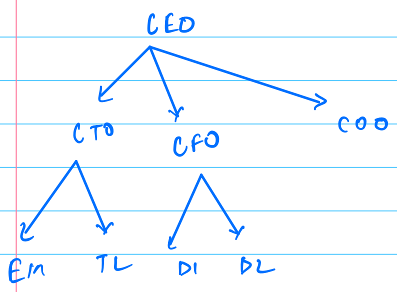
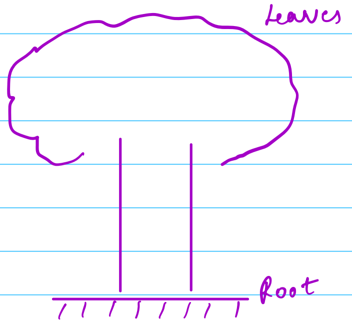
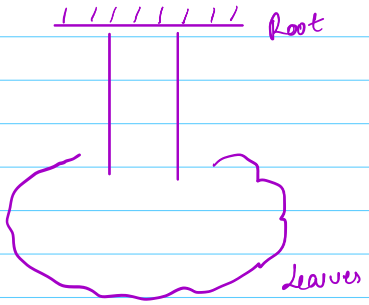
17/1/2024

Trees - 1

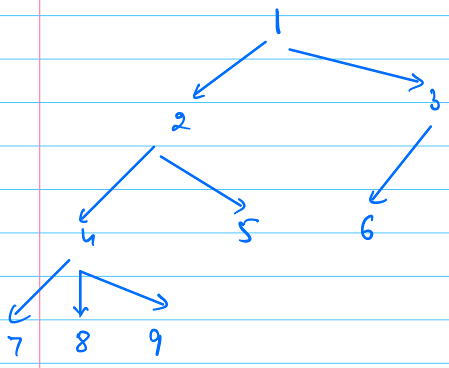
Hierarchical data structure



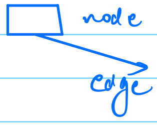


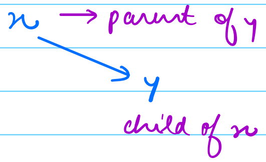


Trees in CS → Inverted tree



data → next pointer

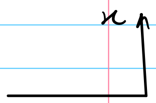




Root → only node without parent
Root → Top most node of tree, it is the tree representative

Leaf → Node without children

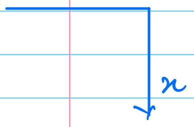
Height → # edges to travel from node x to farthest leaf.



height(2) = 2

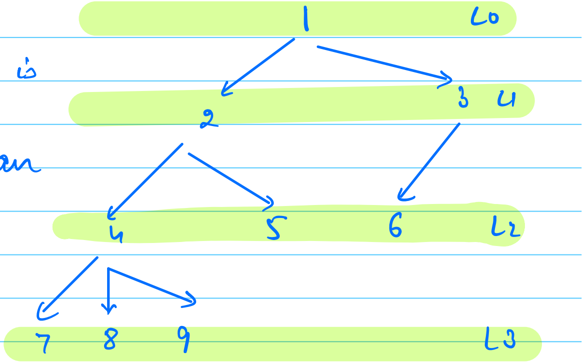
Height of tree = Height of root = 3

Depth/Level → # edges to travel from root to current node x.



depth(2) = 1

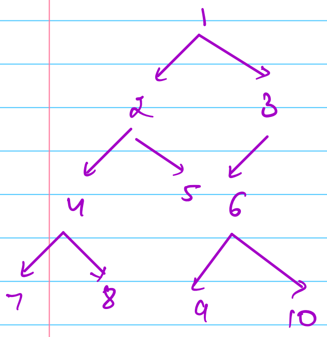
Subtree – subtree of a node x is the part of tree which includes all the nodes that can be travelled from x.



can leaf node be a subtree → Yes

do all nodes have parent → root node.

