9:05 pm

Tree

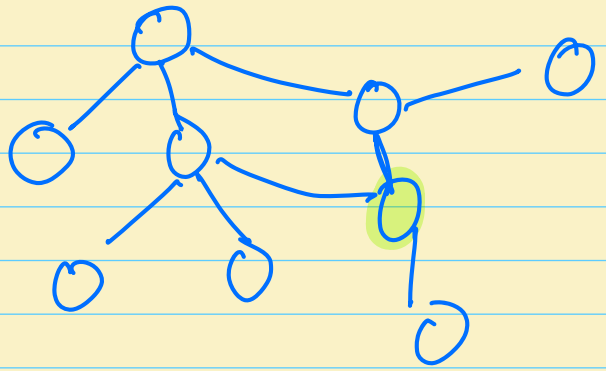
Array, Linked list, stack, queues :

↳ Linear data structure.

→ Non-Liner data structur.

↳ Trees

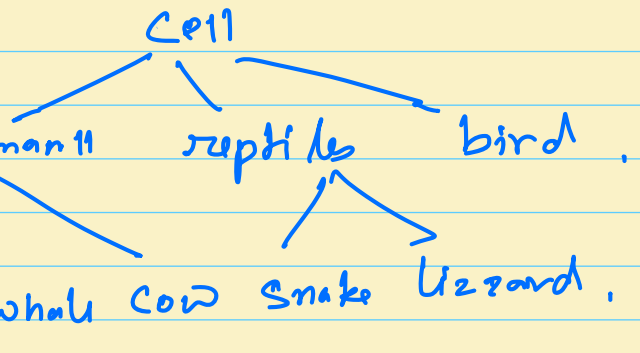
↳ graphs.



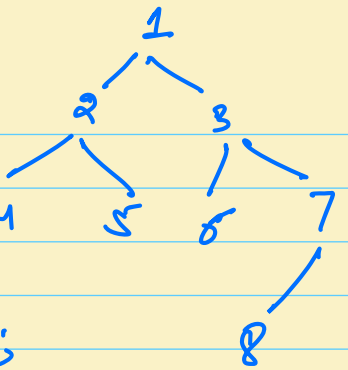
Tree → heirachal way of representing data.

Tree

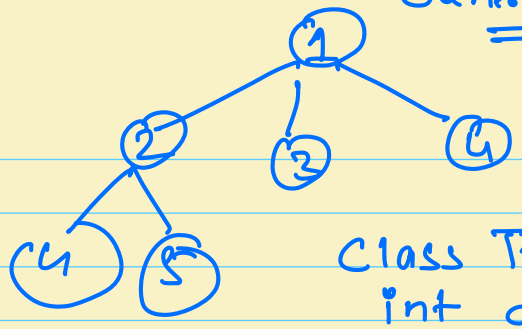
family trees.



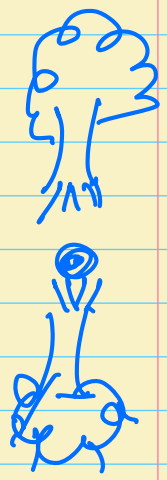
Binary



Generic tree



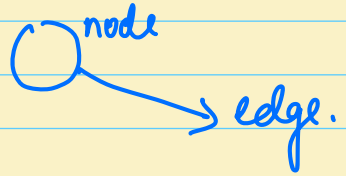
Naming



```
class TreeNode
{
    int data;
    Tree left;
    Tree right;
}
```

```
class TreeNode{
    int data;
    List<Tree>
}        ref;
```

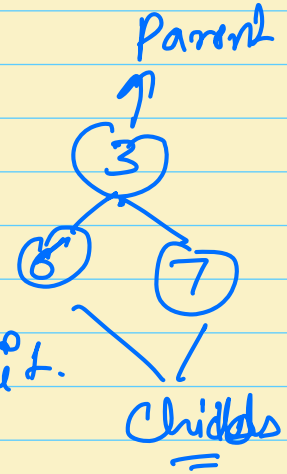
1) Node : An element in the tree contains data and may have child nodes connected.



2) Root : Top most node of the tree.

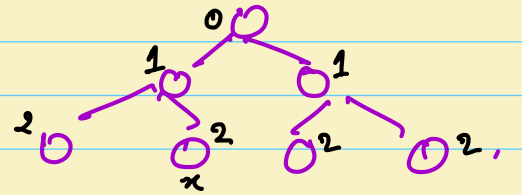
3) Parent : A node that have child connected to it.

