AGENDA

- What is Tree?
- Terminology
- Traversal
- Inorder Traversal (Iterative)
- Construct Tree Ques

① Monday 25th Dec 2023
Monday 1st Jan 2024

② Mock Interview → DSA

Linear Data structures

Access data in sequential order

↓

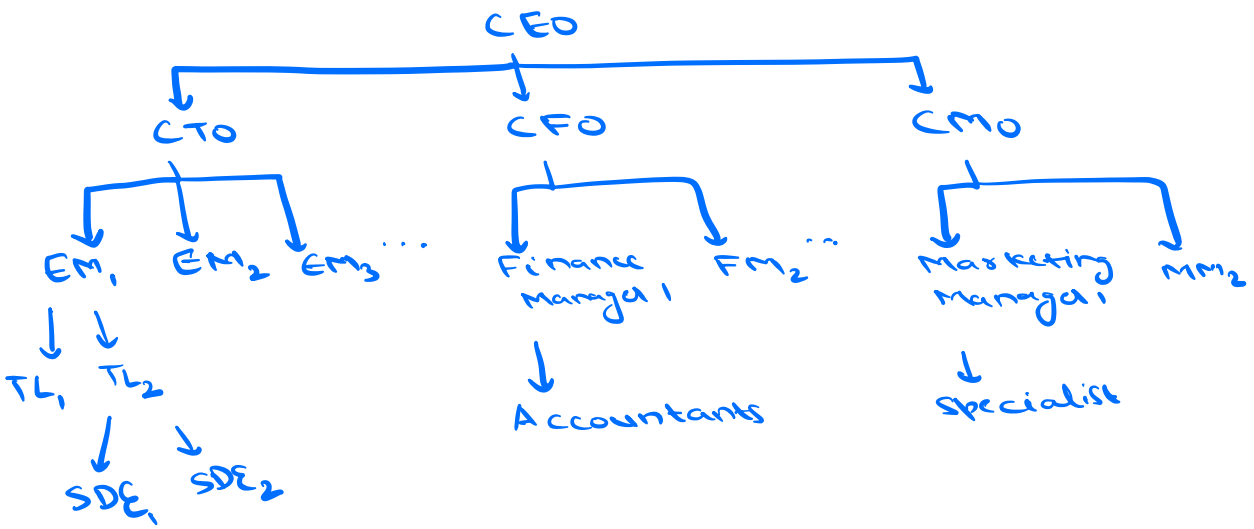
Array, LL, stack, Queues etc.

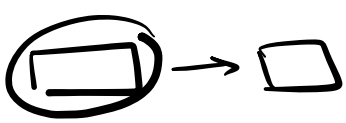
Tree : Fav topic among interviewers

Why ?
1. Requires solid recursion understanding
2. Ability to visualize concepts

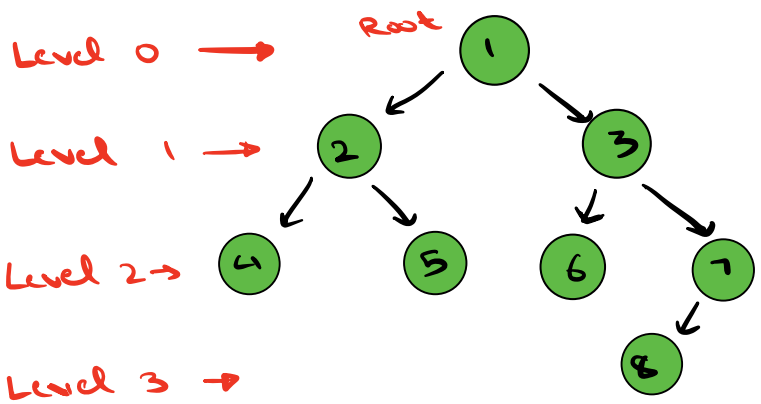
Tree is a non-linear data structure used to represent hierarchy in information.

