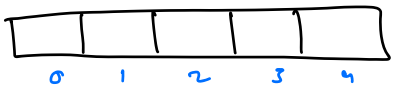
Trees Basics

AGENDA:

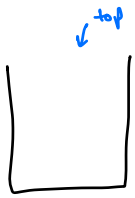
- ✓ Basics of Trees
- ✓ Implementation
- ✓ Traversals
- ✓ Size
- ✓ Height
- ✓ Basic Problems
- ✓ Use cases of Trees

Linear Data Structures

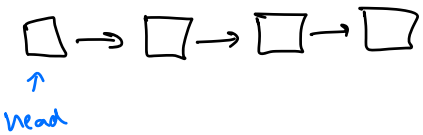
Arrays



Stacks



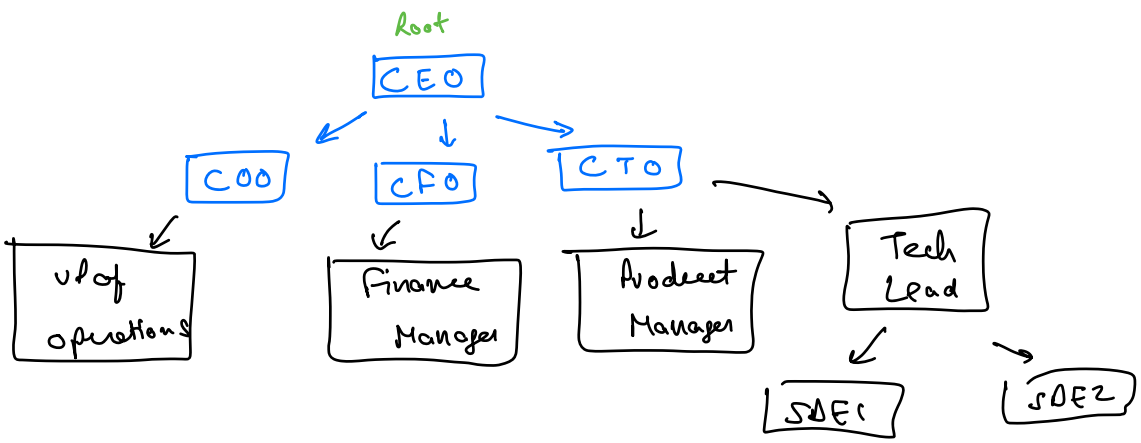
Linked List



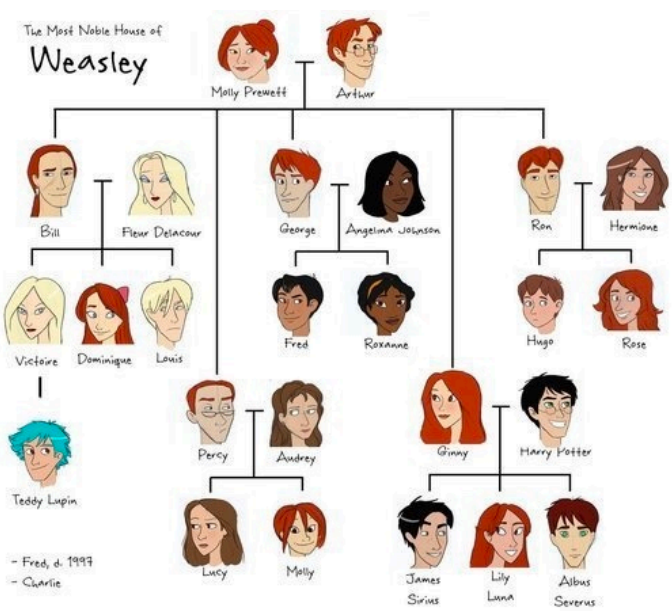
Queues

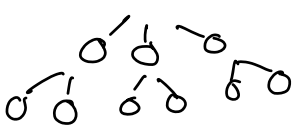
Hierarchial Data Structures

Example: Company Hierarchy

