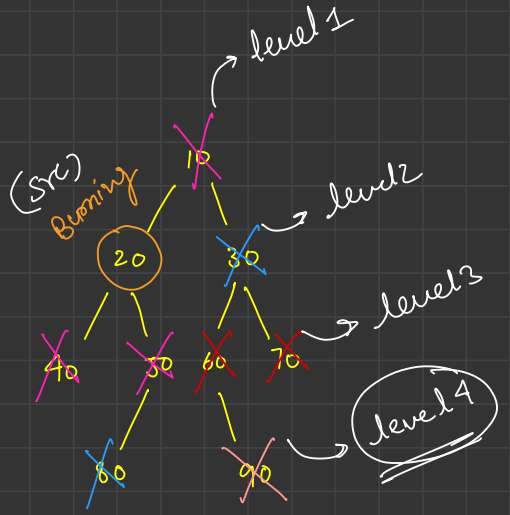
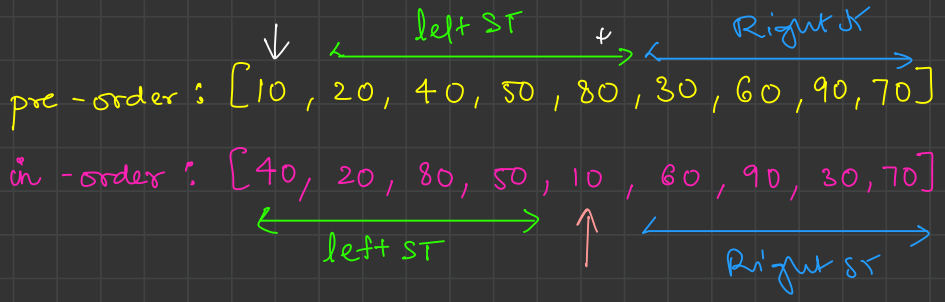
Burning Trees

time = 0/1/2/3/4



Build a Binary Tree from pre-order & in-order Traversal

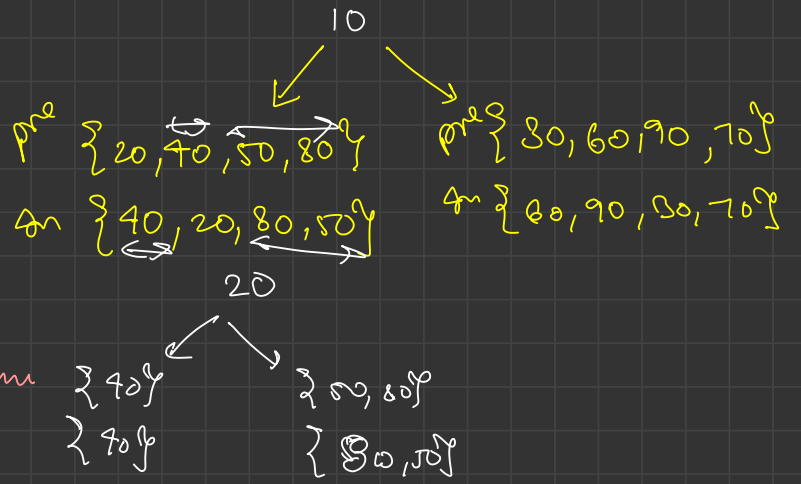


pre
root + left + right

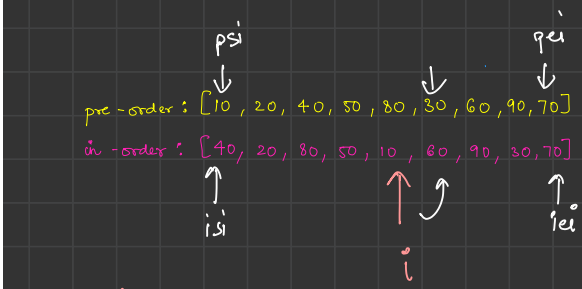
in
left + root + right

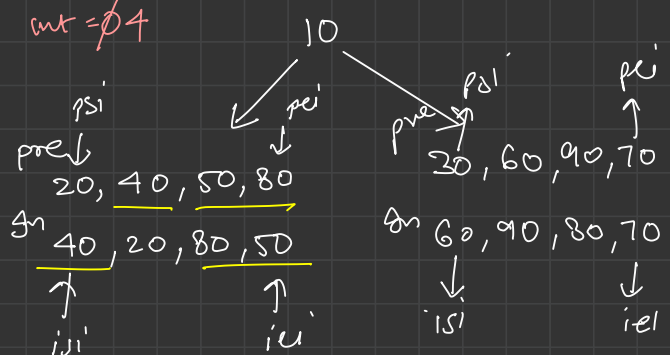
TreeNode Construct (pre[], in[])