For eg. Family Tree, Org Structure, Folder structure

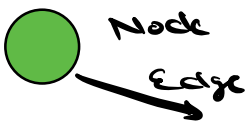




Tree Naming Convention



Node : element of tree with data and child nodes are connected to it.



Root: Topmost node of a tree from which all other nodes descend. It has no parent.
Root → Node 1

Parent: A node that has child node connected to it. Node 1, 2, 3, 7

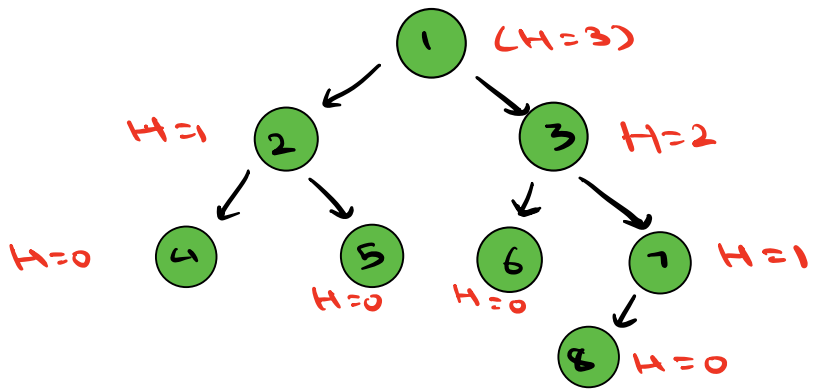
Child: A node that has a parent node connected to it. All Nodes except 1 (root)

Leaf: Node with 0 children. It's a terminal node. Node 4, 5, 6, 8

Depth / level: Distance from root node
Depth of root node → 0

Height: Length of longest path from a node to a leaf.

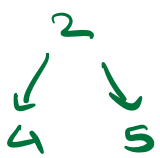
Height of tree = Height of root node



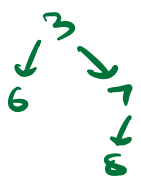
Child → 1 Parent
Par → Multiple Children

Subtree : A tree structure which is part of a larger tree

Subtree rooted at 2



Subtree rooted at 3



Siblings : Nodes that share same parent node.
P → 2 ; 4 and 5 are siblings

Ancestor : All nodes from parent to root upwards are ancestors of a node.
8 → 7, 3, 1

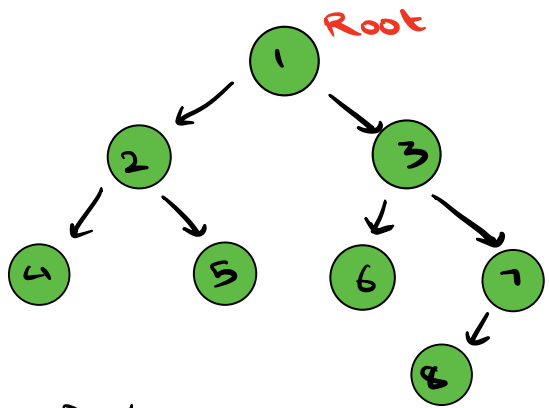
Descendant : All nodes from child to leaf node along all the paths.

Can a leaf node also be a subtree?
YES

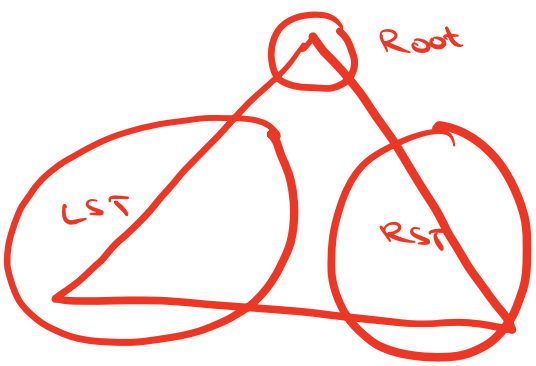
Do all nodes have a parent node?
NO (Everyone has parent except root)

