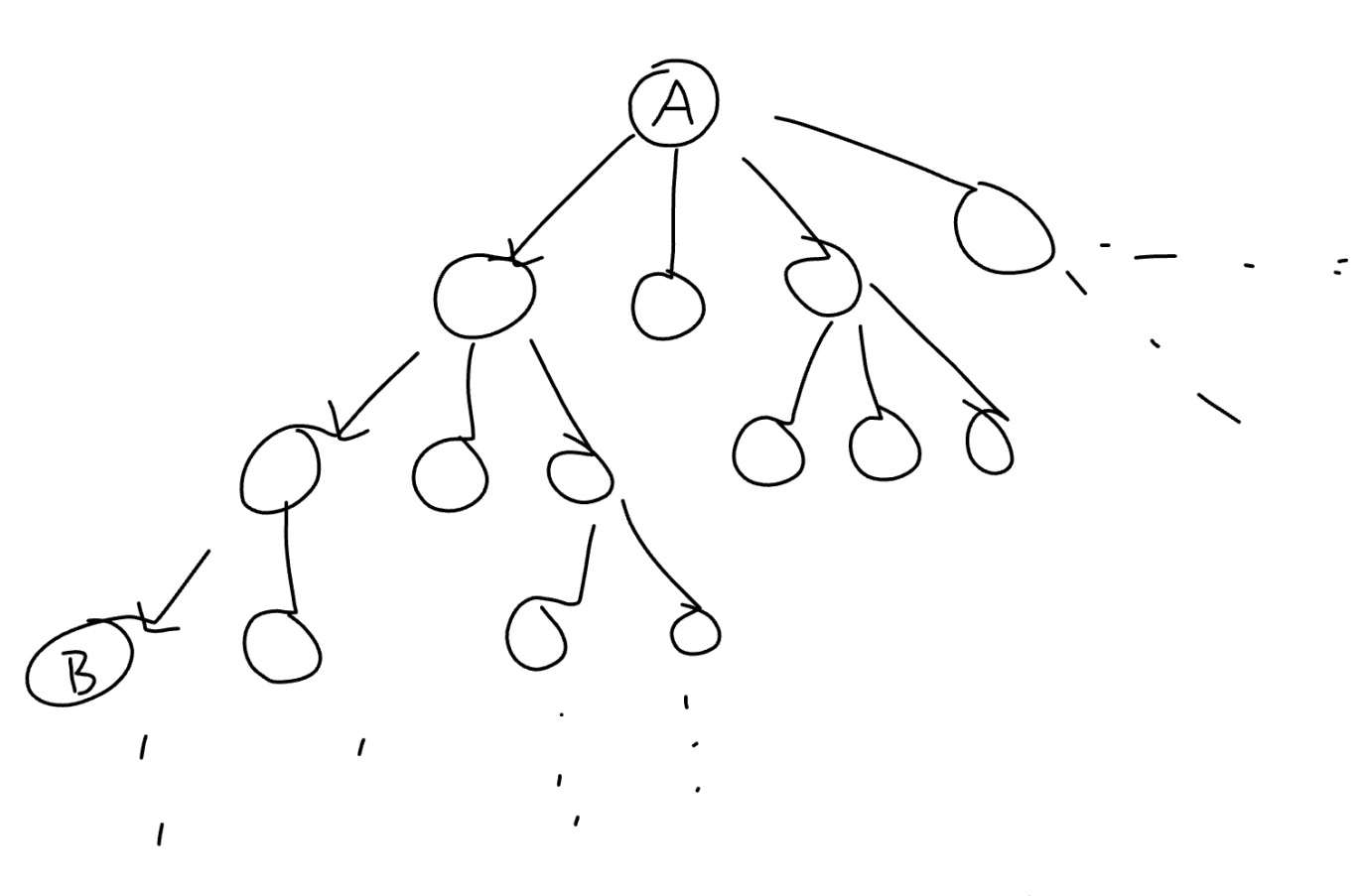
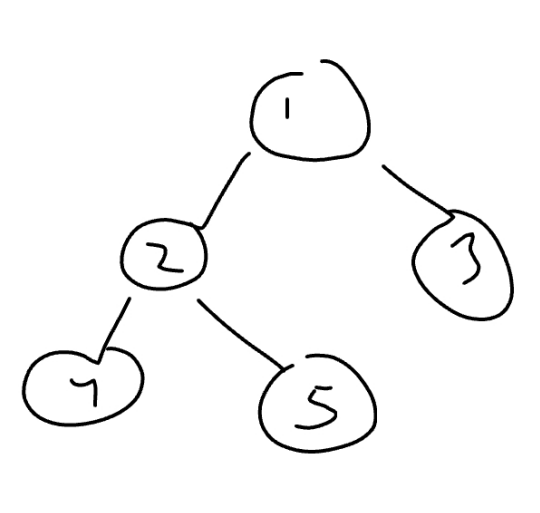




Binary Tree: Atmost 2 child nodes for all the nodes.

