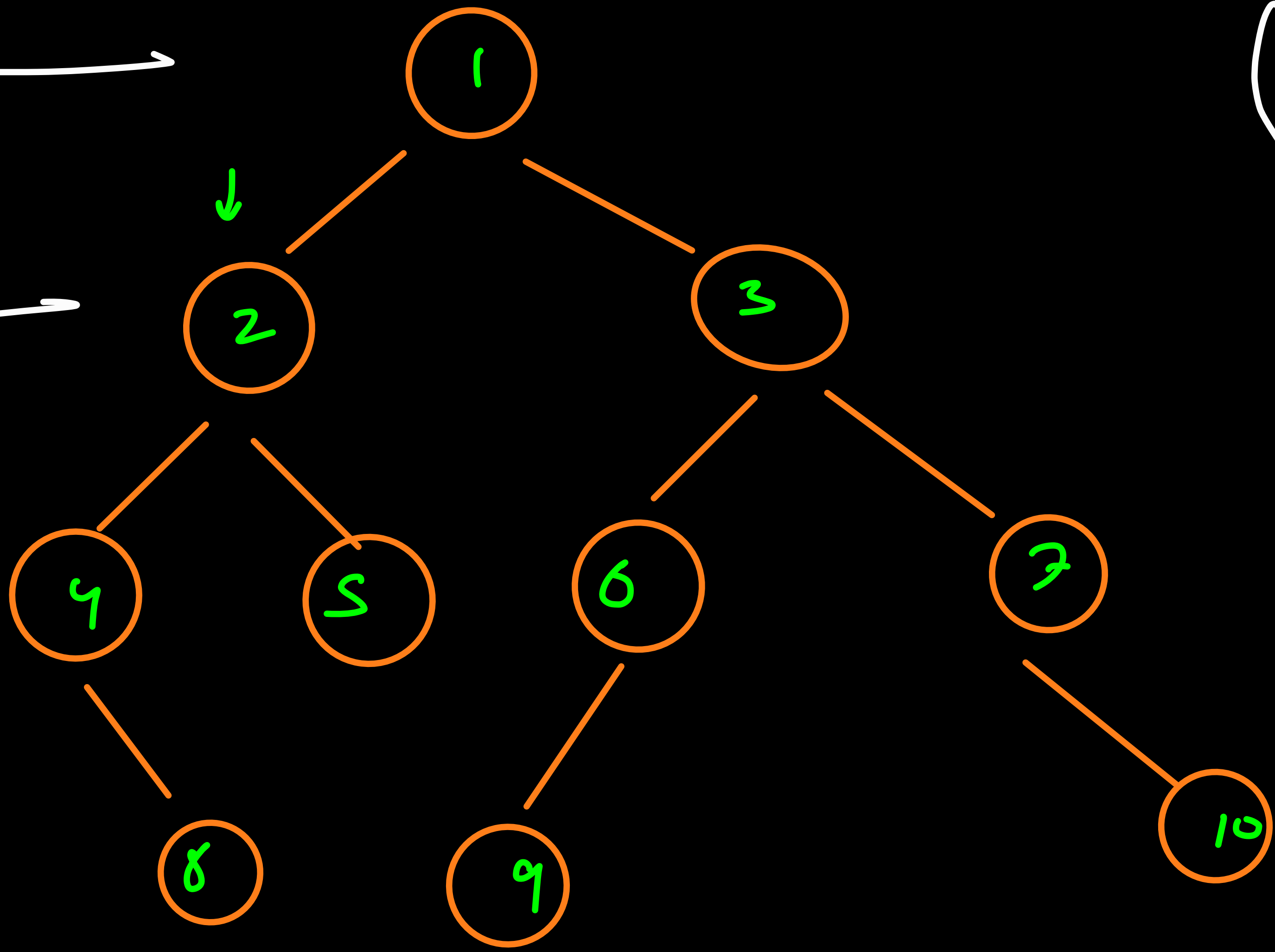
↳ level order traversal) bfs

level 0

level 1

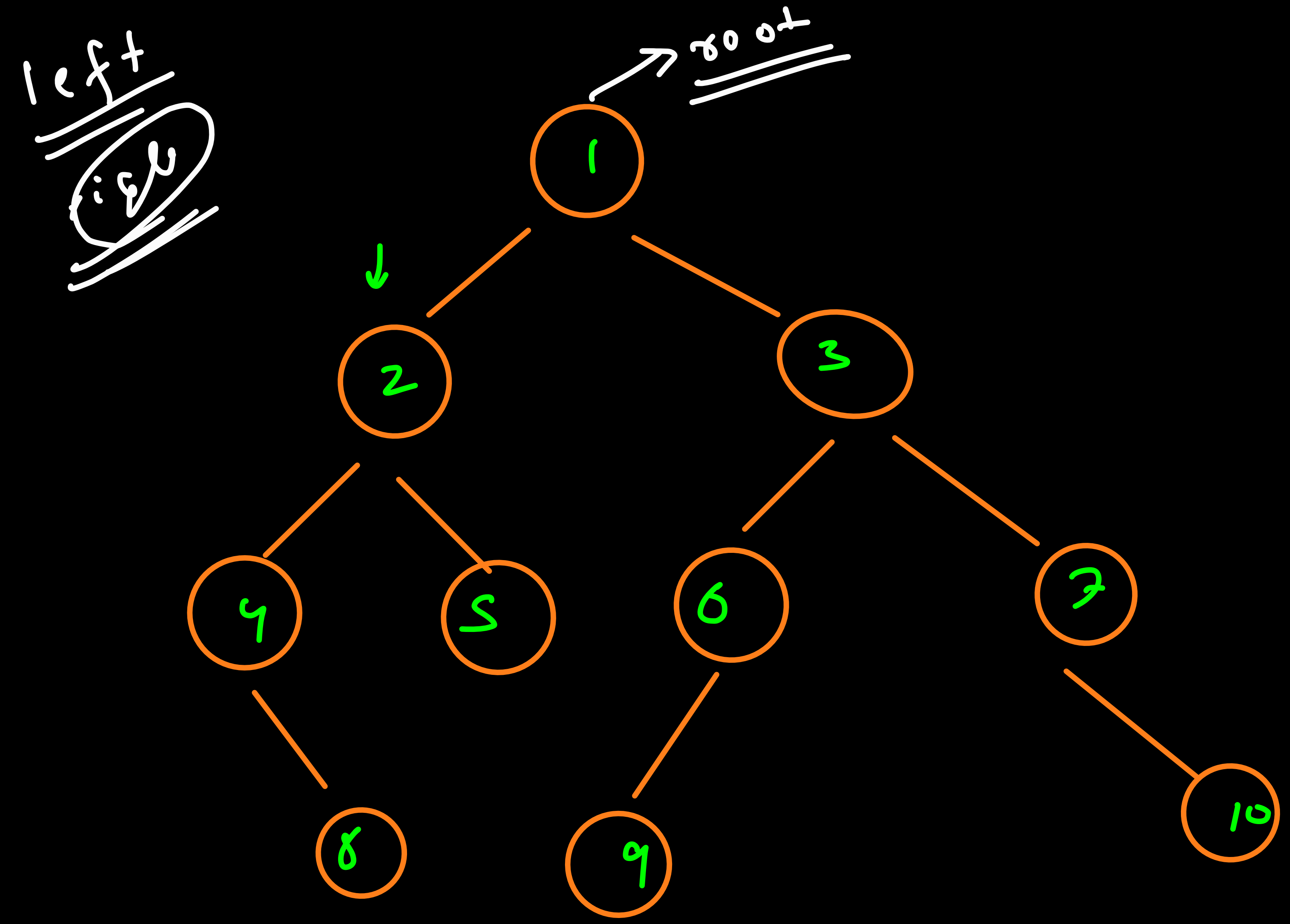
level 2

⋮



dfs

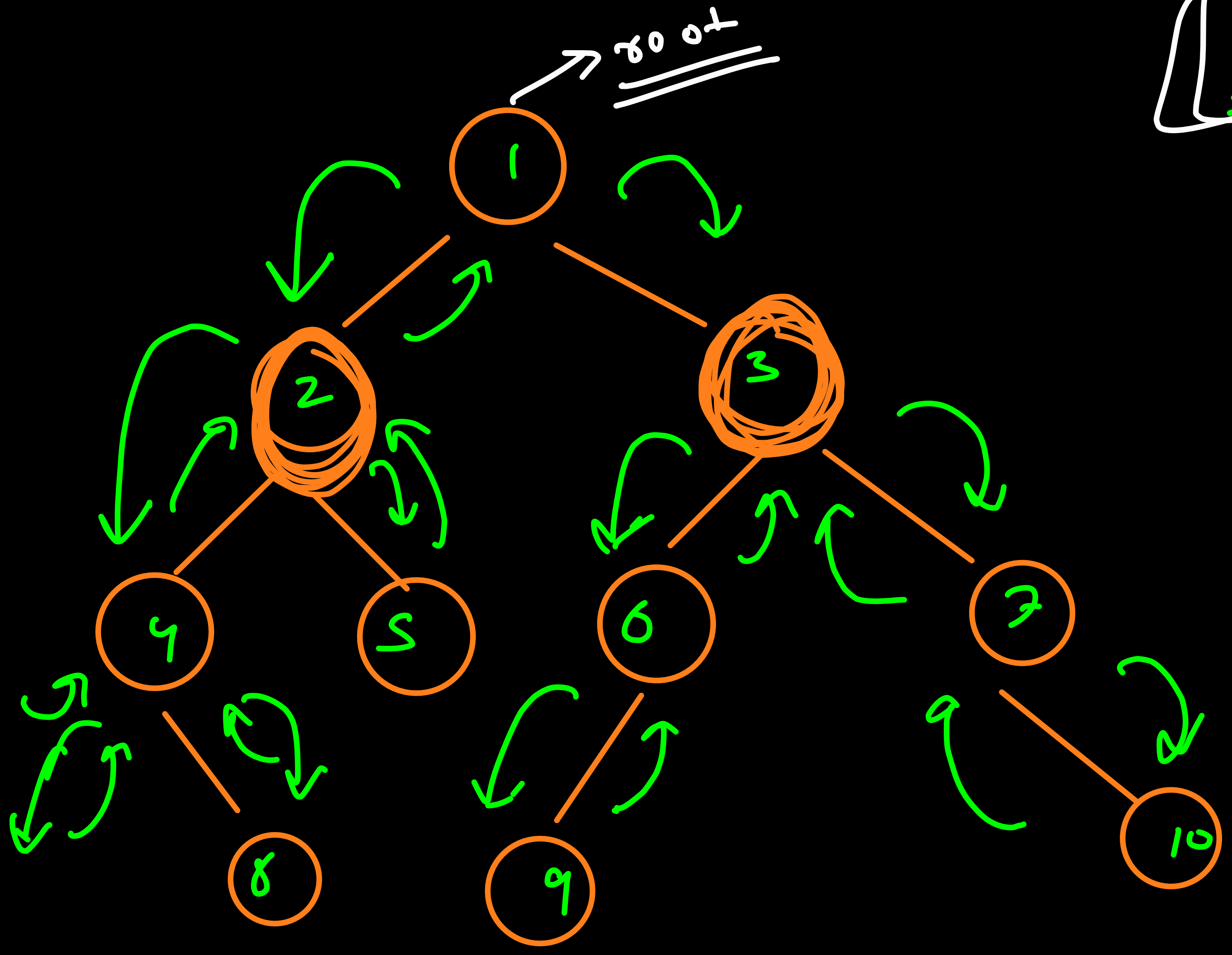
- pre order
- in order