Height of leaf node
Length of path from leaf to leaf → 0
H → 0

Binary Tree
A type of tree in which each node can have atmost 2 children i.e. either 0, 1, 2.
They are referred as left and right child




```
class TreeNode {
    int data
    TreeNode left
    TreeNode right

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



How can we traverse a tree?

L → Left subtree

R → Right subtree

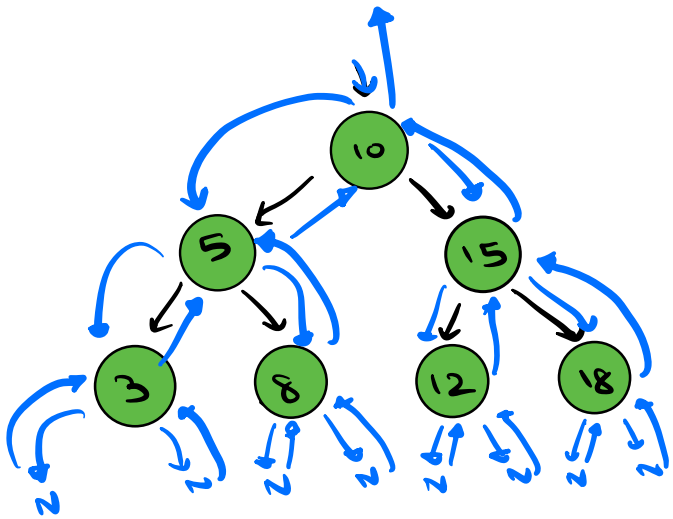
N → Node

Inorder L N R

→ Left subtree

→ Root

→ Right subtree



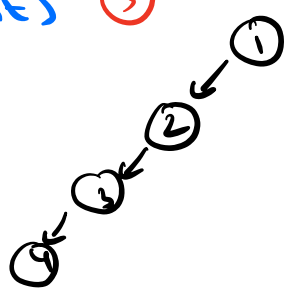
O/P → 3 5 8 10 12 15 18

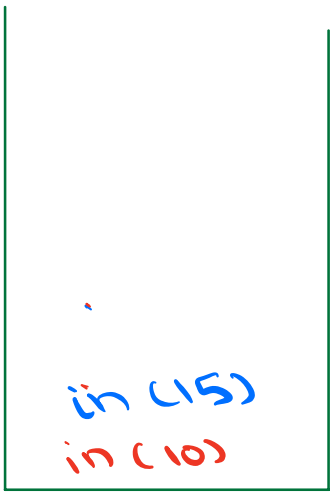