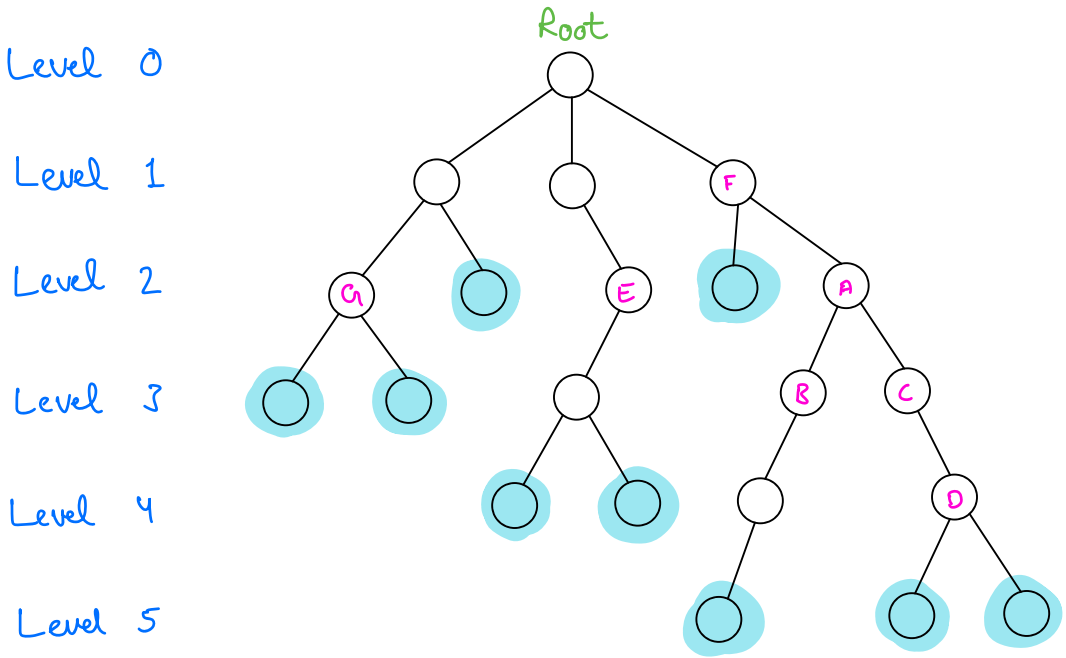


Example: Family Tree





Trees



Root, F, A are ancestors of C

C is the descendant of Root, F, A

B & C are siblings

Naming Conventions

- Parent & Child
- Ancestor & Descendant
- Siblings / Nodes at same level
- Leaf Nodes
- Root Node

B, C are children of A

A is the parent of B, C

Nodes without children

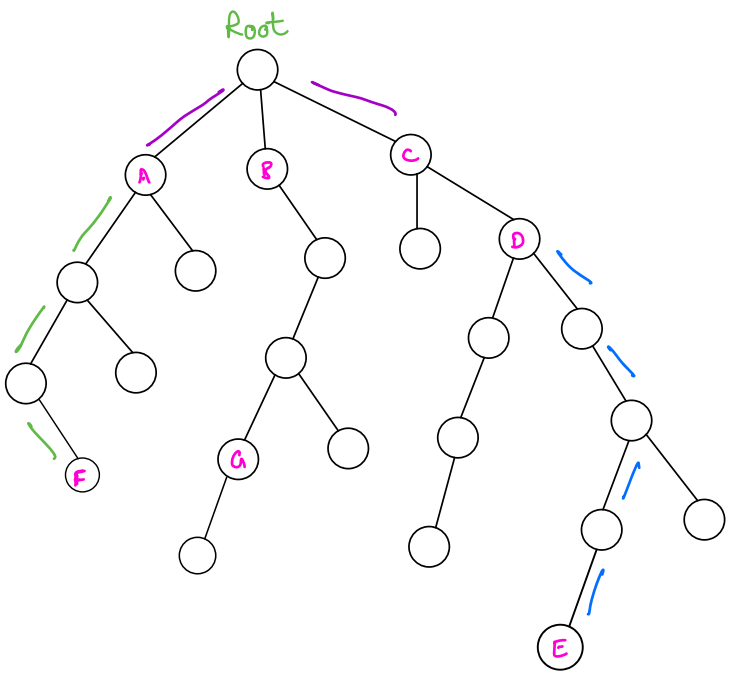
Starting node of the tree

The only node without a parent

Properties of Trees

Path(X, Y)