Binary Tree

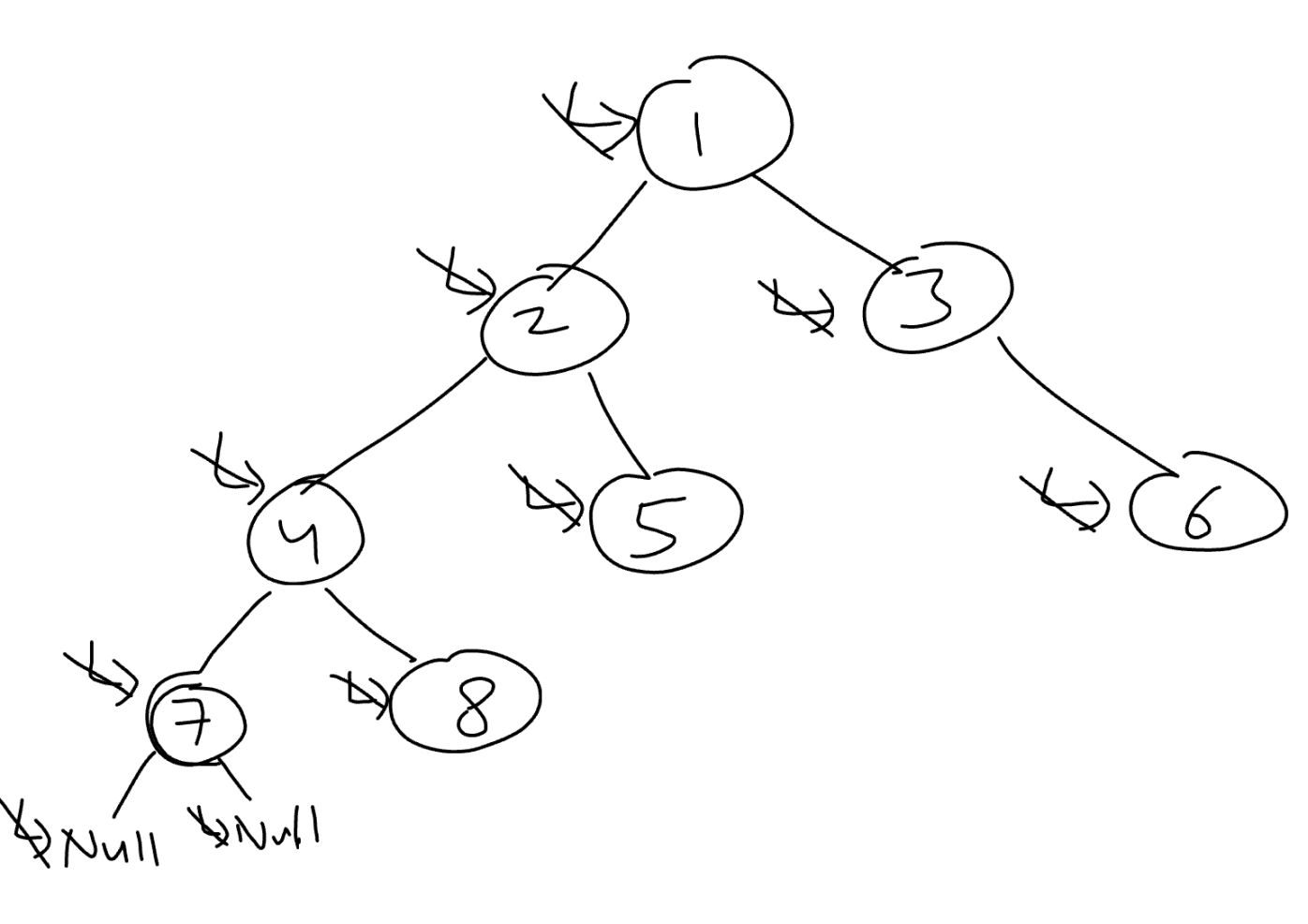
Node {
data
Pointer1 → left
pointer2 → right
}

Generic Tree

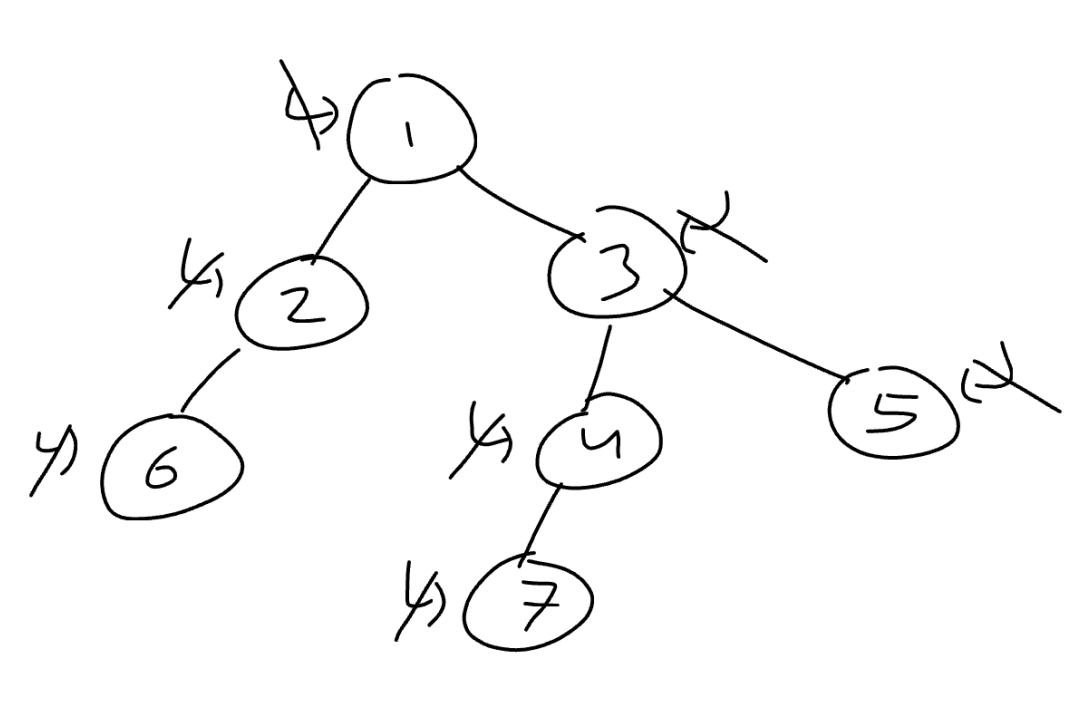
Node {
data
list of pointers
↓
children
}