```
void inorder (Node root) {
    if (root == NULL)
        return
    inorder (root.left)     ①
    print (root.data)       ②
    inorder (root.right)    ③
}
```

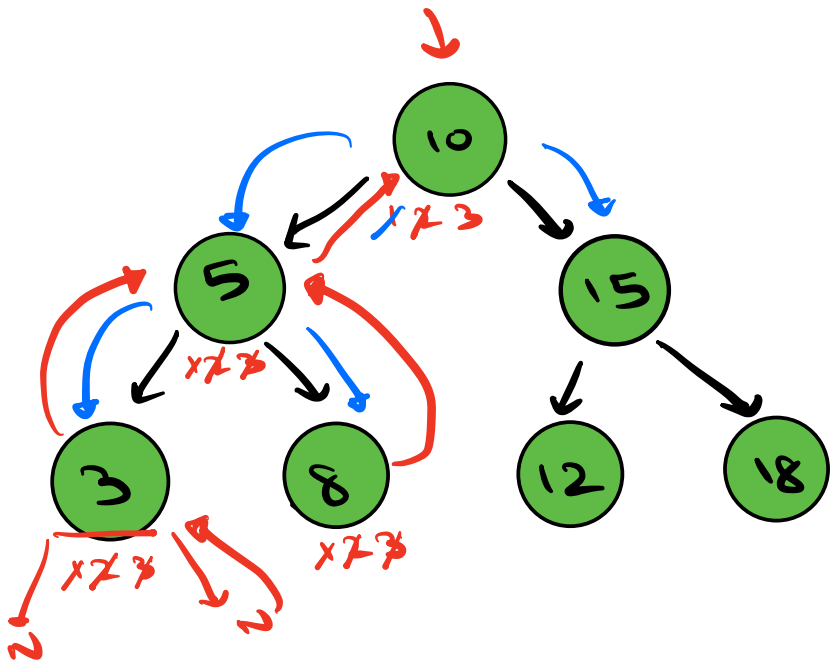
N → nodes in a tree

TC: O(N)

SC: O(N)







3 fn calls

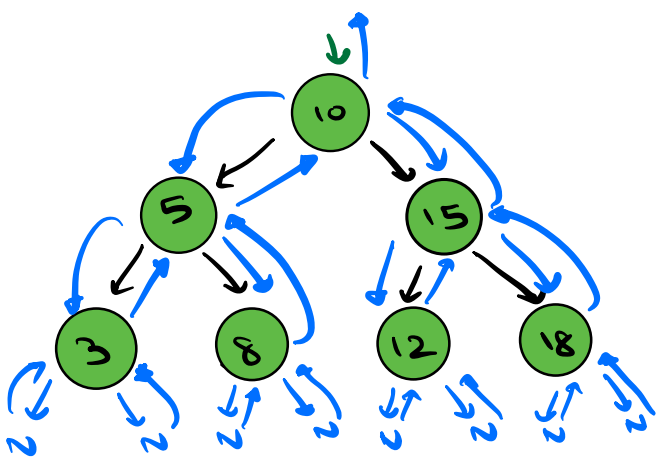
3 fn calls

3 5 8 10

Preorder

N L R

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

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



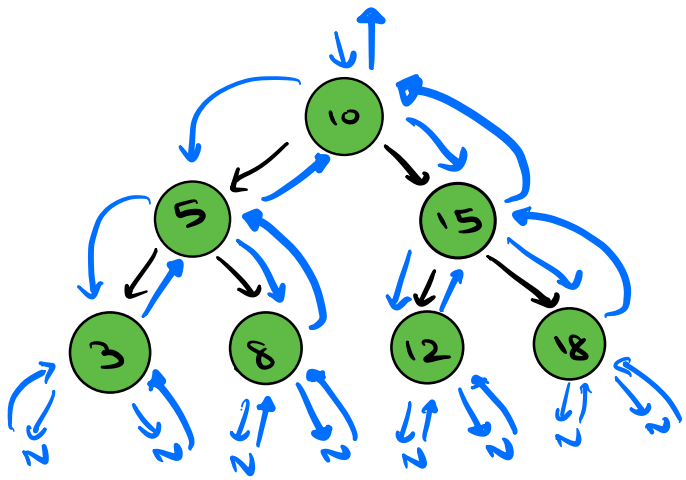
yp: 10 5 3 8 15 12 18

TC: O(N)

SC: O(H)

Postorder

L R N



o/p: 3 8 5 12 18 15 10