No of edges from X to Y



Path(A, F): 3

Path(B, G): 3

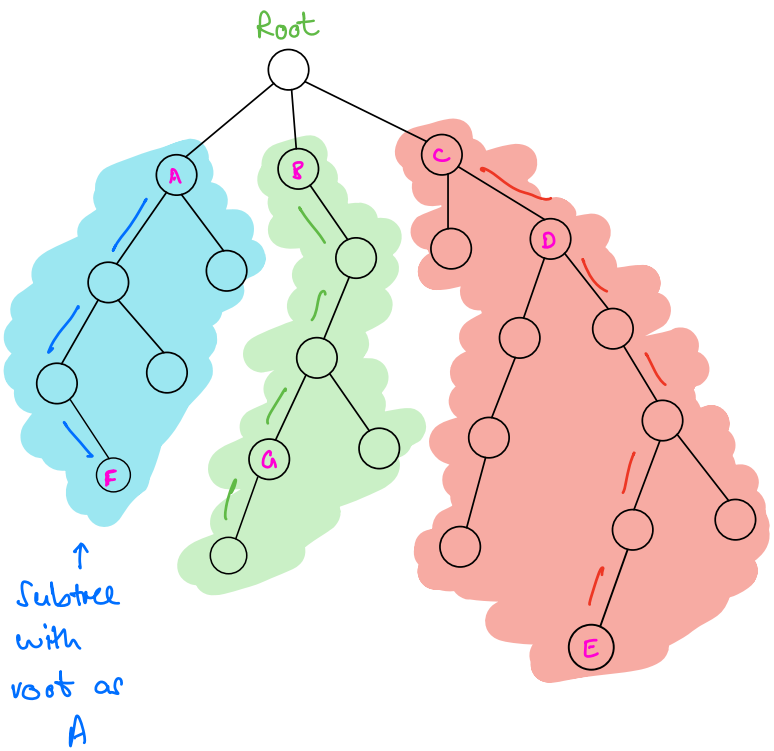
Quiz 1

Path(D, E): 4

Path(A, C): 2

Height(node)

Length of largest path from the Node to any of its descendant leaf nodes



Height(A): 3

Height(B): 4

Quiz 2

Height(C): 5

Height(root): 1 + max(height of child nodes)

= 1 + max(3, 4, 5) = 1 + 5 = 6

Height(node) = 1 + max(height of its child nodes)

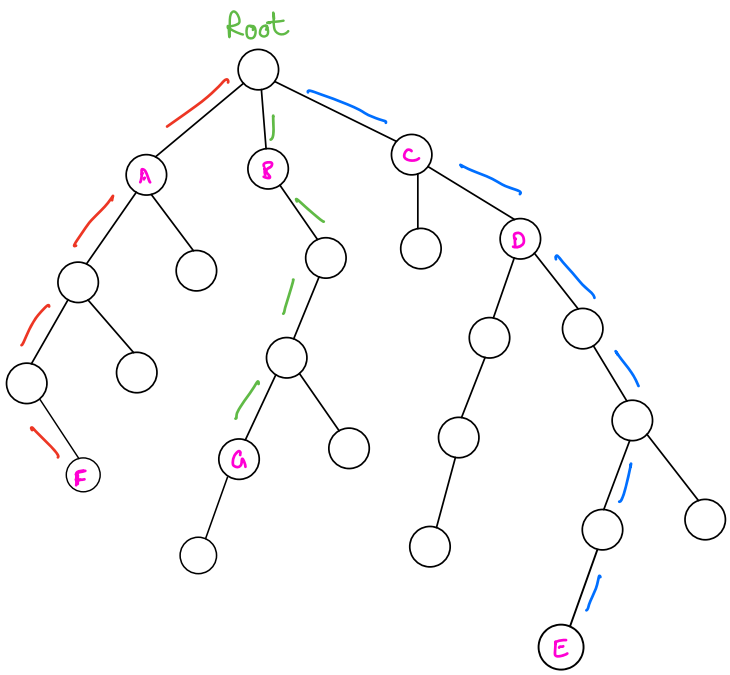
Important:

Height(Leaf Node) = 0

Quiz 3

Depth of a Tree

Length of path from root node to given node



Depth(G): 4

Quiz 4

Depth(E): 6

Depth(F): 4

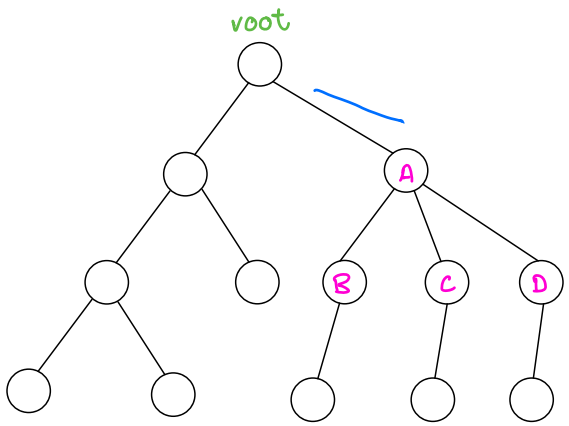
Example

Depth(A) = 1

Depth(B) = Depth(C) = Depth(D)

= 1 + Depth(A)

= 2



Observation

If depth(node) = k

Depth of its child nodes = k+1

Height of Tree = Height (root)

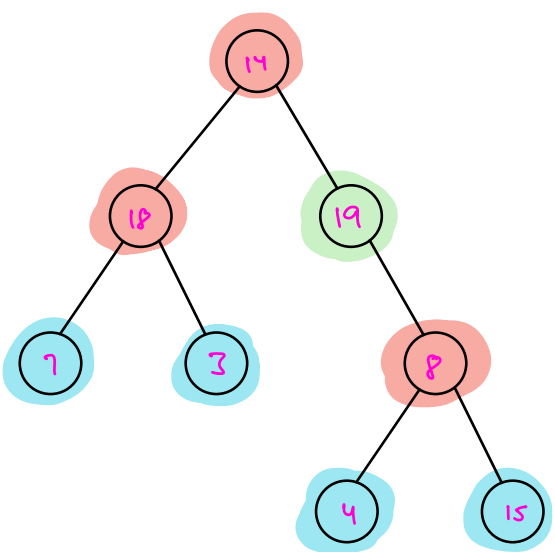
Depth of the tree = Max depth of all leaf nodes = Height of Tree

Binary Tree

Every node can have atmost 2 children

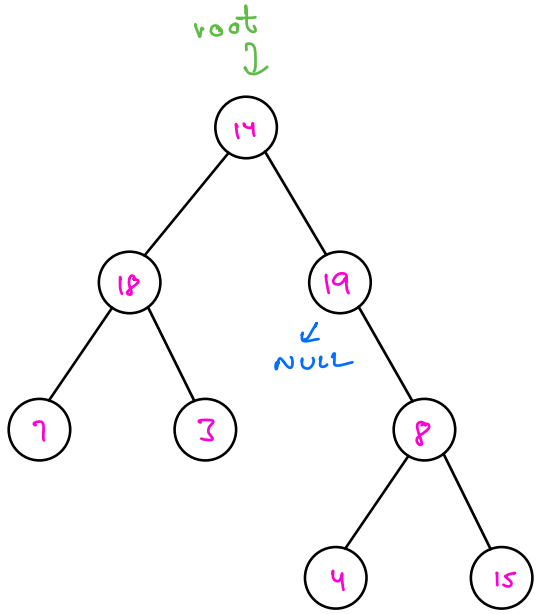
0, 1, 2

Example



- 2 children
- 1 child
- 0 child (Leaf Nodes)

Node structure




```
class Node {
    int data
    Node left
    Node right

    constructor( int x) {
        data = x
        left = NULL
        right = NULL
    }
}
```