Preorder Traversal

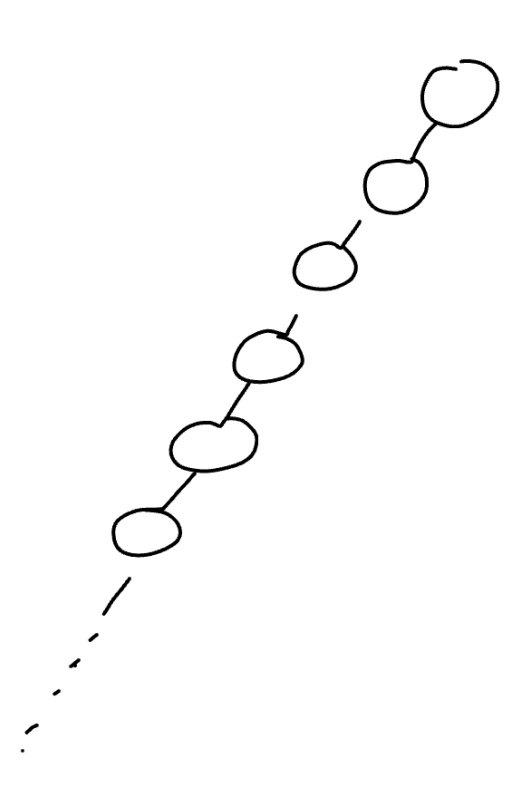
→ Visit the current node
→ Traverse the left subtree
→ Traverse the right subtree


1
2
3
4
5
6
7
8
Null
Null

preorder: [1, 2, 4, 7, 8, 5, 3, 6]



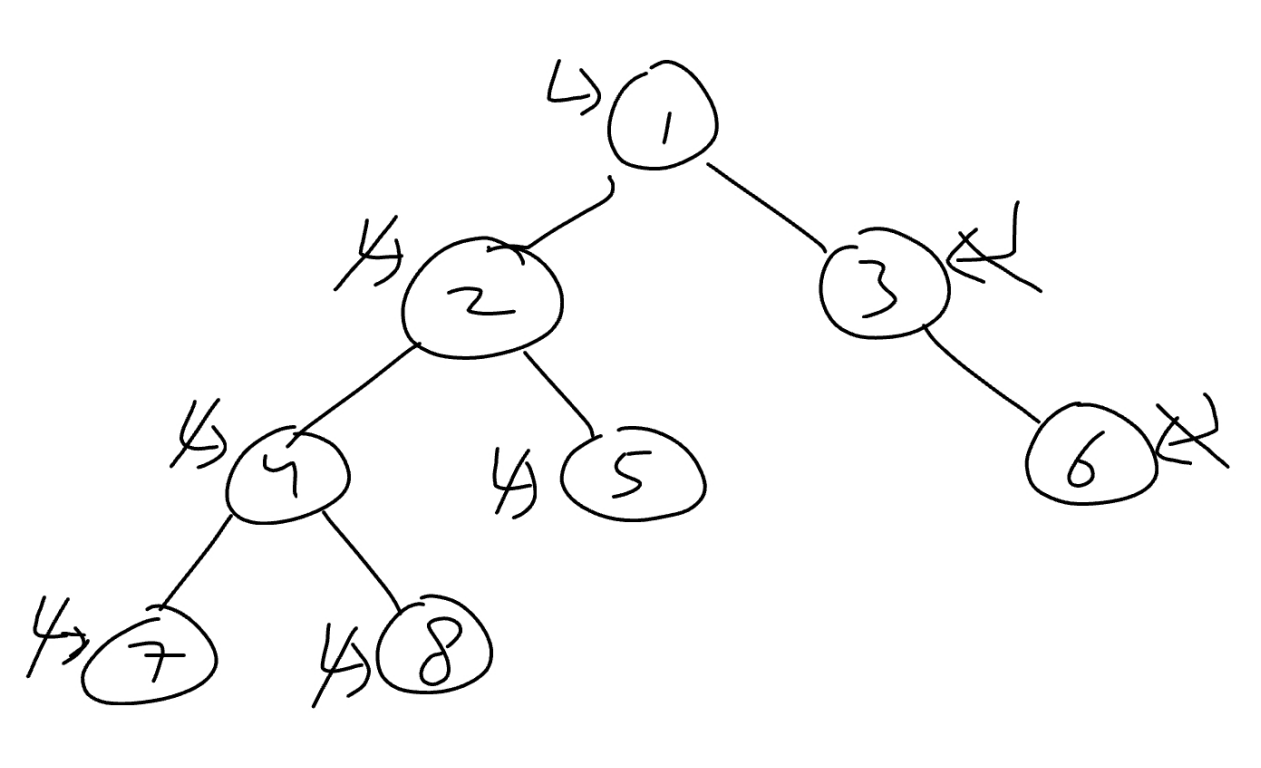
[1, 2, 6, 3, 4, 7, 5]



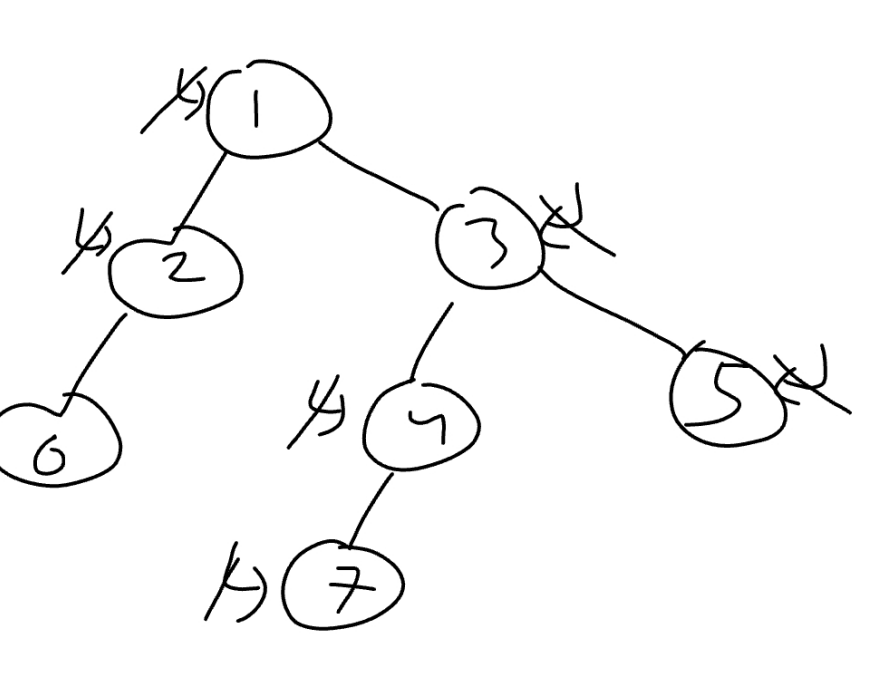
TC: O(N)
AS: O(H)

postorder Traversal

→ Traverse the left subtree
→ Traverse the right subtree
→ Visit the current node