- post order



DFS → depth first Search

pick any one child and explore the complete subtree of it first, meanwhile the other child will wait.

This should be done recursively.



Dfs

pre	in	post
root	left	left
left	root	right
right	right	root

pre(1) → print(1)
pre(2)
pre(3)

pre → 1 2 4 8 5 3 6 9 7 10

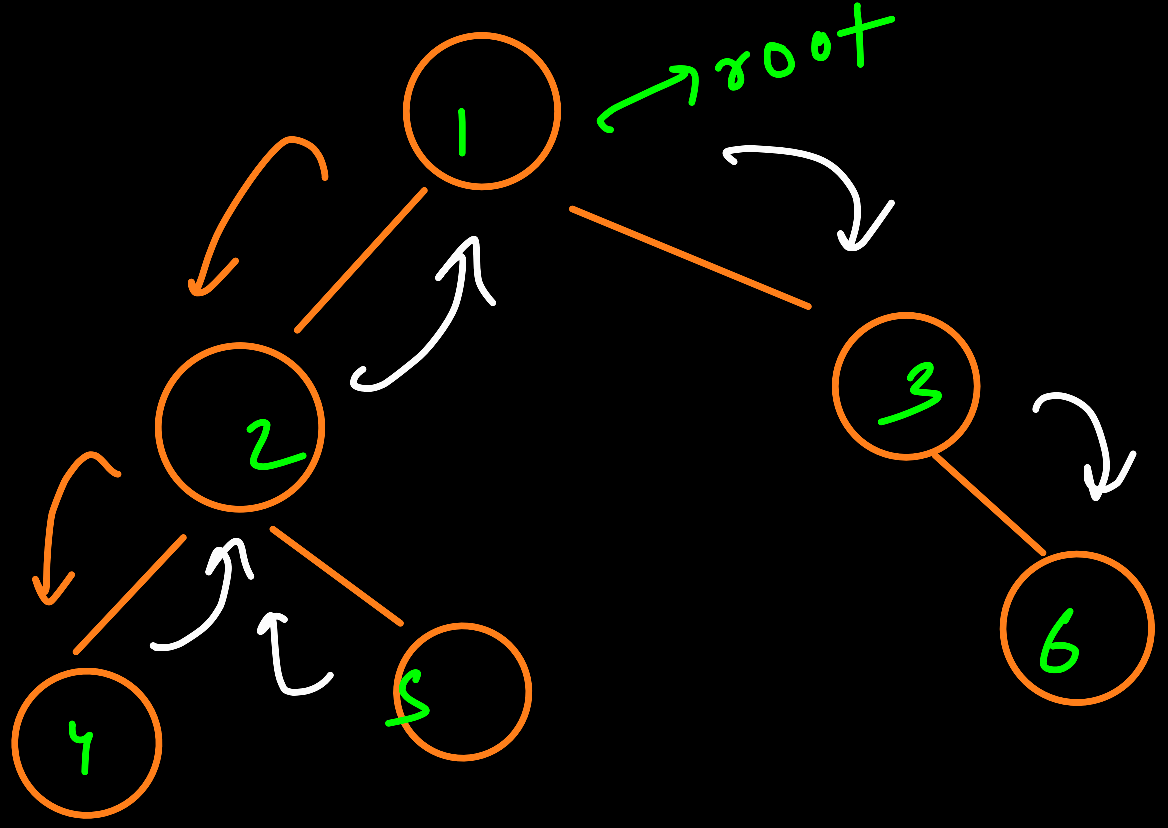
pre (r) $\longrightarrow$ print (r) $\rightarrow$ read root first