4) Child : A node that has parent connected to it.



5) leaf : A node that has no child.

6) Depth : The level at which node resides in the tree.



depth of x = 2

7) Height : The length of the longest path from node to leaf.
Height of the tree is length from root to leaf [longest]

8) Subtree : A tree that is part of larger tree.

9) Sibling : Nodes that share the same parent node.

(1) → 2 and 3 are their sibling

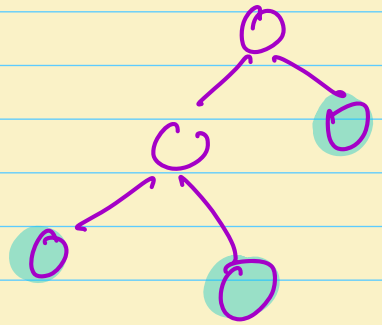
10) Ancestor : All the node from parent to the root node upwards.

(8) → 1 , 3, 7

11) Descendant:

All nodes from child to the leaf node along that path.

Quiz 3

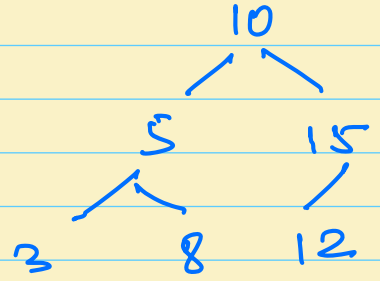


ans = 0

Binary Tree

↳ It can have atmost 2 childs

0, 1, 2 childs.