[7, 8, 4, 5, 2, 6, 3, 1]

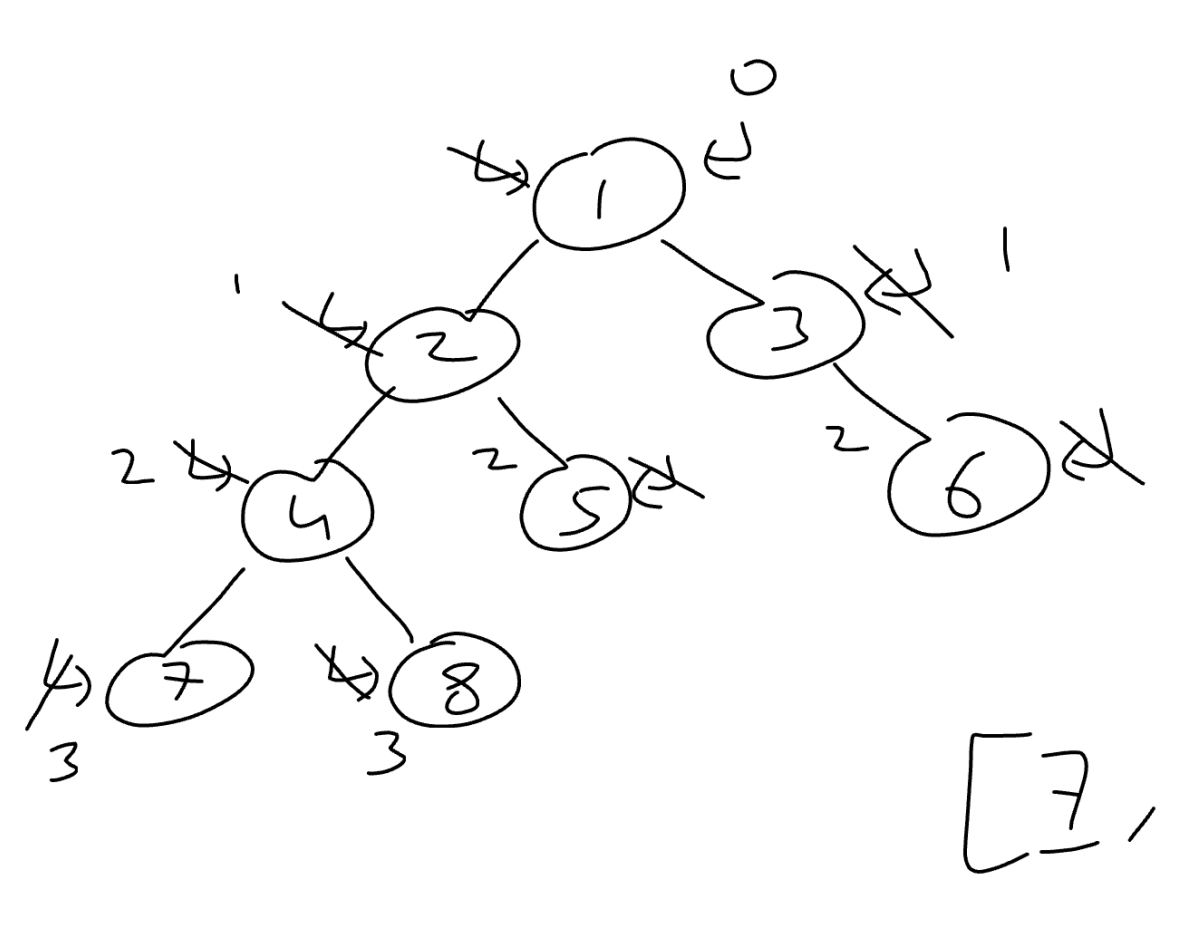


[6, 2, 7, 4, 5, 3, 1]

TC: O(N)
AS: O(H)

Inorder Traversal

→ Traverse the left subtree
→ Visit the current node
→ Traverse the right subtree


0
1
2
3
4
5
6
7
8
1
2
2
2
3
3

[7, 4, 8, 2, 5, 1, 3, 6]