pre(r.left)

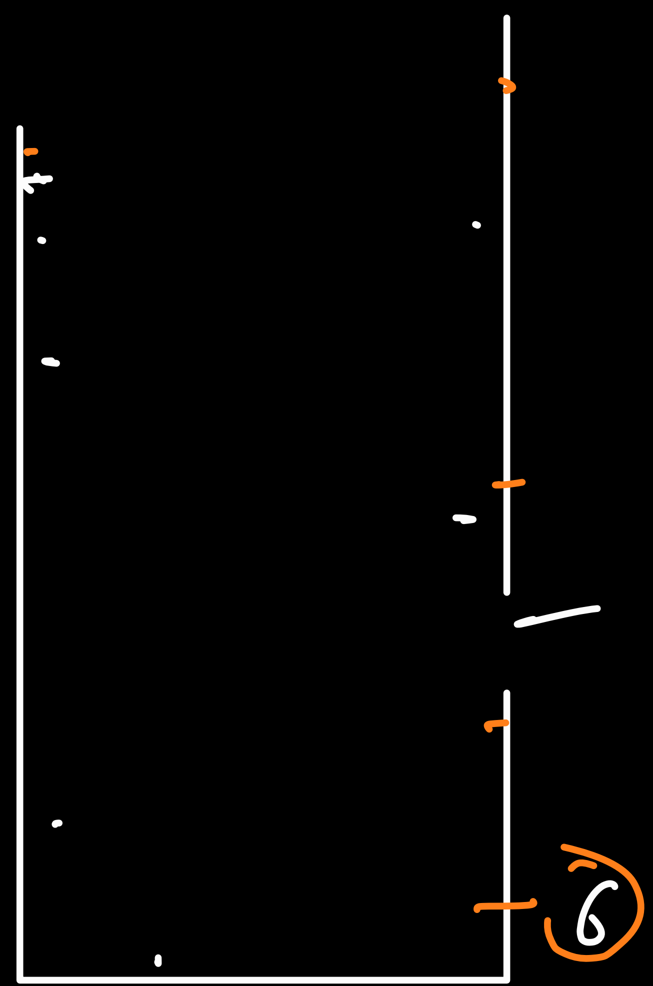
pre (r. right)

func^n does a pre order traversal on a tree rooted at r.

