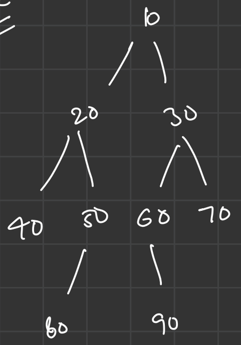
Binary Trees

Serialize and Deserialize BST

Serialize

Binary Tree

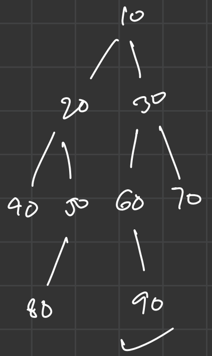


serialization → String = " "

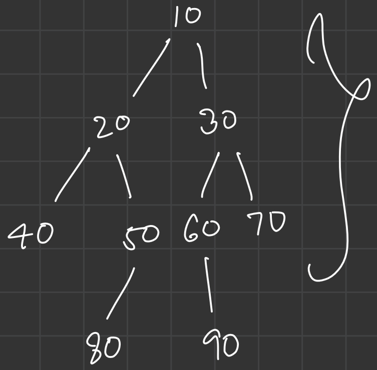
message

deserialization → BT

$$\text{BT} \xrightarrow{\text{serialization}} \text{message} \xrightarrow{\text{deserialization}} \text{BT}$$



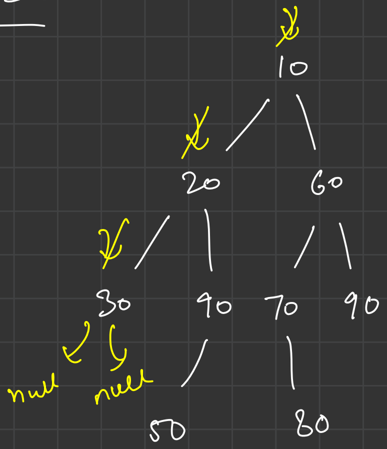
→ String →



root, left, right

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

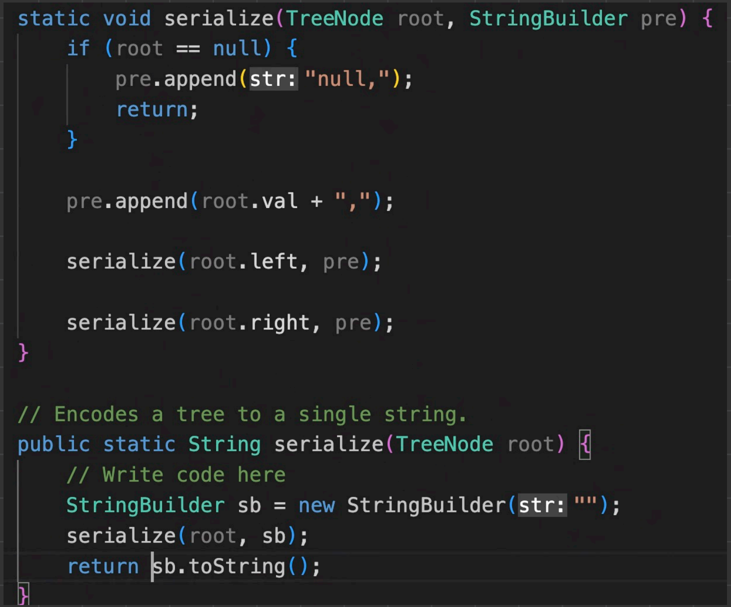
Serialize



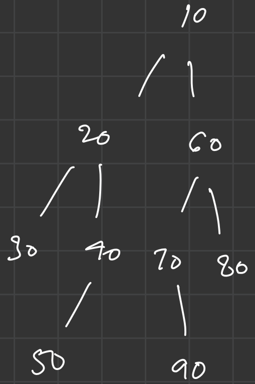
String = "10, 20, 30, null, null

- → add to String
- → left move
- → right move

```
fun ( TreeNode root, StringBuilder Sb)
{
  if (root == null)
  {
    sb.append("null, ");
    return;
  }

  sb.append(root.val, " ");

  fun(root.left, sb);
  fun(root.right, sb);
}
```



```
static void serialize(TreeNode root, StringBuilder pre) {
    if (root == null) {
        pre.append(str: "null,");
        return;
    }

    pre.append(root.val + ",");

    serialize(root.left, pre);

    serialize(root.right, pre);
}

// Encodes a tree to a single string.
public static String serialize(TreeNode root) {
    // Write code here
    StringBuilder sb = new StringBuilder(str: "");
    serialize(root, sb);
    return sb.toString();
}
```