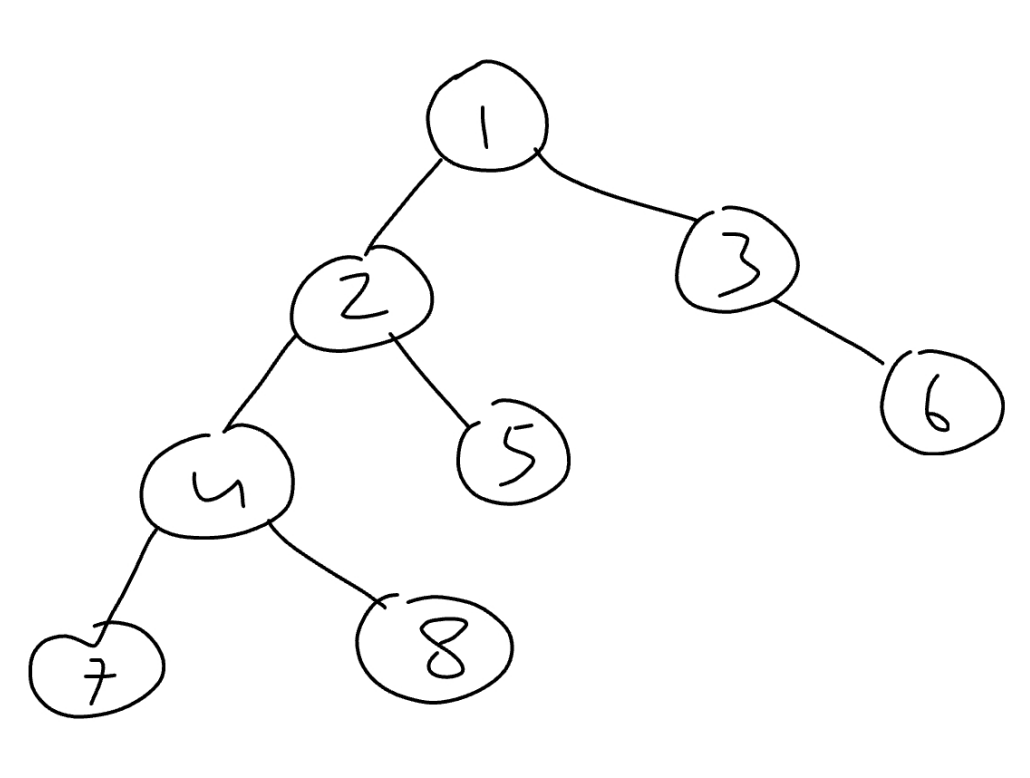
TC: O(N)
AS: O(H)

Level order Traversal

[1, 2, 3, 4, 5, 6, 7, 8]

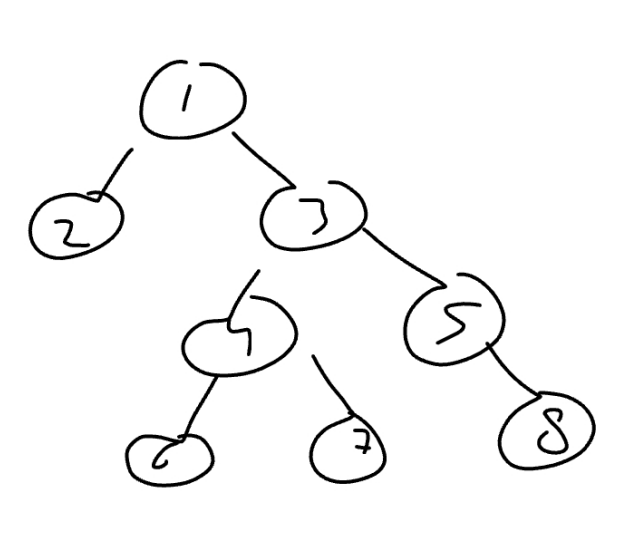
current Level

[[1]
[2, 3]
[4, 5, 6]
[7, 8]]

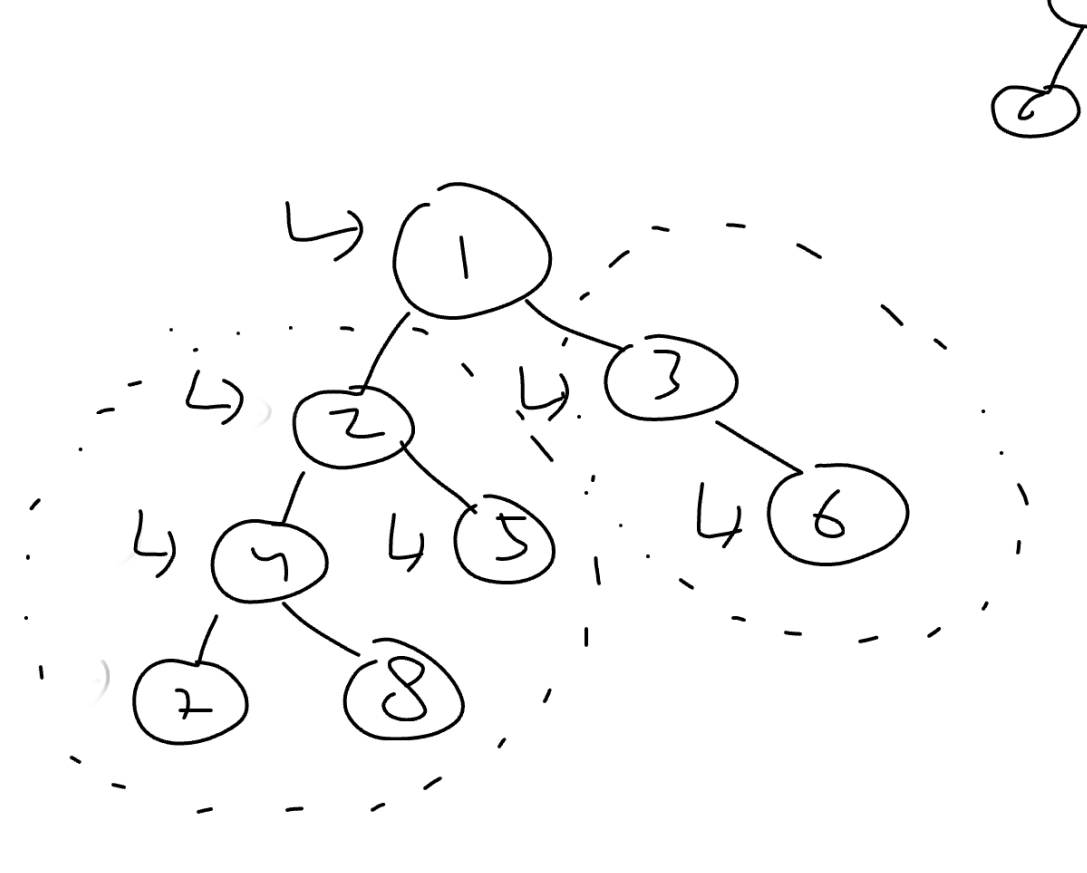


↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1, 2, 3, 4, 5, 6, 7, 8

⇒ Queue



[[1]
[2, 3]
[4, 5, 6]
[7, 8]]