↳ it take pre, in and returns tree from them ↓ root of the



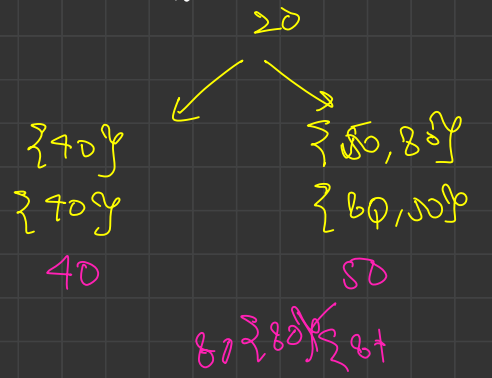
```java
public TreeNode construct(int[] pre, int psi, int pei, int[] in, int isi, int iei) {
    if (psi > pei) {
        return null;
    }

    if (isi > iei) {
        return null;
    }

    TreeNode root = new TreeNode(pre[psi]);

    int i = isi;
    int cntNumberOfNodesInLeftSubtree = 0;
    while (in[i] != root.val) {
        i++;
        cntNumberOfNodesInLeftSubtree++;
    }

    root.left = construct(pre, psi + 1, psi + cntNumberOfNodesInLeftSubtree, in, isi, i - 1);
    root.right = construct(pre, psi + cntNumberOfNodesInLeftSubtree + 1, pei, in, i + 1, iei);

    return root;
}
```





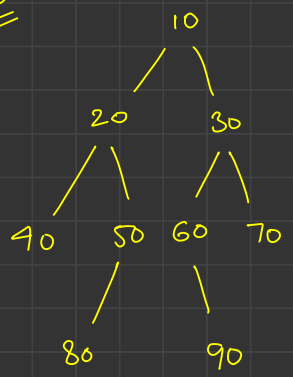
TC: O(N^2)

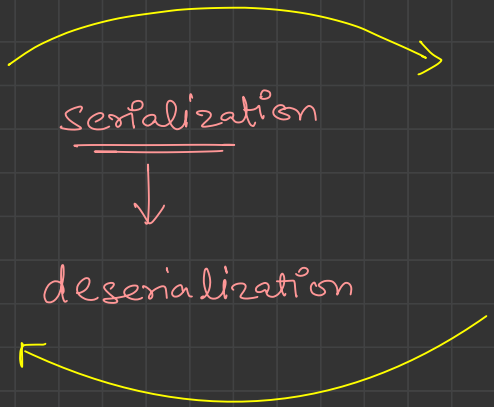
SC: O(H)



Serialize and Deserialize Binary Tree

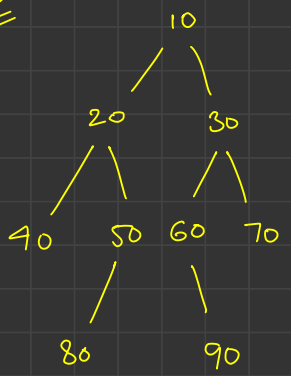
Tree





String Message = " "

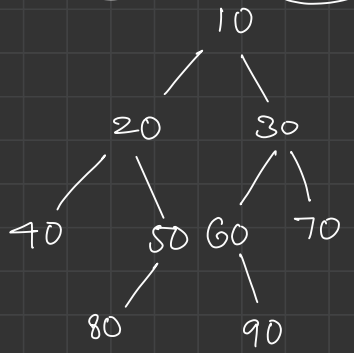
Tree



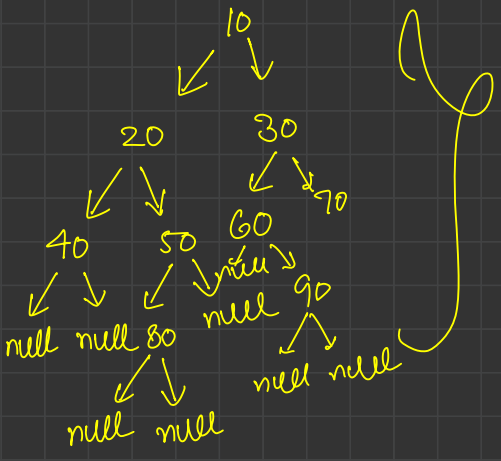
Serialize

pre-order

"10, 20, 40, null, null, 50, 80, null, null, null, 30, 60, null, 90, null, null, 70, null, null"



[10, 20, 40, null, null, 50, 80, null, null, null, 30, 60, null, 90, null, null, 70, null, null]