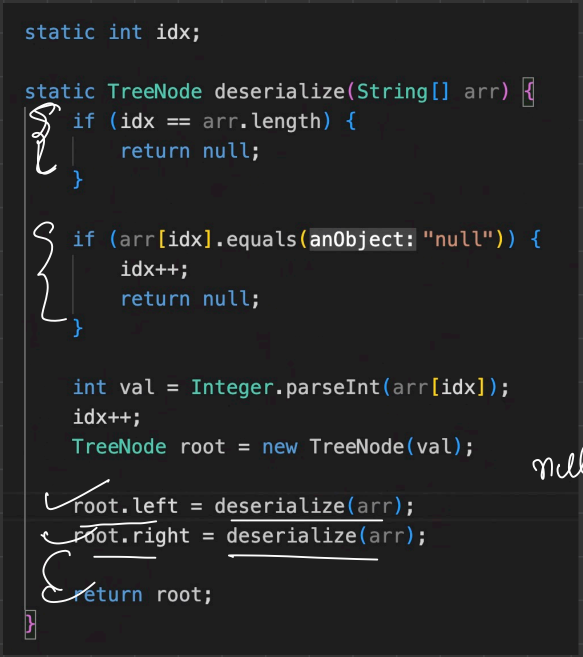
sb = " 10, 20, 30, null, null, 40, 50, null, null,
null, 60, 70, null, 90, null, null, 80,
null, null,"