Base Case $\rightarrow$ (r == null) return

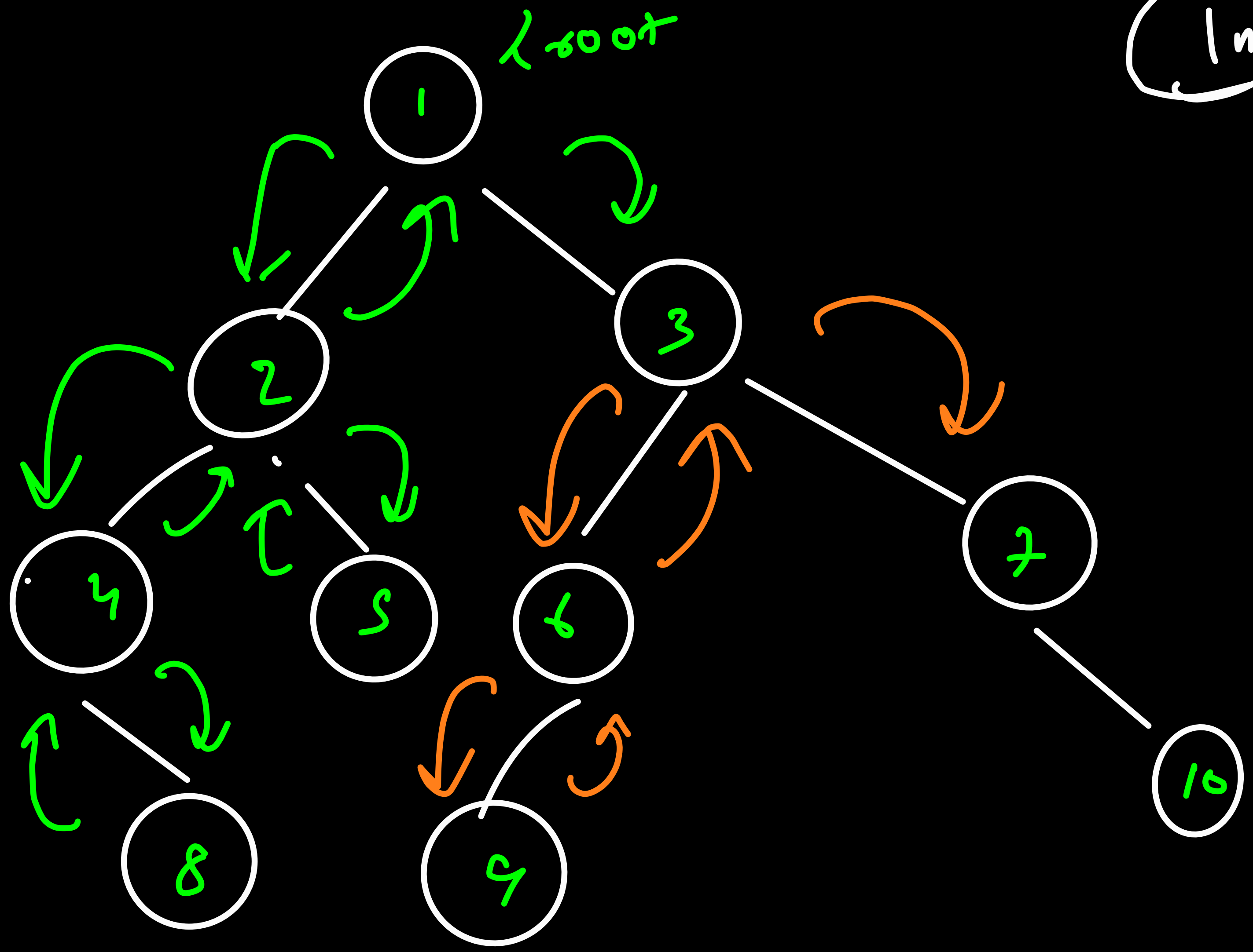



```
1) pre(r) {
2)      if (r==null)
            return;
3)
4)      print(r.data)
5)      pre(r.left)
6)      pre(r.right)
7) }
```



→ pre(root)

1 2 4 5 3 6



In → left
→ root ←
right

In :→ 4 8 2 5 1 9 6 3 7 10

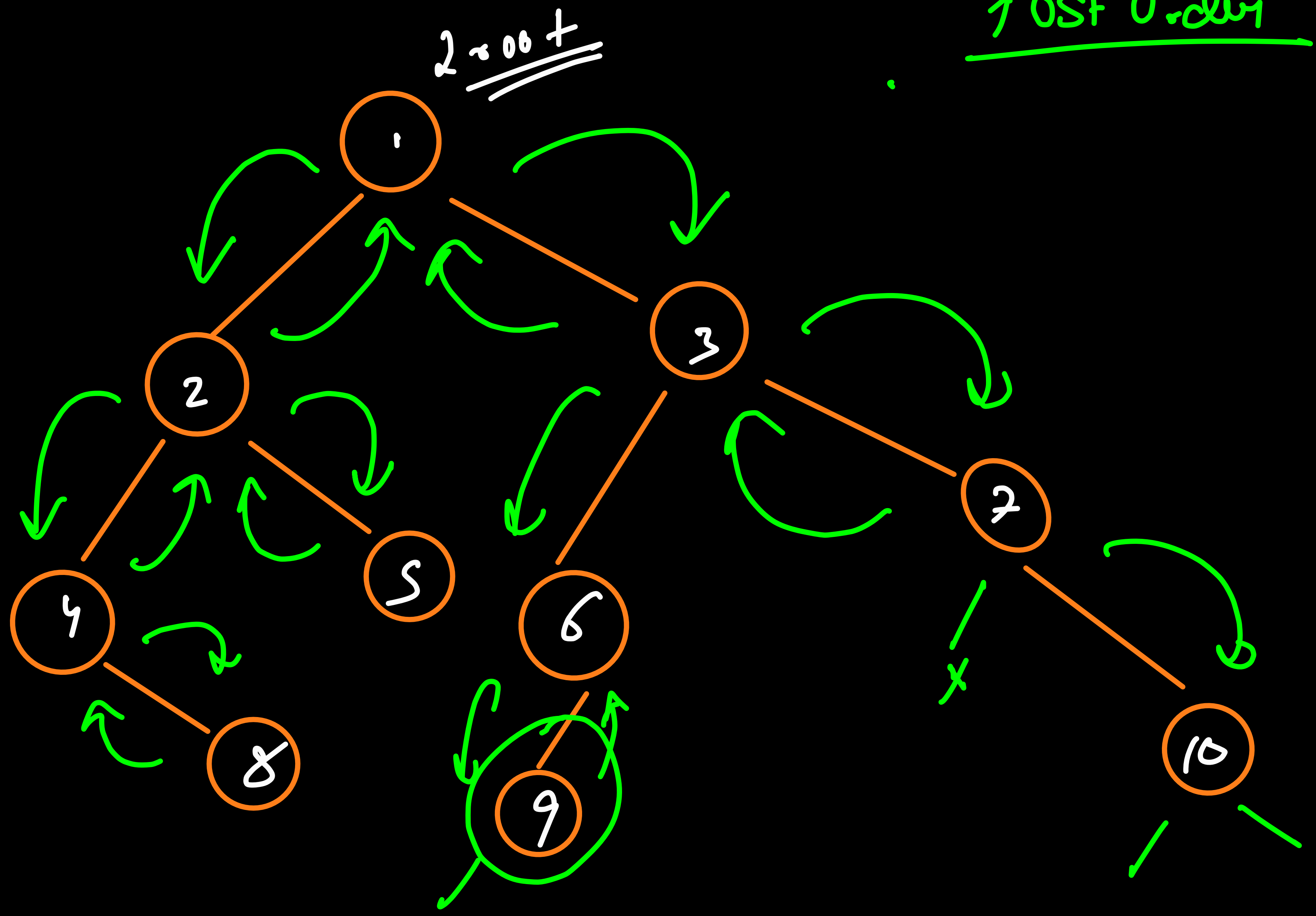
In (r) $\longrightarrow$ In (r.left)
print (r.value)
In (r.right)