Binary tree - tree in which $\forall$ nodes, # children = {0, 1, 2}



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

    Node(x) {
      data = x;
      left = right = null;
    }
}
```

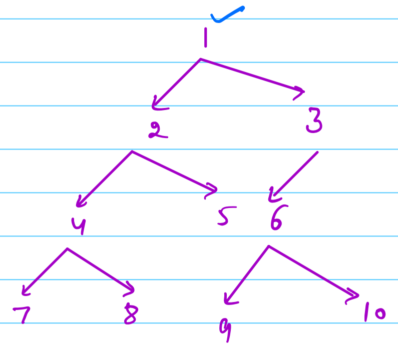
Tree traversals

1>	Preorder traversal	Node	Left	Right
2>	Inorder traversal	Left	Node	Right
3>	Postorder traversal	Left	Right	Node

4> Level order traversal ⟶ next class

1> Preorder traversal

$\underbrace{1}_{N}\ \underbrace{\overset{N}{2}\ \overbrace{4\ \ 7\ \ 8}^{L}\ \overset{R}{5}}_{L}\ \underbrace{\overset{N}{3}\ \overbrace{6\ \ 9\ \ 10}^{L}}_{R}$



```
void  preorder ( root) {
      if (root ==null)return;

   print(root.data);        ⟶ Node
   preorder(root.left);     ⟶ Left
   preorder (root.right);   ⟶ Right

}
```

N = # nodes
H → height of tree

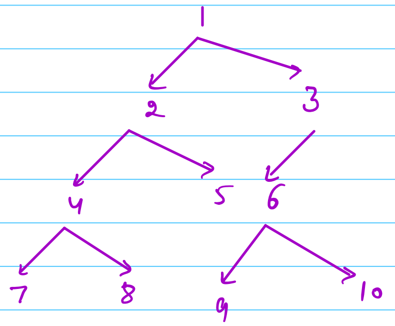
TC: O(N)
SC: O(H)

2> Inorder traversal

```
void    inorder ( root) {
        if (root = =null)return;

        inorder (root. left);  ⟶ Left
        print (root. data);    ⟶ Node
        inorder (root. right); ⟶ Right.

}
```



TC: O(N)
SC: O(H)

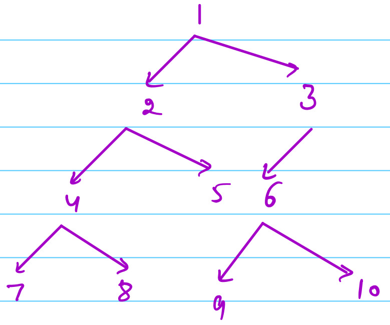
$$\underbrace{\overbrace{7\ 4\ 8}^{L}\ \overbrace{2}^{N}\ \overbrace{5}^{R}}_{L}\ \underbrace{1}_{N}\ \underbrace{\overbrace{9\ 6\ 10}^{L}\ \overbrace{3}^{N}}_{L}$$

3> Postorder traversal

```
void  postorder ( root) {
        if (root = =null)return;

        postorder(root. left);   ⟶ Left
        postorder(root. right);  ⟶ Right.
        print (root. data);      ⟶ Node

}
```



Meet at 8:15 am IST

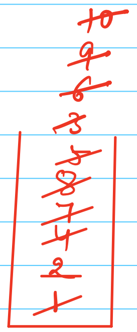
Q Iterative inorder traversal

```
void inorder ( root) {
        if (root = =null) return;
        inorder (root. left);  ⟶ Left
        print (root. data);    ⟶ Node
        inorder (root. right); ⟶ Right

}
```

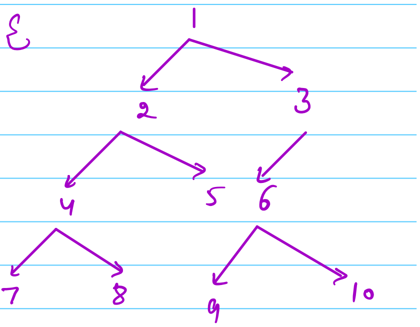
recursion ⟶ iteration.
stack   stack.

curr = ~~1~~ ~~2~~ ~~4~~ ~~7~~ ~~null~~ ~~null~~ ~~8~~ ~~null~~ ~~null~~ ~~5~~ ~~null~~ ~~null~~
~~3~~ ~~6~~ ~~4~~ ~~null~~ null ~~3~~ null



$op \rightarrow \underbrace{7\ 4\ 8\ 2\ 5}_{L}\ \underbrace{1}_{N}\ \underbrace{9\ 6\ 10\ 3}_{R}$