X, N, ~~2, 3,~~ N

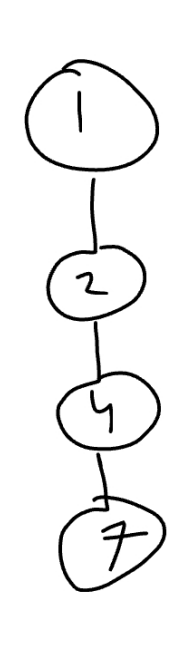
x ~~2, 3, 4, 5, 6, 7, 8~~

x ~~2~~, ~~3~~, ~~4, 5, 6~~

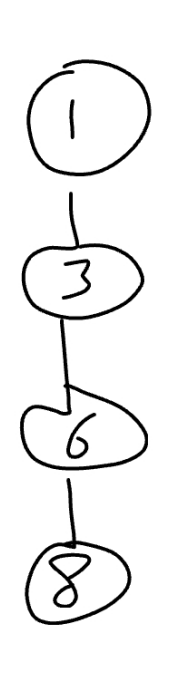
[[1]
[2, 3]
[4, 5, 6]
[7, 8]]

1

Left View

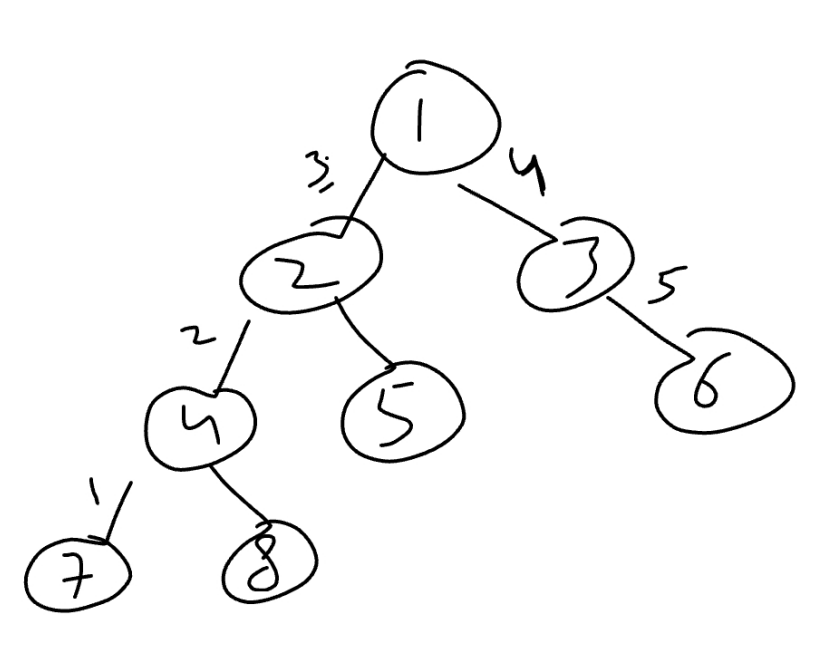


Right View



Height:

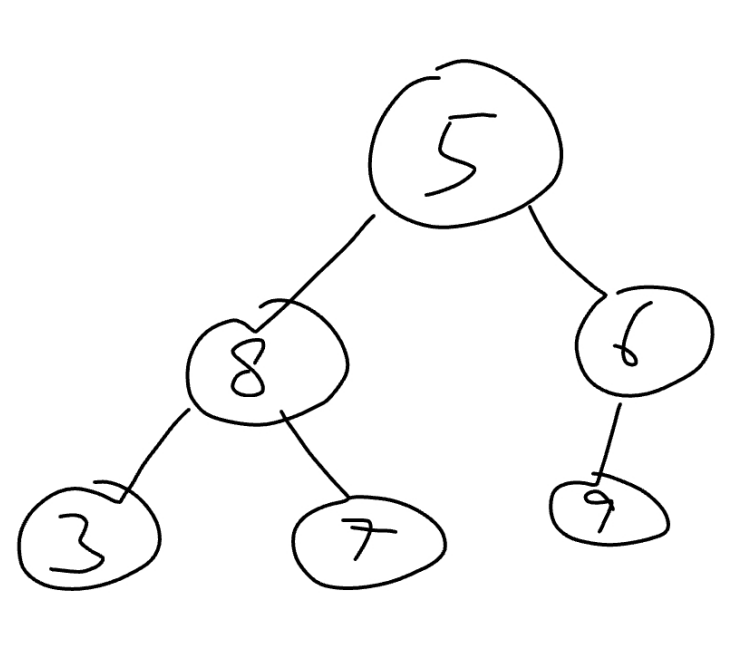
Height(root) = 1 + max(height(root → left), height(root → right))