performs inorder traversal of the tree rooted at r

r == null
return

Post Order

{ left
right
root



Post → 8 4 5 2 9 6 10 7 3 1

JOIN THE DARKSIDE

post (r) $\Rightarrow$ post (r.left)

post (r.right)

print (r.value)

performs postorder

traversal on a tree

rooted at r

r == null

↳ return

DFS → Pre
→ in
→ post

} → time → $O(n)$
Space → $O(h)$

Reverse Pre → root → right → left
" In → right → root → left
" Post → right → left → root

Q →

Maximum Element

Minimum Element

Sum of all elements

Search an element in a BT

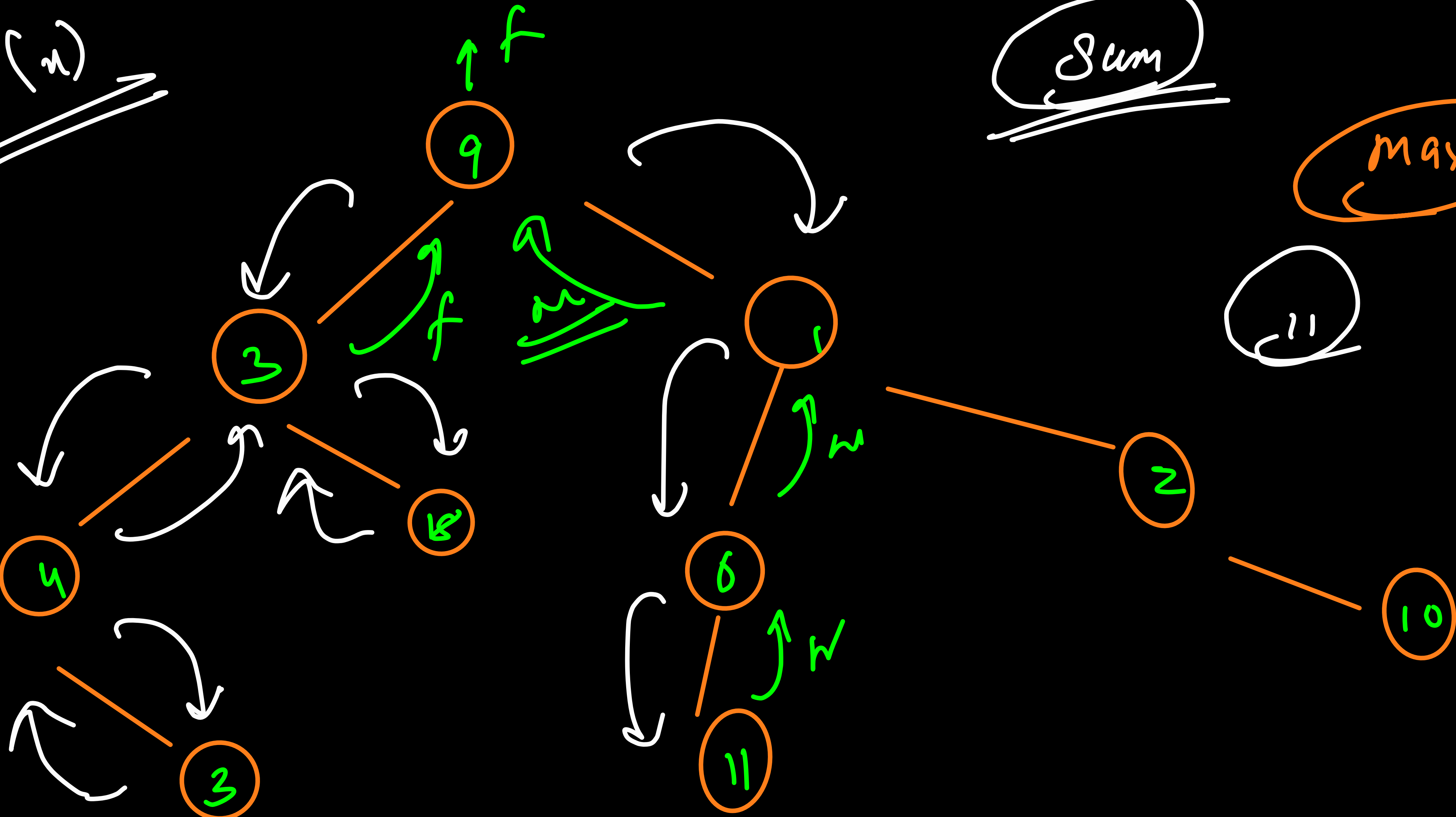
Pre

Steal(n)