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

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

TC : O(N)

SC : O(H)

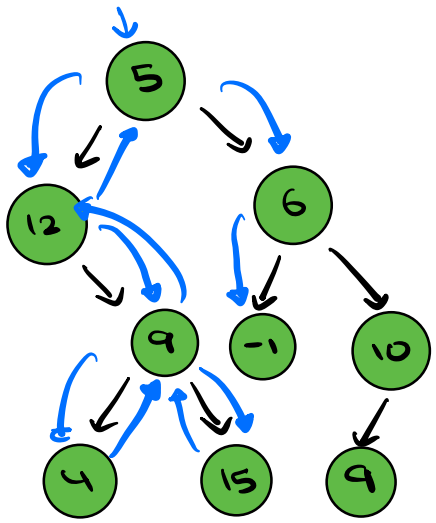
L	N	R	Inorder
L	R	N	Postorder
N	L	R	Preorder

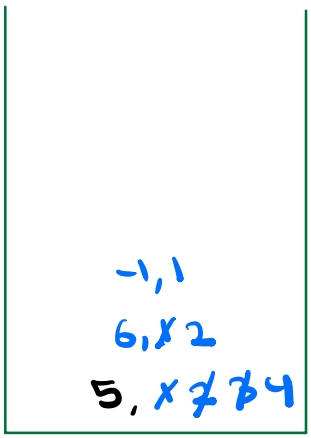
10:37

Iterative Inorder Traversal

L N R

1) call left child
2) print data
3) call right child





Node, Task

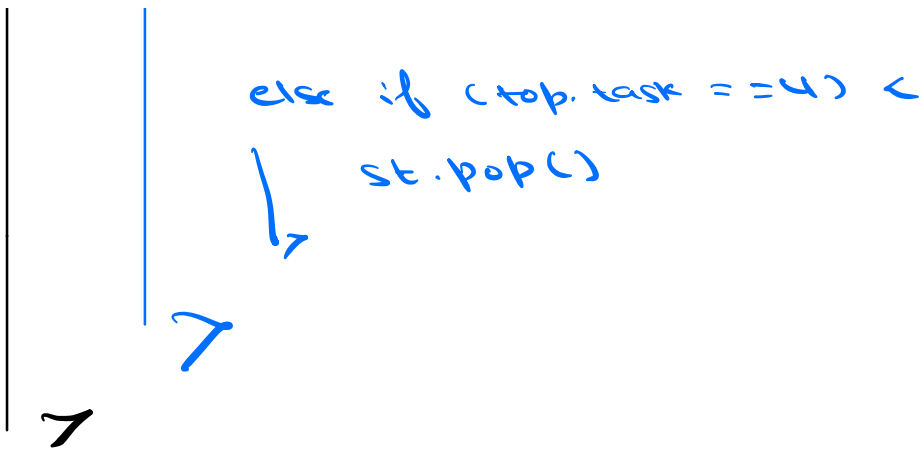
O/P : 12 4 9 15 5

Task	Action
4	Pop from stack
2	Print

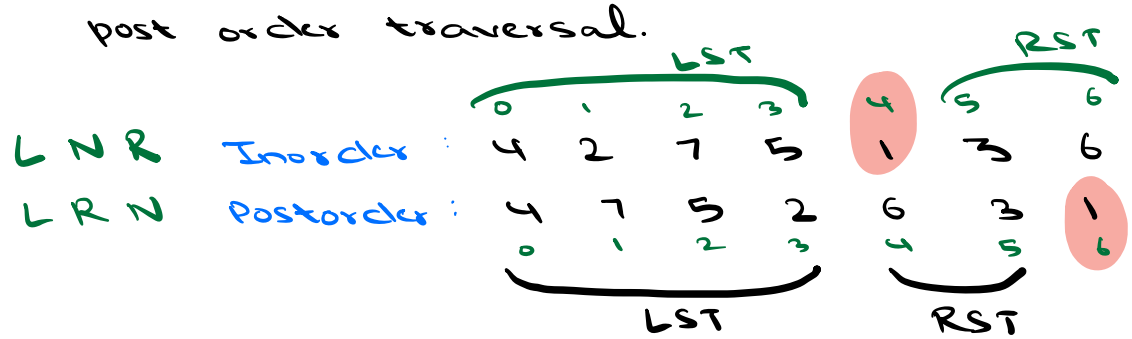
```
class Pair <
    Node cur
    int task
    Pair (Node t) <
        cur = t
        task = 1
    >
>
```