deserialize

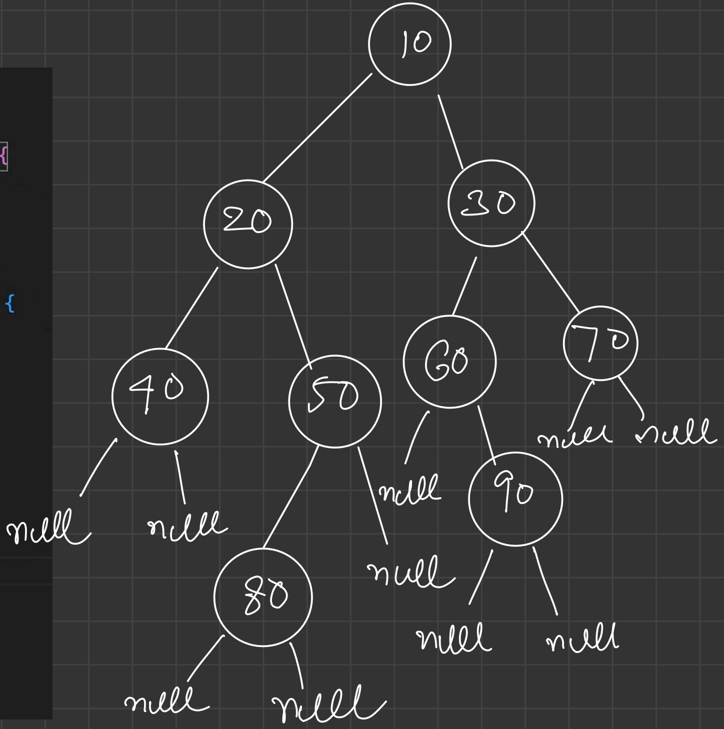
“50” → parseInt

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

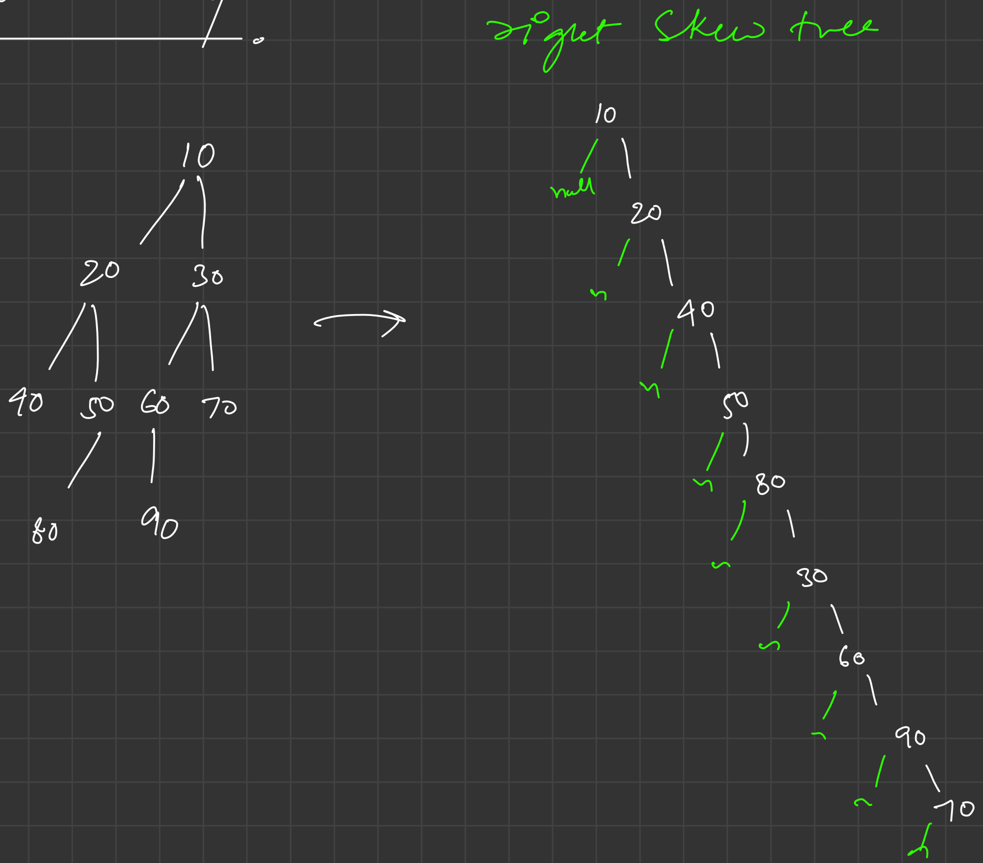
0

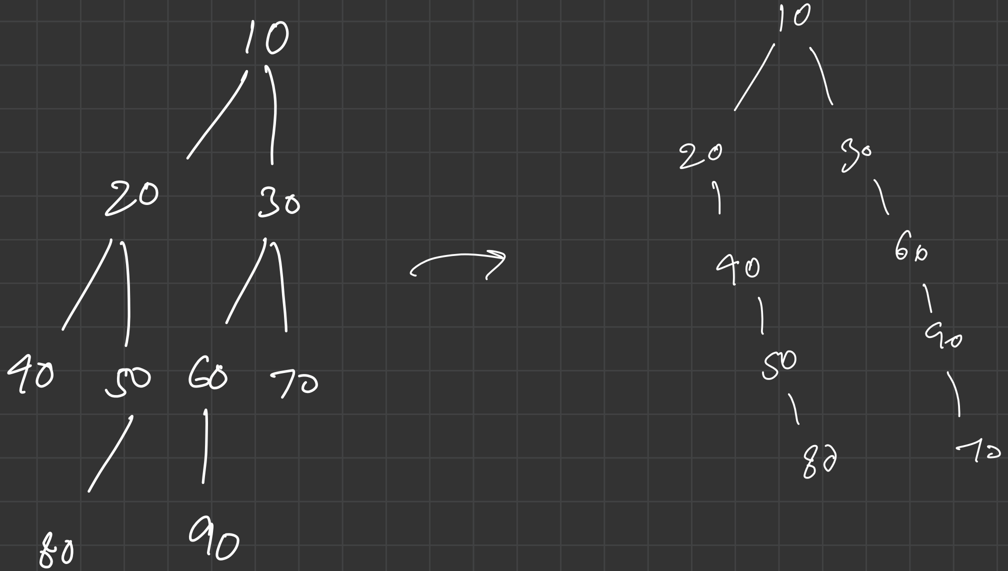


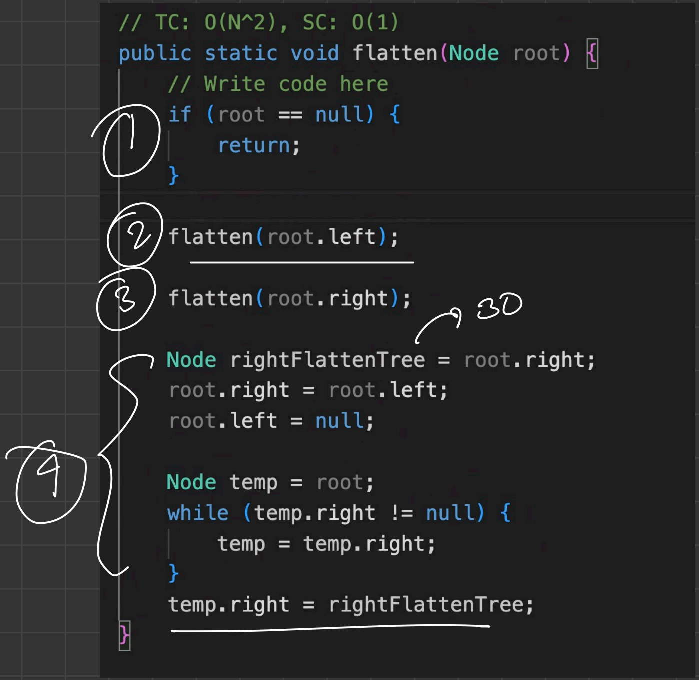
```
static int idx;

static TreeNode deserialize(String[] arr) {
    if (idx == arr.length) {
        return null;
    }

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

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

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

    return root;
}
```



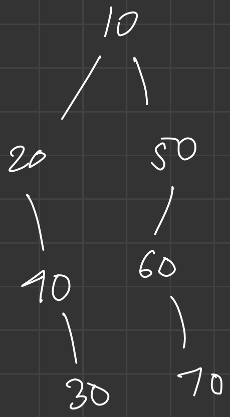