Sum

max

O(n)



11

sum_ = 0

$$f(r, x) = (r.val == x) \text{ or } f(r.left, x) \text{ or } f(r.right, x)$$

Searches whether x is present in any nodes rooted at r

r == null → false

$f(r) = \max[\,r.val,\ f(r.left),\ f(r.right))$

max of all nodes rooted at r

$(r == null)$

$\hookrightarrow$ -Infinity

$f(r) = r.val \cancel{\#} + f(r.left) + f(r.right)$

function return sum of all the nodes rooted at r.

(r == null)

↳ return 0;

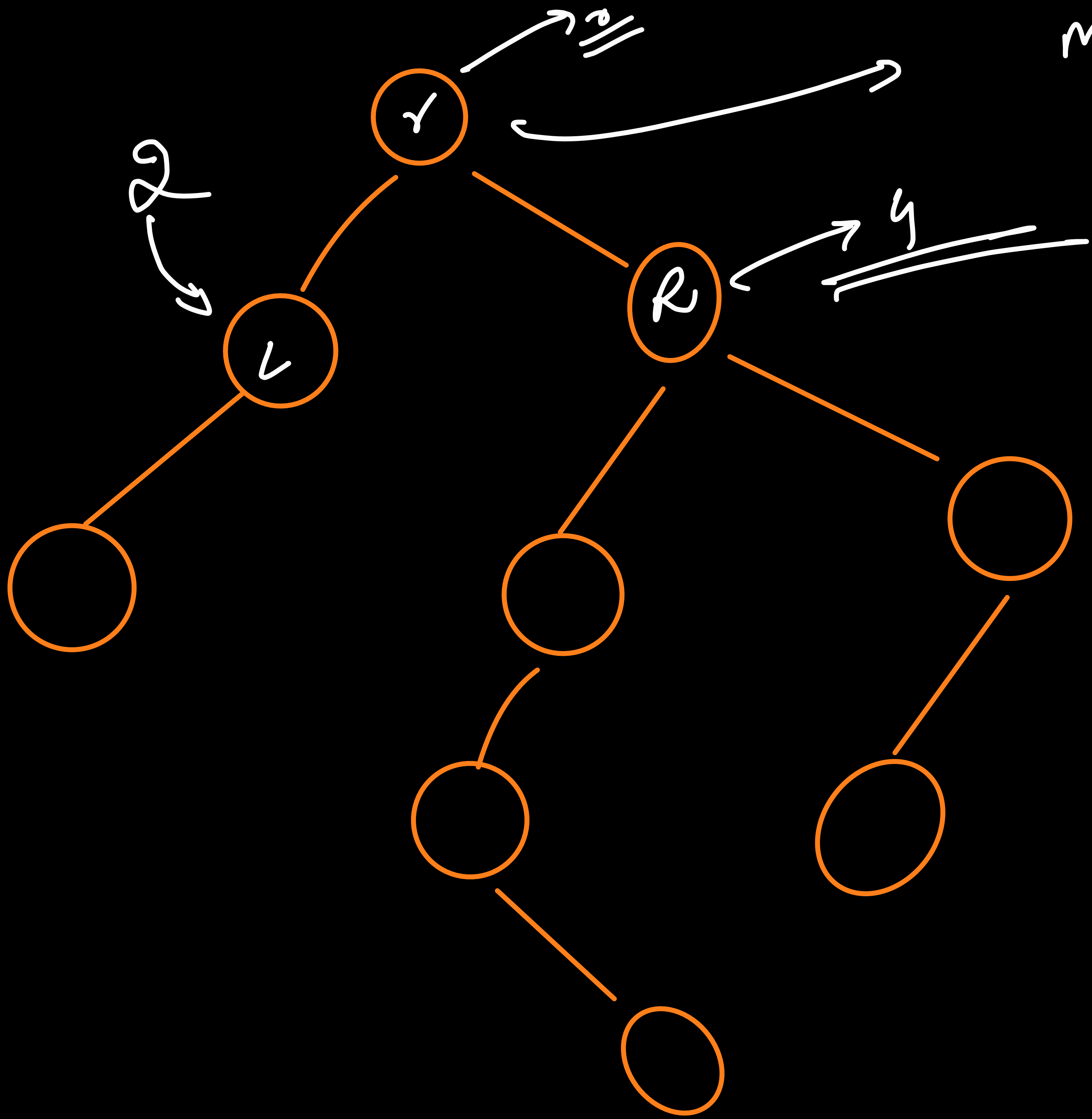
```
sum_ = 0

pre(r) {
    if (r == null) return

    sum_ += r.val;

    pre(r.left)
    pre(r.right)

}
```