```
void inorder (Node root) {

    stack <Pair> st
    Pair p = new Pair (root)
    st.push (p)

    while (st.size() > 0) {
        Pair top = st.top()
        if (top.task == 1) {
            st.top().task++
            if (top.cur.left != NULL) {
                Pair left = new Pair (top.cur.left)
                st.push (left)
            }
        }
        else if (top.task == 2) {
            st.top().task++
            print (top.cur.data)
        }
        else if (top.task == 3) {
            st.top().task++
            if (top.cur.right != NULL) {
                Pair right = new Pair (top.cur.right)
                st.push (right)
            }
        }
    }
```

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


else if (top.task == 4) {
st.pop()
}
}
}

Q. Construct binary tree from inorder and post order traversal.



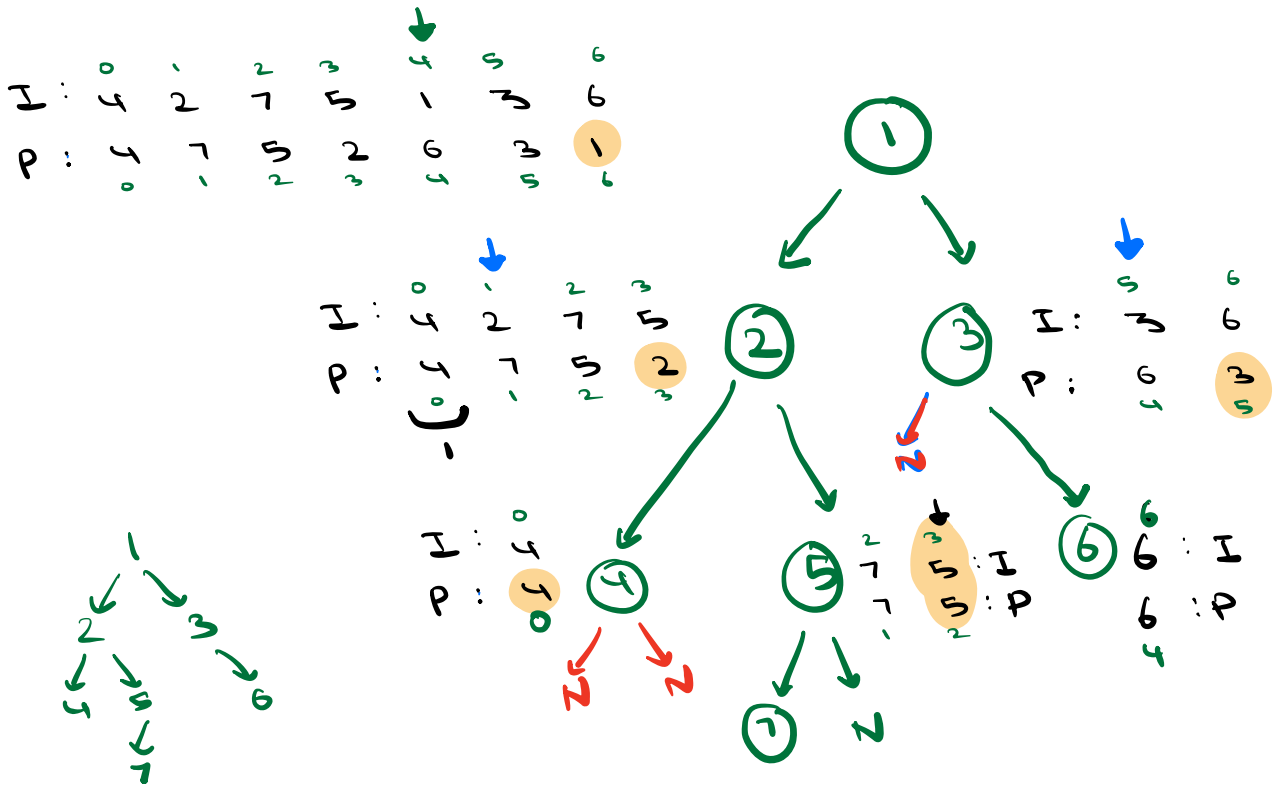
Observation

① 1 is root node (end node of postorder)

② LST of 1
4, 2, 7, 5

③ RST of 1
3, 6

is, ie
ps, pe



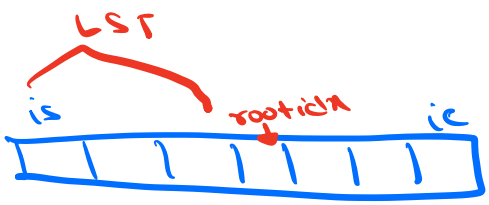
TC : N^2 | TC : O(N)
SC : H | SC : O(H + N)

O N-1 O N-1