root . data → 14

Node t = root . left → 18

t . data → 18

t . left → 7

t . right → 3

Node t_2 = root . right → 19

t_2 . data → 19

t_2 . left → NULL

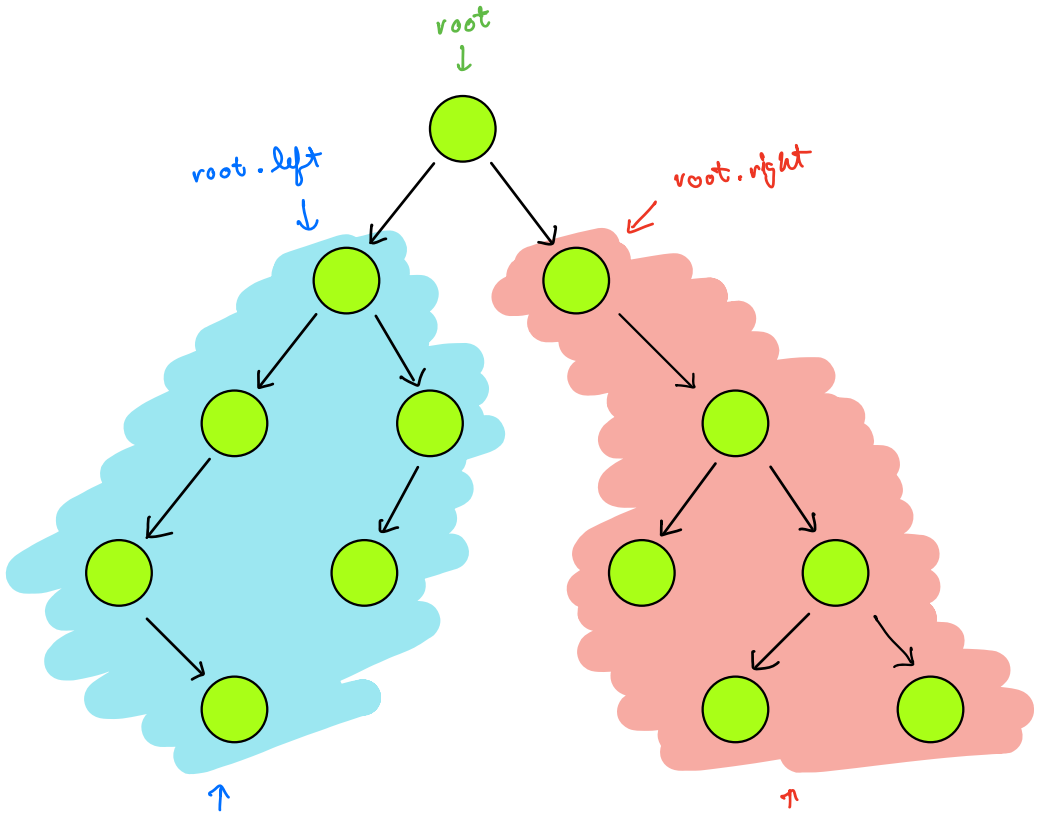
t_2 . right → 8

Important Note:

Construction of a binary tree will be covered later on in the Advanced Batch. For now, a ready-made tree will be given to you in all the problems.

Most used Technique

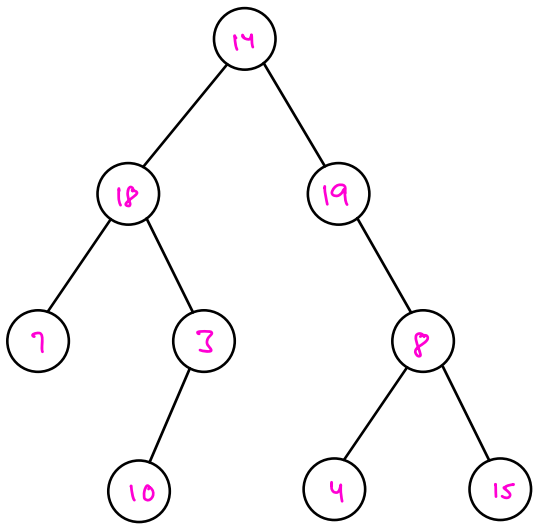
⇒ Recursion
(90% problems)