$\max(2,4)+1$

$f(r) = \max\left(f(r.\text{left}),\ f(r.\text{riget})\right) + 1$

longest path of tree rooted r

Max depth

Height

Base Case

(r == null)

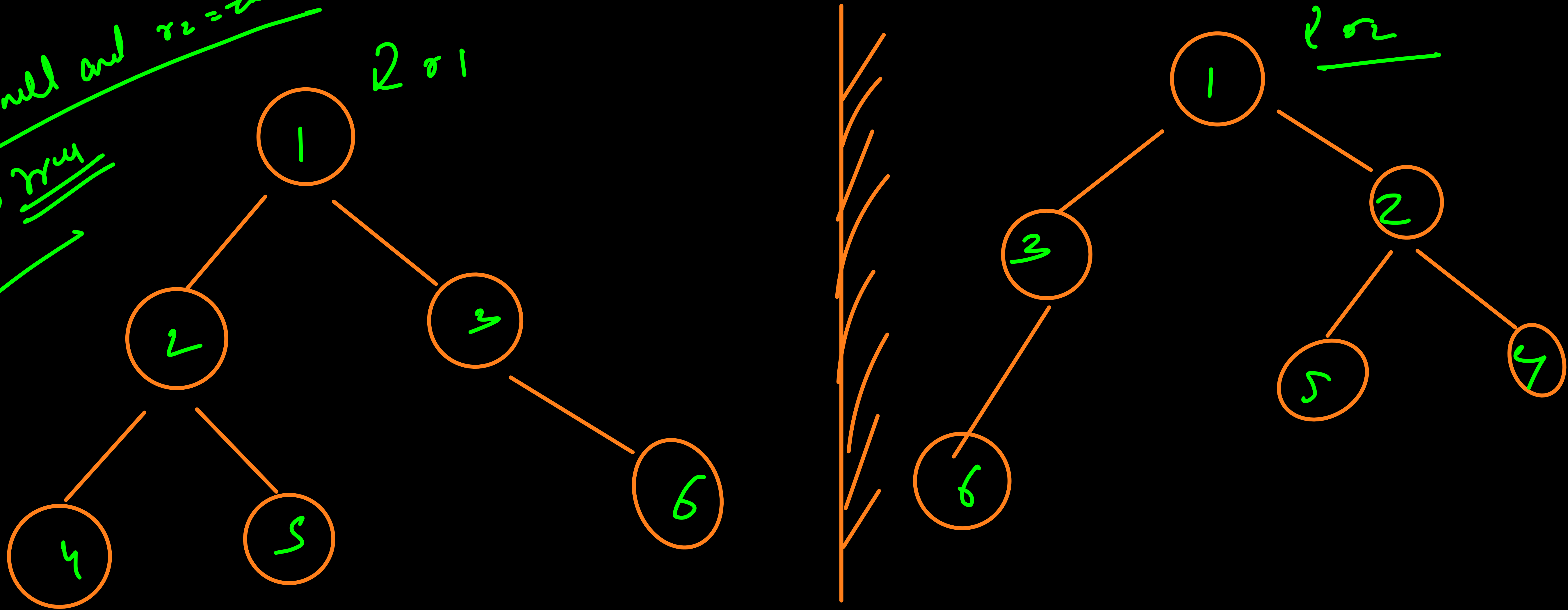
0

Base

$r_1 == null$ and $r_2 == null$

↳ true

↳



$$f(r_1, r_2) = (r_1.val == r_2.val) \text{ and } f(r_1.left, r_2.right) \text{ and } f(r_1.right, r_2.left)$$

↓

while r_1 & r_2 are mirror image

$f(\overset{\uparrow}{root}, \overset{\uparrow}{root})$

$O(n)$

$O(n)$