```
Node buildTree(in[], post[], is, ie, ps, pe){

    if (is > ie)
        return NULL

    rootVal = post[pe]
    TreeNode root = new TreeNode(rootVal)
    rootIdx = find(rootVal, in[])
    cnt = rootIdx - is      // Nodes in LST

    root.left = buildTree(in, post, is, rootIdx-1,
                          ps, ps+cnt-1)

    root.right = buildTree(in, post, rootIdx+1, ie,
                           ps+cnt, pe-1)

    return root
}
```

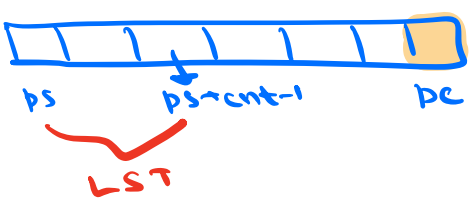
LST → (is → rootIdx-1)

I →

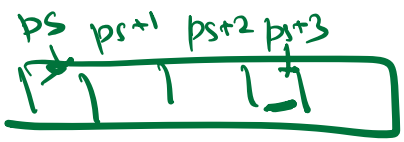


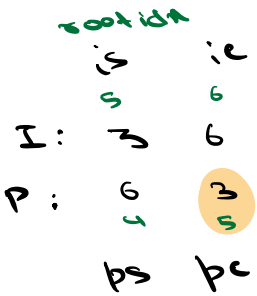
No. of nodes in LST = rootIdx - is

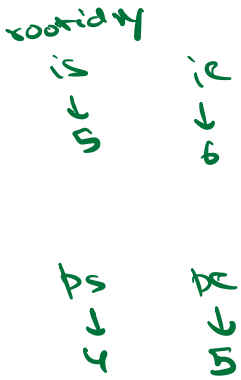
P →



= rootIdx - is

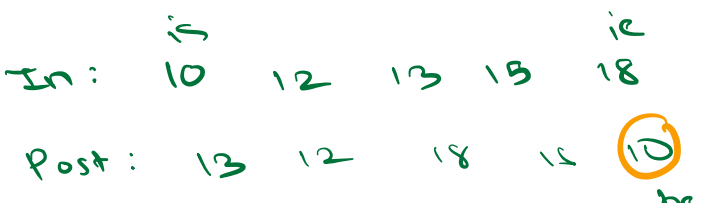


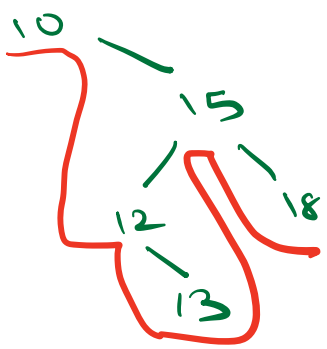


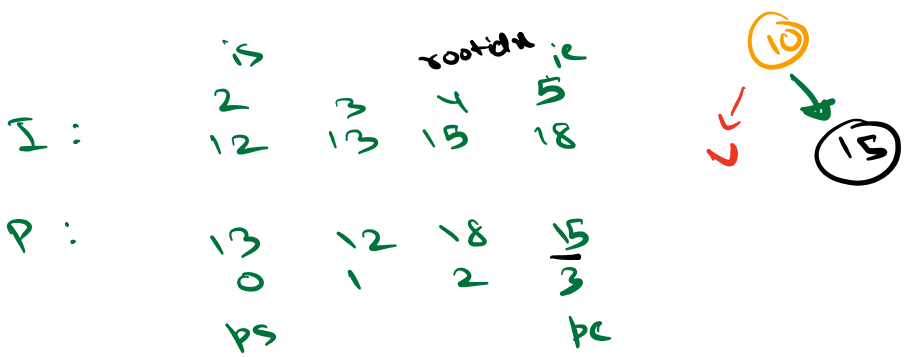


LST

I: is rootidx-1







is, rootidx-1
2, 3

ps, rootidx-1
0 3