1
2
3
4
5
6
7
8
1
2
3
4
5

left ht = 3
right ht = 2

ht = 1 + max(3, 2)
= 1 + 3 = 4

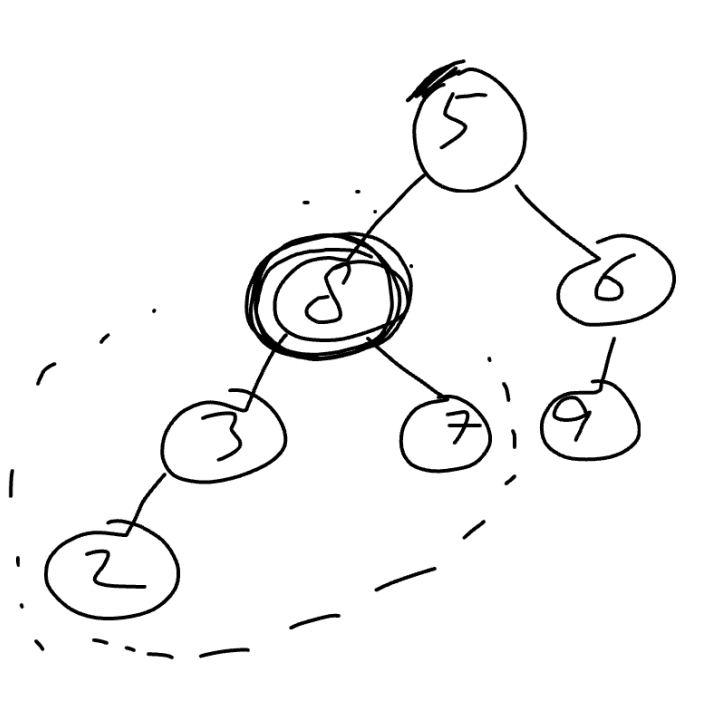
Diameter



Leaf: 3, 7, 9

Len (3, 7) = 2
Len (3, 9) = 4
Len (7, 9) = 4

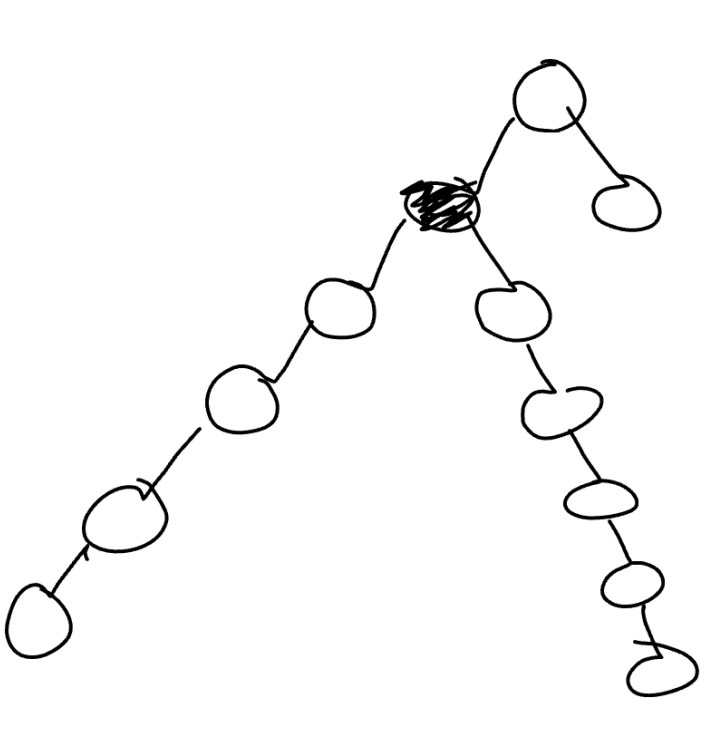
Diameter = 4



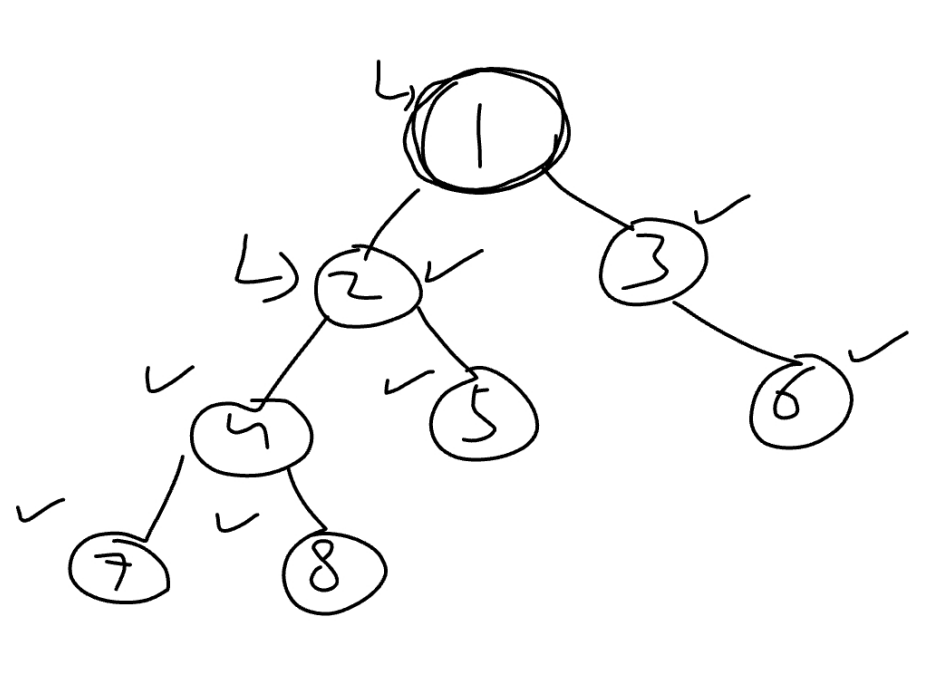
ht(8) = 3
ht(6) = 2

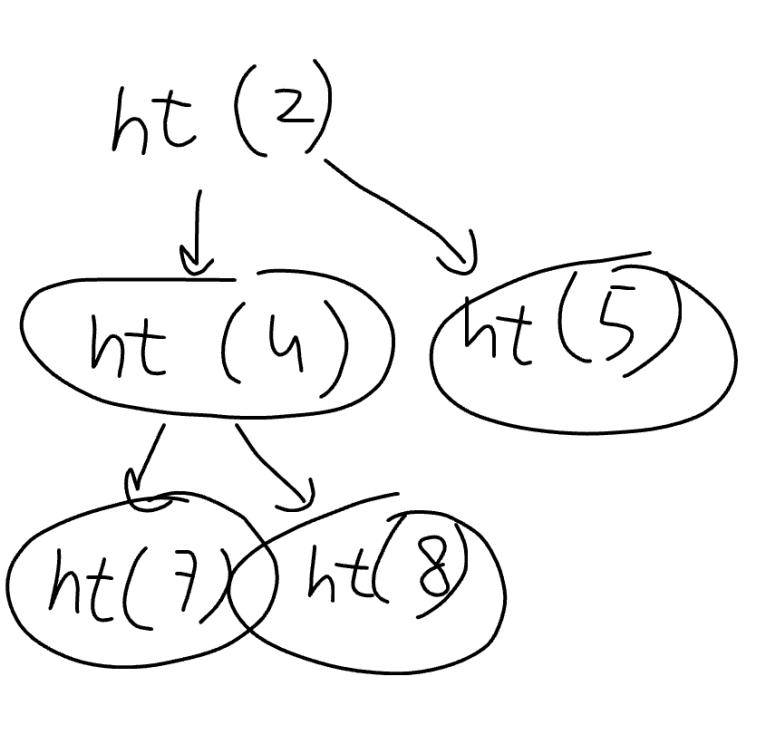
Diameter considering 5 being a turning node = ht(8) + ht(6)