```
curr = root
while ( curr != null || !st.isEmpty()) {
    if (curr != null) {
            st.push (curr)
            curr = curr. left;   // Left
    }
    else {
            curr = st.pop();
            print (curr. data);  // Node
            curr = curr. right;  // Right
    }
}
```

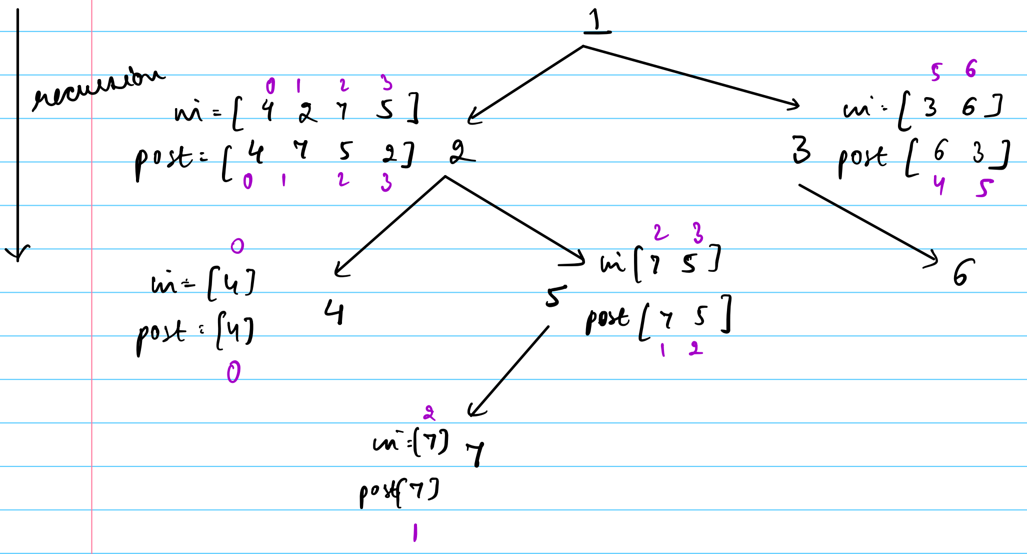


TC: O(N)
SC: O(H)

HW → Iterative preorder &
iterative postorder.

Q Construct binary tree from the given inorder & post order traversal (distinct nodes)

Eg
```
            0  1  2  3  4  5  6
in   =  [   4  2  7  5  1  3  6 ]      L N R
post =  [   4  7  5  2  6  3  1 ]      L R N
            0  1  2  3  4  5  6
```



```
Node build (in[], post[], in_L , in_R , post_R){

        if( inL > inR) return null;

    root = new Node (post[post_R]);

    // find index of root in inorder array
            1) Traverse the inorder array
            2) Hashmap (value → index) for in[] → TC: O(N)
                                                → SC: O(N)

     idx = map.get(root.data)
     cntR = inR - idx                      [idx+1, inR]

    root.left = build(in, post, in_L, idx-1 , postR-cntR-1 );
    root.right = build (in, post, idx+1, in_R , postR-1);

     return root;
  }
```

TC: O(N)
SC: O(N)

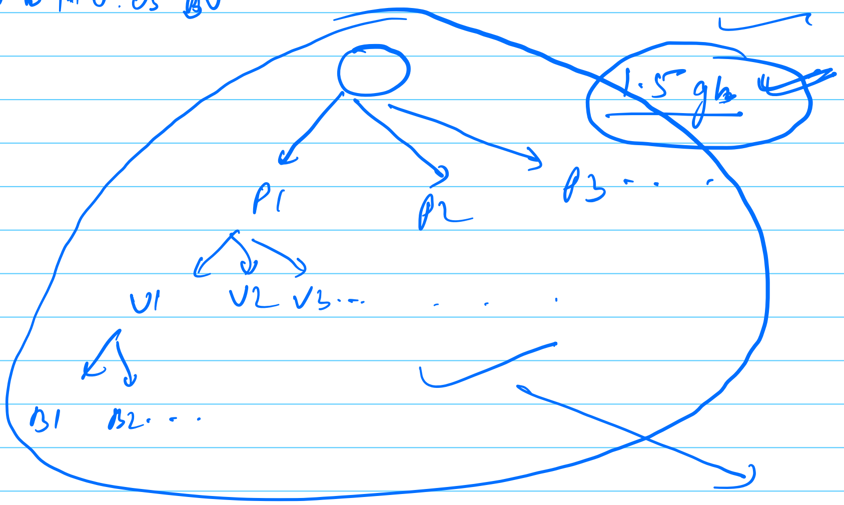
Doubts

Photoshop

P V B ×0.05 BV

~~flux~~

flash



Elasticsearch