Flatten a Binary Tree.

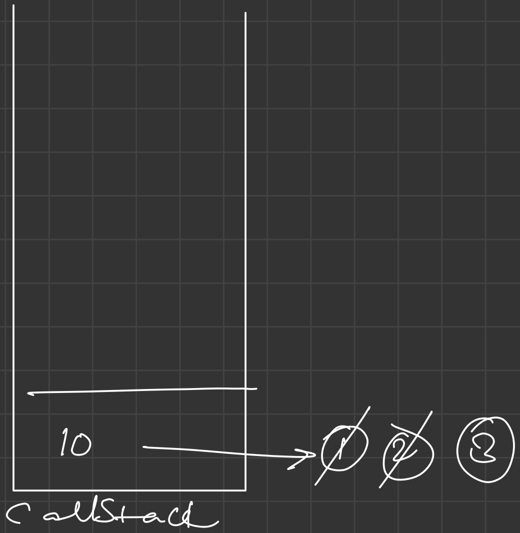



10
20
30
40
50
60
70
80
90
10
20
30
40
60
50
90
80
70

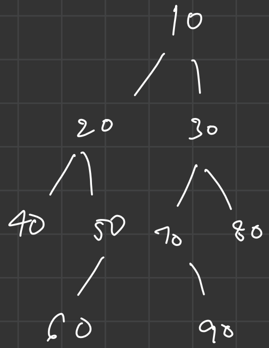


```java
// TC: O(N^2), SC: O(1)
public static void flatten(Node root) {
    // Write code here
    if (root == null) {
        return;
    }

    flatten(root.left);
    flatten(root.right);

    Node rightFlattenTree = root.right;
    root.right = root.left;
    root.left = null;

    Node temp = root;
    while (temp.right != null) {
        temp = temp.right;
    }
    temp.right = rightFlattenTree;
}
```