Break till 10:11 PM

Tree Traversals

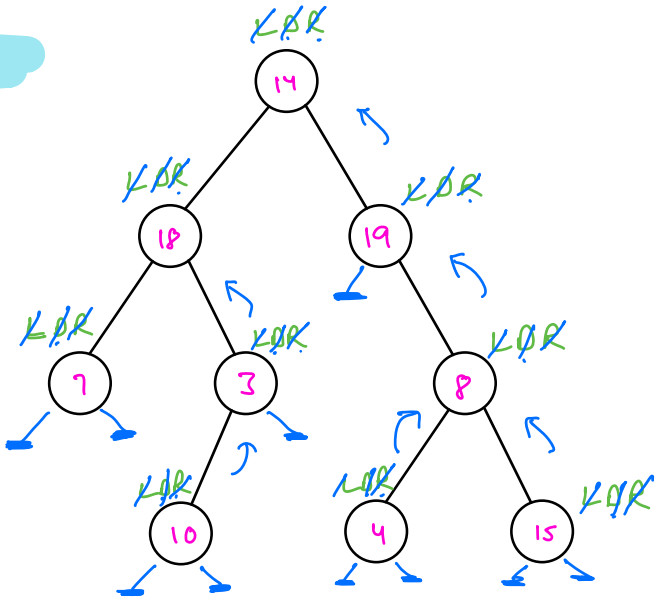
1. Pre order : D L R
2. In order : L D R
3. Post order : L R D
4. Level order traversal
   ↳ Advanced Batch



Inorder Traversal

LDR

7, 18, 10, 3, 14, 19, 4, 8, 15

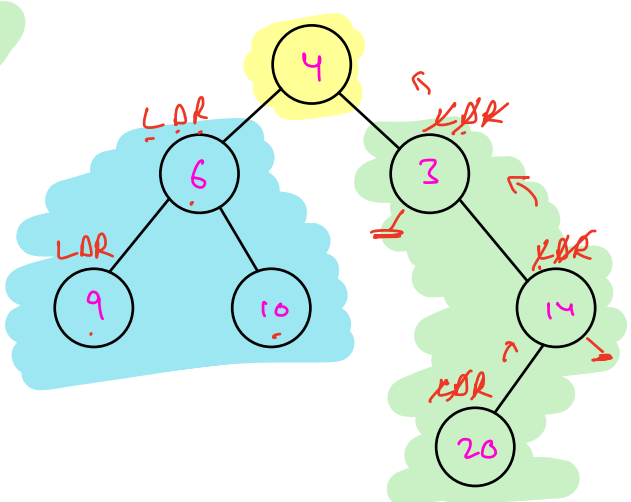


LDR

Example

Quiz 5

9, 6, 10, 4, 3, 20, 14



In order Traversal Implementation - Recursive

LDR

Assumption

Given a node, inorder() should print the inorder traversal of its tree.

Main Logic

→ Print left subtree in inorder

→ Print root.data

→ Print right subtree in inorder

```
void inorder (Node root) {
    if (root == NULL)
        return

    inorder (root.left)
    print (root.data)
    inorder (root.right)
}
```

void / None

← Base Case

← Main Logic

N nodes

TC: O(N)

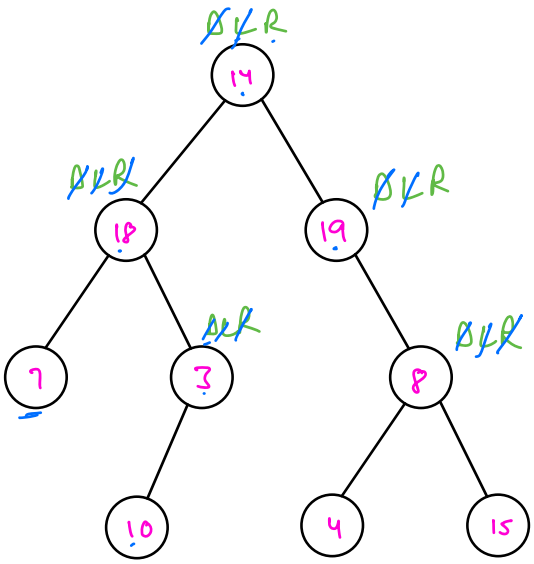
Preorder Traversal

DLR

Quiz 6

14 , 18 , 7 , 3 , 10 , 19 , 8 , 4 , 15

Option A



Pre order Traversal Implementation - Recursive

DLR

Assumption

preorder(root) prints the subtree of root in preorder traversal.