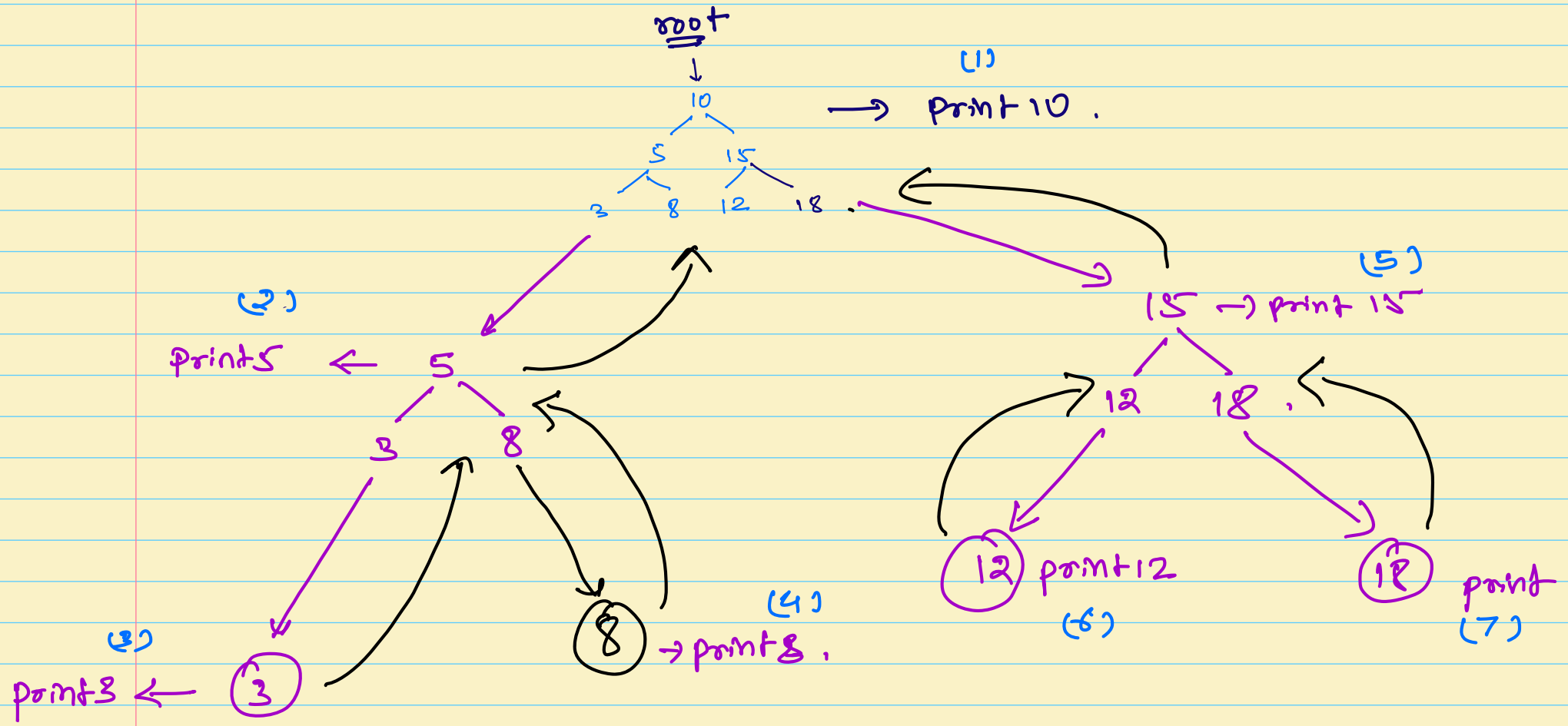


Traversal of the Tree

(1) Preorder Traversal

node left right
(N L R)

1) visit the current node. (print)
2) left of the sub tree.
3) Right of the sub tree.



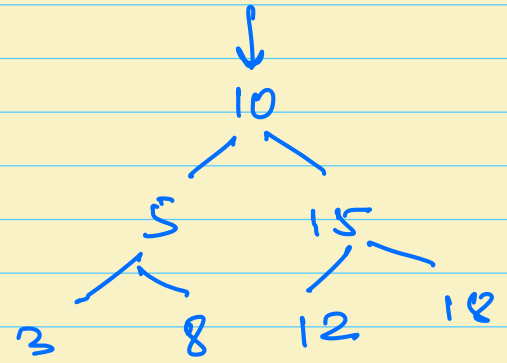
10, 5, 3, 8, 15, 12, 18

Pseudo code

```
void preorder (root)
{
    if (root == null) return
    print (root.data)
    preorder (root -> left)   // left
    preorder (root -> right)  // right
}
```

(2) Inorder traversal [L N R]

1) Traverse left subtree
2) print the current node.
3) Traverse the right subtree.

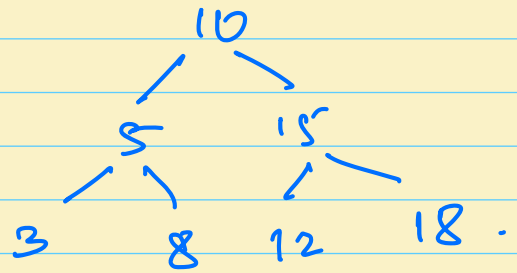


BST ⇒ inorder traversal ↓ sorted list.

3, 5, 8, 10, 12, 15, 18

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

    inorder(root -> left)   // left
    print (root.data)
    inorder.(root -> right) // right
}
```

(B) Post order traversal (L R N)



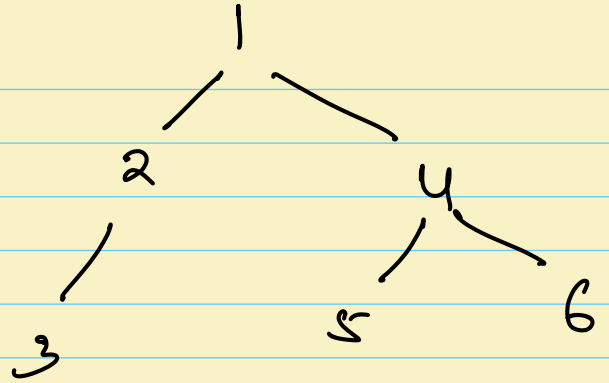
(1) Traver left subtree
(2) Traverse right subtree
(3) print node.

3, 8, 5, 12, 18, 15, 10

pseudo code,

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

    postorder (root -> left)  // left
    postorder (root -> right) // right
    print (root.data)
}
```

Quiz



inorder. LNR

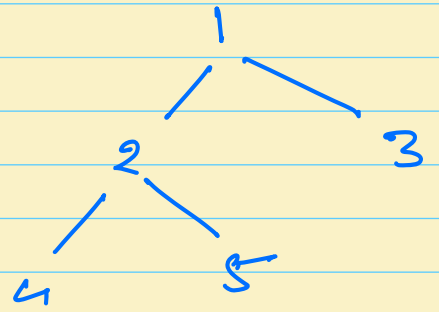
left
print
Right.