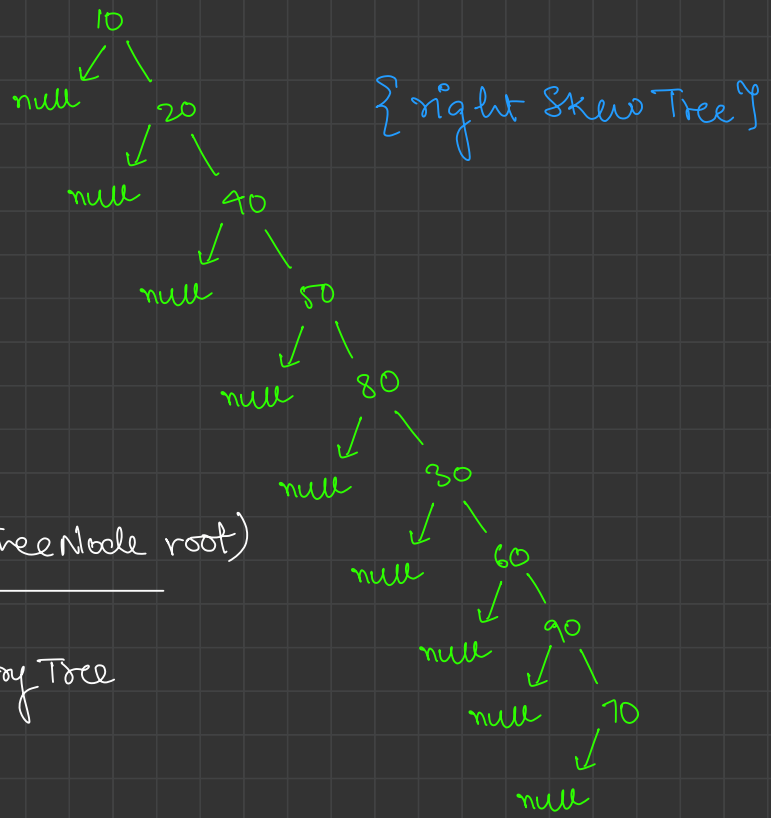

```java
static int idx = 0;
public static TreeNode buildTree(String[] arr) {
    if (idx == arr.length) {
        return null;
    }

    if (arr[idx].equals("null") == true) {
        idx++;
        return null;
    }

    TreeNode root = new TreeNode(Integer.parseInt(arr[idx]));
    idx++;

    root.left = buildTree(arr);
    root.right = buildTree(arr);

    return root;
}
```

Flatten a Binary Tree { In-place }

↓

Modify the ip, can't create new.

(Tree)

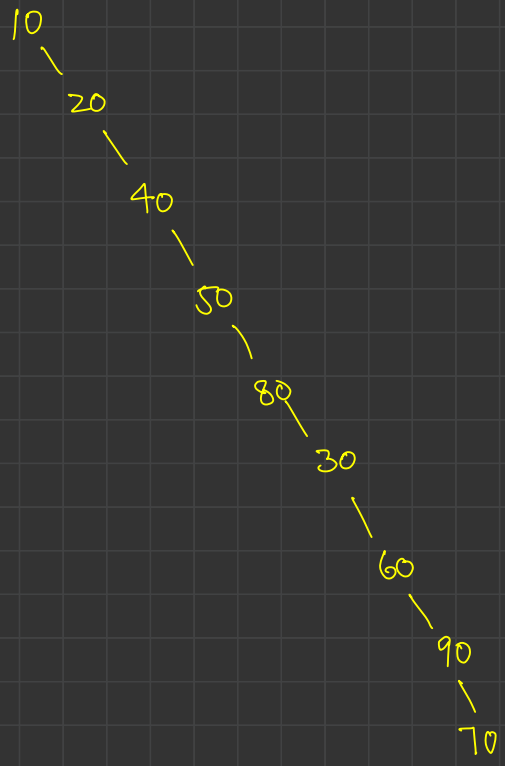


{ right Skew Tree }

```
TreeNode flatten(TreeNode root)
```

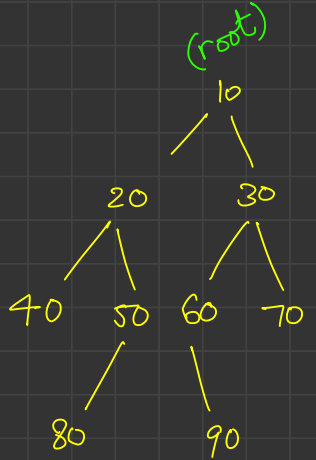
↓

flatten a binary Tree

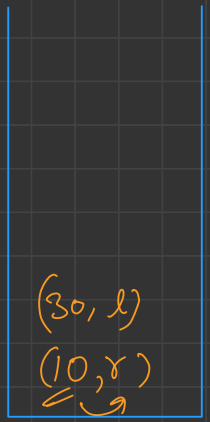

10
20
40
50
80
30
60
90
70

✓ inorder traversal (iterative Method)

↳ without recursion



Ans: 40, 20, 80, 50, 10, 60, 90, 30, 70



} 40, 20, 80, 50, 10