Main Logic

→ Print root.data
→ Print left subtree in preorder
→ Print right subtree in preorder

```
preorder( Node root ) {
    if (root == NULL)
        return
```

← Base Case

```
    print(root.data)
    preorder(root.left)
    preorder(root.right)
```

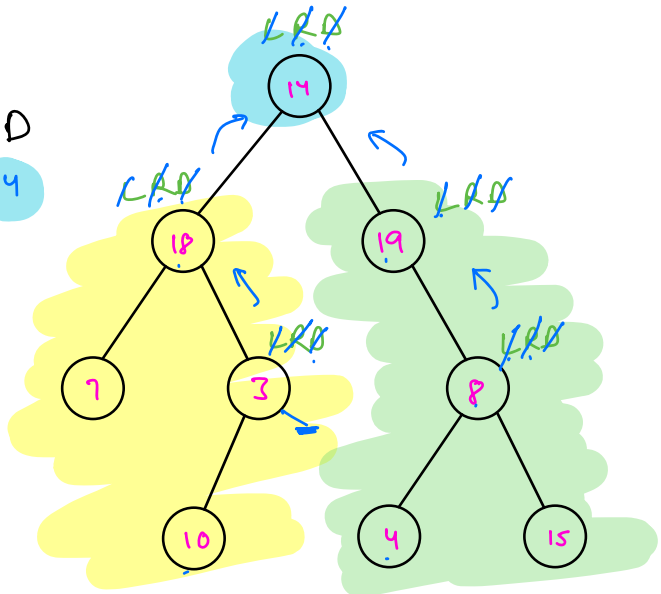
← Main Logic

```
}
```

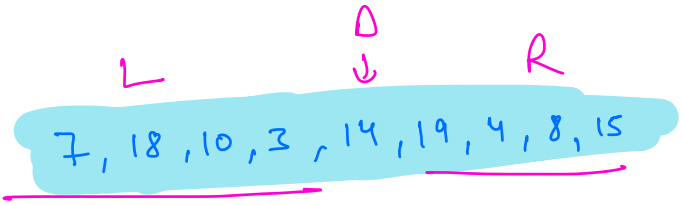
Postorder Traversal

LRD

Quiz 7



Inorder



Post order Traversal Implementation - Recursive

↳ TODO

2-3 lines change

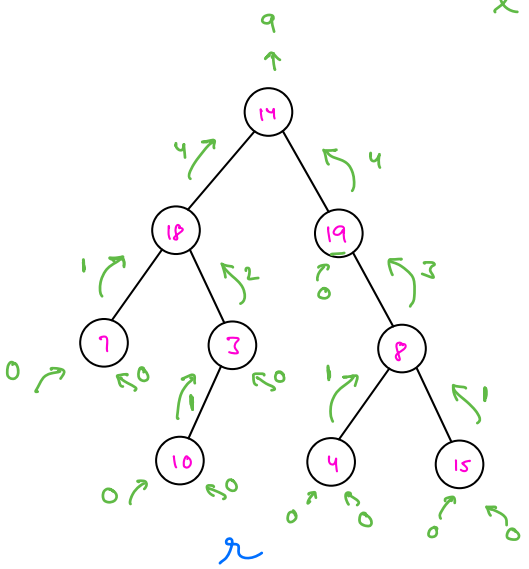
Important Note:

In your assignments / homework, it is expected to write iterative codes. However for now, you may write & submit recursive codes only.

Size of a tree

l + r + 1

No of nodes



l r

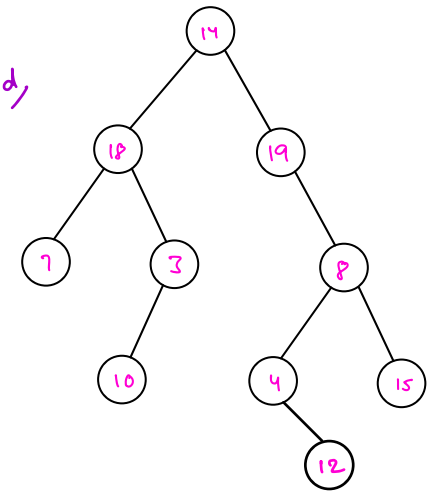
size of tree = size of left subtree + size of right subtree + 1

```
int size ( Node root) {
    if ( root == NULL )
        return 0

    l = size ( root.left)
    r = size ( root.right)
    return l + r + 1
}
```

Height of a tree

height(node) = 1 + max (
l → height of left child,
r → height of right child
)




```
int height (Node root) {
    if (root == NULL)
        return -1
    l = height(root.left)
    r = height(root.right)
    return 1 + max(l, r)
}
```