3, 2, 1, 5, 4, 6

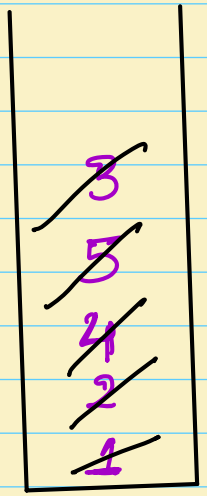
O(n)

number of nodes in the tree

Iterative Inorder traversal



L N R

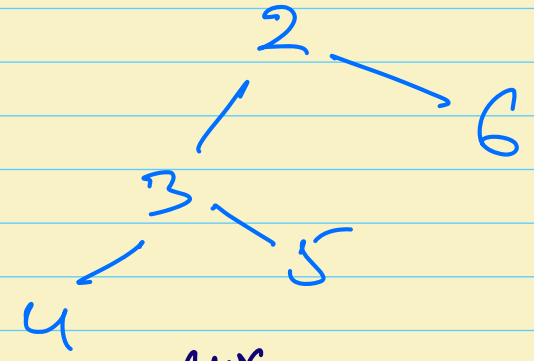


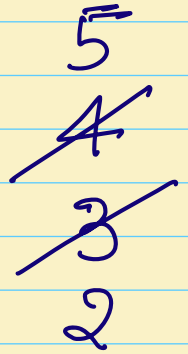
4, 2, 5, 1, 3

pseudo code

```
cur = root.
st = ()          // stack.
=

while (cur != null || !stack.isEmpty())
{
    if (curr != null)
    {
        st.push(cur)
        cur = cur.left;
    }
    else
    {
        cur = st.pop();
        print(cur.data);
        cur = curr.right;
    }
}
```

```
if ( st.empty() )
    break;
}
```





4, 3

Q48

Flipkart →

↓

order is placed

↓

provide efficient mechanism

↓

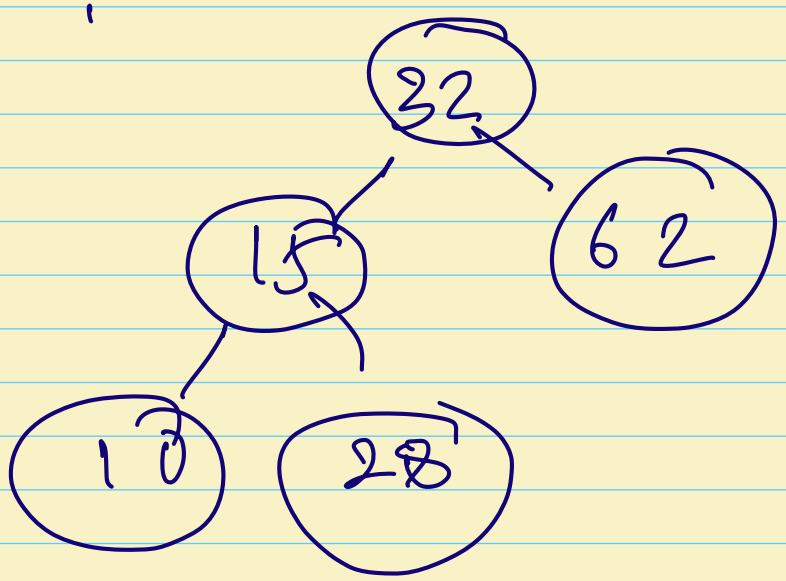
return the order based on time of placing.