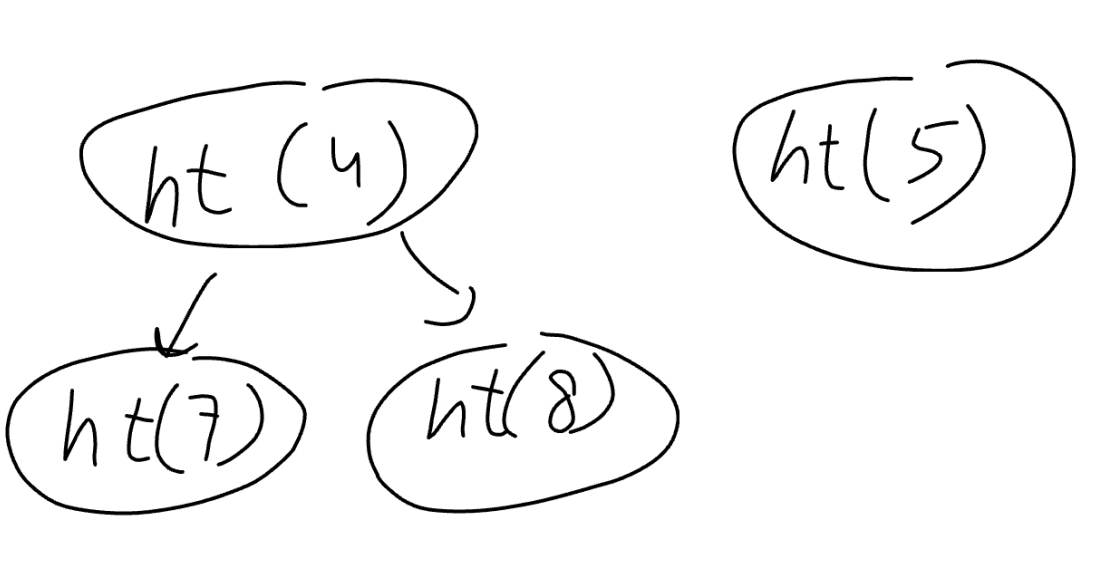
ht(8) + ht(6)



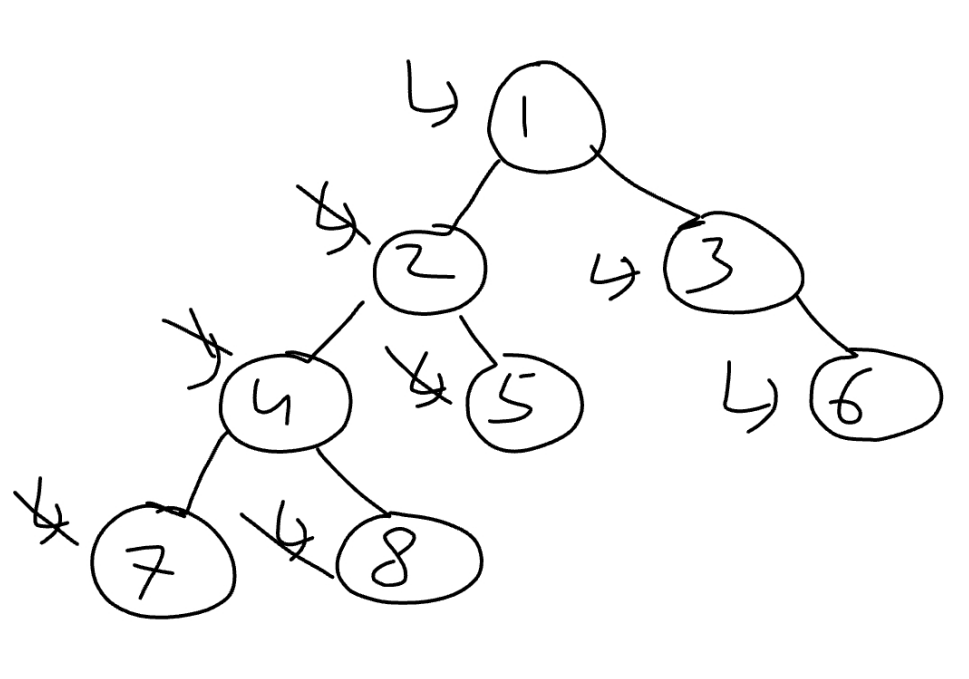
Diameter = Max (left_height + right_height)




ht (2)
ht (4)
ht (5)
ht(7)
ht(8)


ht (4)
ht(7)
ht(8)
ht(5)

heights: Node* → int



n1 = 7
n2 = 5

[1, ~~2~~, ~~4~~, ~~7~~, ~~8~~
~~5~~ 3, 6]

[1, 2, 4, 7]

[1, 2, 5]

n1 = 2
n2 = 5

[1, 2]
[1, 2, 5]

path = currentPath

TC: O(N)
AS: O(H)

case-1: One node is in the left subtree and One node in the right subtree
⇒ current node is the LCA.

case-2: Both the nodes are in the left subtree.
⇒ LCA is in the left subtree

case-3: Both nodes in the right subtree
⇒ LCA is in the right.