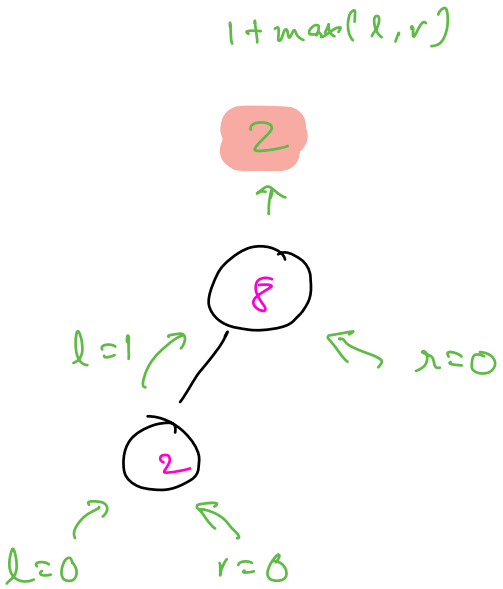
Important Note:

In your assignment, height is calculated as per no of nodes, not as per the number of edges. Thus some changes (just a one line change honestly) will be needed to adjust for it.

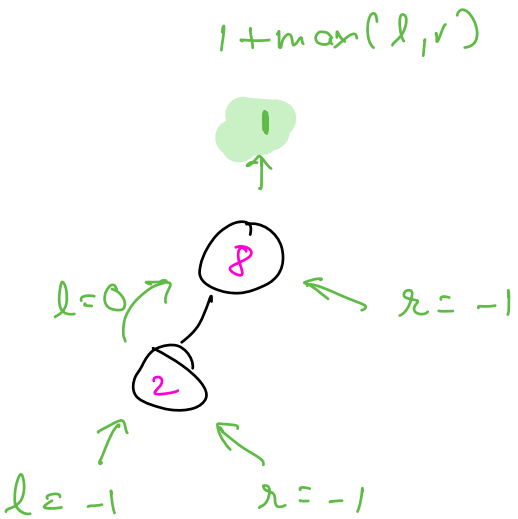
```
if (root == NULL)
    return 0
```

Height = 1

Height (Leaf node) = 0



```
if (root == NULL)
    return -1
```



Extensions of Binary Trees

- BST
- AVL Trees
- Red Black Trees
- B Trees
- B+ Trees
- Priority Queues
- Tries

} Competitive

} Directories

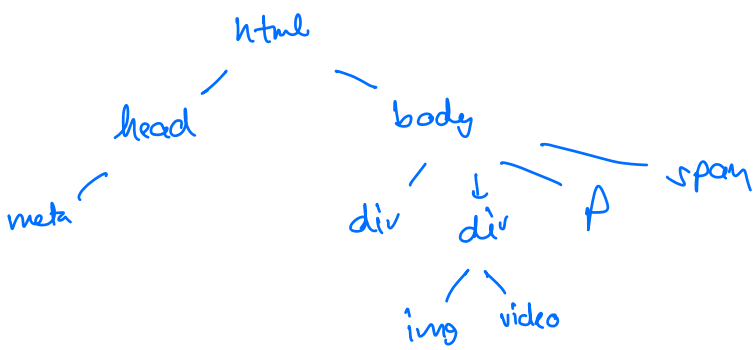
Applications

- Databases
- Directory System
- Browser DOM ←
- Hash Trees
- ...and many more

HashMap, HashSet

↓ ↓

TreeMap TreeSet



Advanced

- Arrays – Arrays
- Bit Manipulation – Bit
- Maths
  - Modular – %
  - GCD
  - Prime Numbers
  - Permutations & Combinations
- Recursion – Recursion
- Sorting
- Binary Search
- 2 pointer
- Linked List – Linked List
- Stacks, Queues, Deques
- Trees – Trees Basics
- Tries
- Heaps
- Hashing – Hashing
- Greedy Algorithms
- Backtracking → Recursion
- Dynamic Programming ↗
- Graphs

Computer Subjects

- Networking
- Operating Systems
- DBMS

System Design

- OOPS – Classes & Objects
- SOLID Principles
- LLD
- HLD

+ Specialisation

Doubts

Thank You

→ Slack

→ tarun.luthra@scaler.com

→ LinkedIn – tarunluthra123

Thank You Everyone

Good Bye

Have fun