$O_1 \rightarrow t_1$

$O_2 \rightarrow t_2$

$O_3 \rightarrow t_3$

$t_1 < t_2 < t_3$



Inorder traversal

↓

number in sorted format

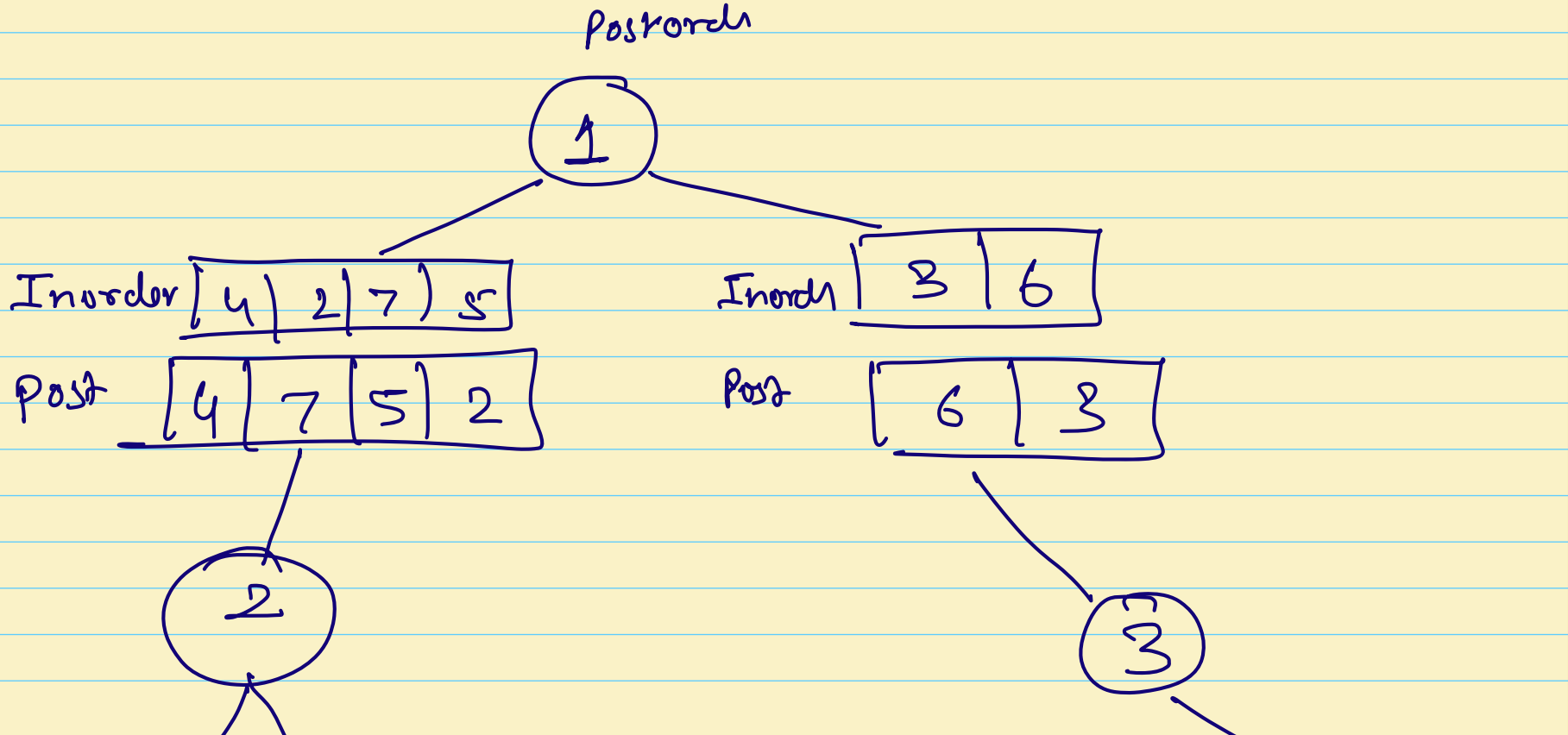
(L N R)

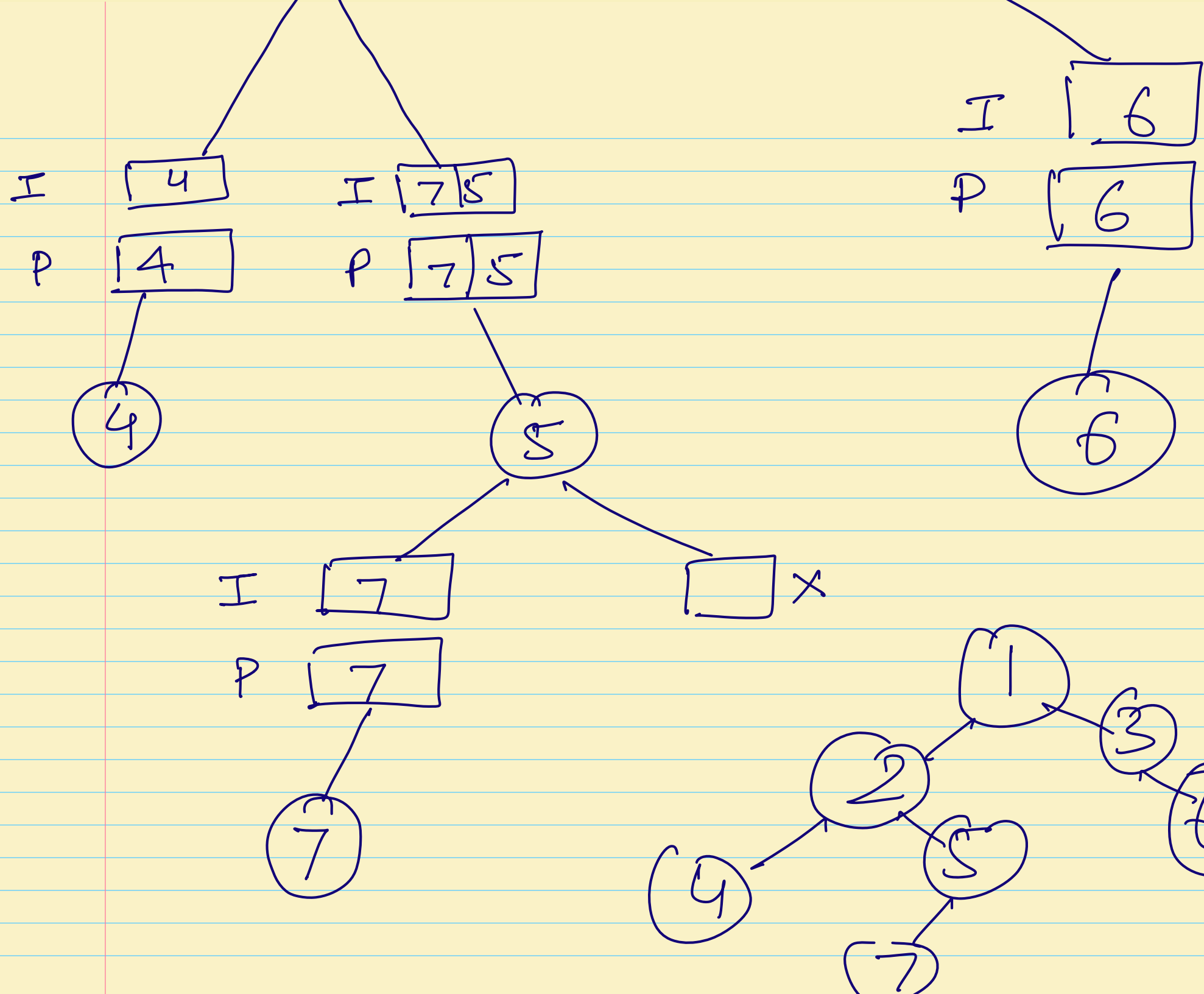
Qu Inorder: [4, 2, 7, 5, 1, 3, 6]

Postorder: [4, 7, 5, 2, 6, 3, 1].

(L R N)

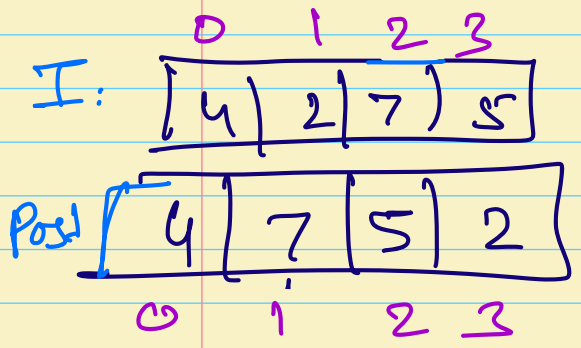
Create a tree using the above traversal.




I
4
P
4
4
I
7
5
P
7
5
5
I
7
P
7
7
X
I
6
P
6
6
1
2
3
4
5
6
7

Last element of postorder → root

```
funtion buildTree (inorder[], postorder[])
{
    if postorder is empty:
        return null;

    rootValue = postorder.last

    root = new TreeNode (rootValue);

    rootIndex = indexOf (inorder, rootValue)

    leftPartI = subarray (inorder, start, rootInd-1)

    rightPartI = subarray (inoder, rootIndx+1, end)

    leftPartP = subarray (postorder, start, rootIndx-1)
```



rootIndex = 1

Inorder

0, 0

leftpartI = 4

rightpartI = 7 5

Post Order.

0, 0

left Part P = [4]

right Part P = [7 | 5]

rightPart P = Subarray(postorder, rootIndex, end-1);

root.left = buildTree (leftPartI, leftPart P)

root.right = buildtree (rightPartI, rightPartP)

return root;

}

T.C: O(N)

S.C: O(N).