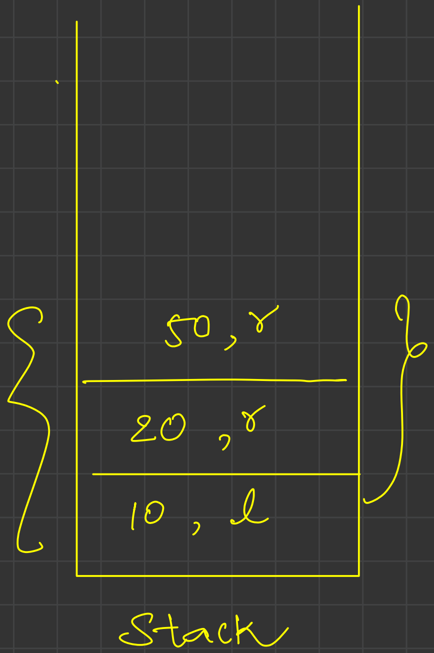




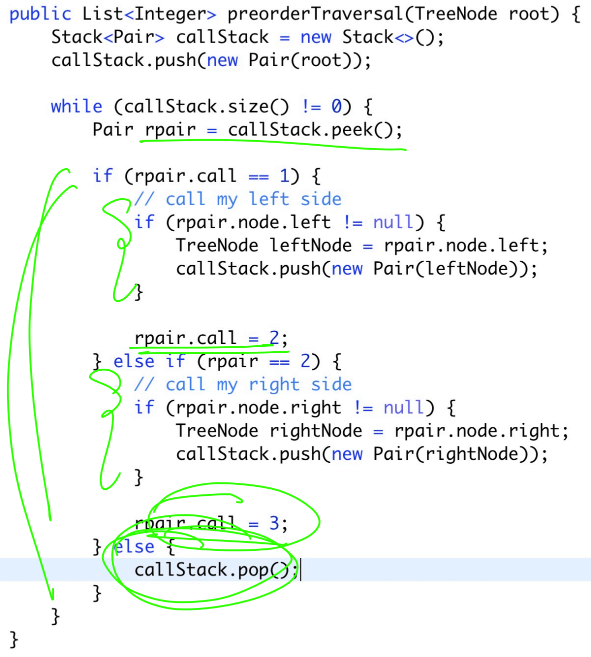
Preorder traversal (iteratively)



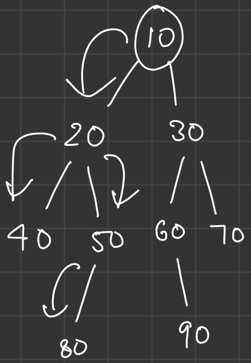
10, 20, 40, 50, 60

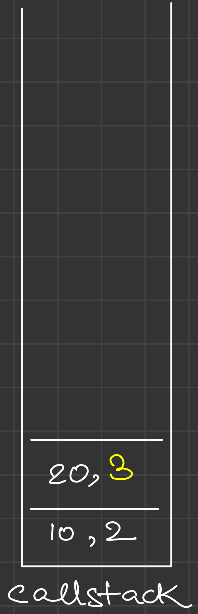


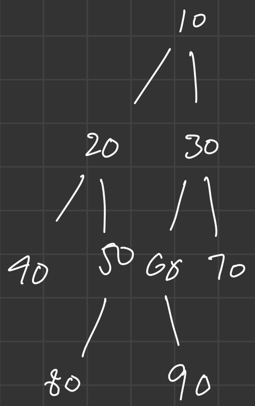
print()
call left
call right



```
public List<Integer> preorderTraversal(TreeNode root) {
    Stack<Pair> callStack = new Stack<>();
    callStack.push(new Pair(root));

    while (callStack.size() != 0) {
        Pair rpair = callStack.peek();

        if (rpair.call == 1) {
            // call my left side
            if (rpair.node.left != null) {
                TreeNode leftNode = rpair.node.left;
                callStack.push(new Pair(leftNode));
            }

            rpair.call = 2;
        } else if (rpair == 2) {
            // call my right side
            if (rpair.node.right != null) {
                TreeNode rightNode = rpair.node.right;
                callStack.push(new Pair(rightNode));
            }

            rpair.call = 3;
        } else {
            callStack.pop();
        }
    }
}
```





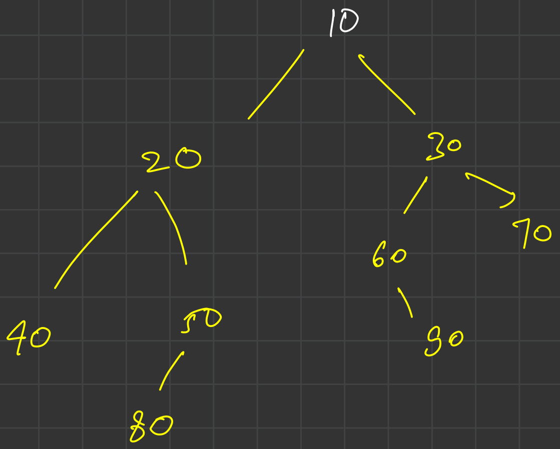


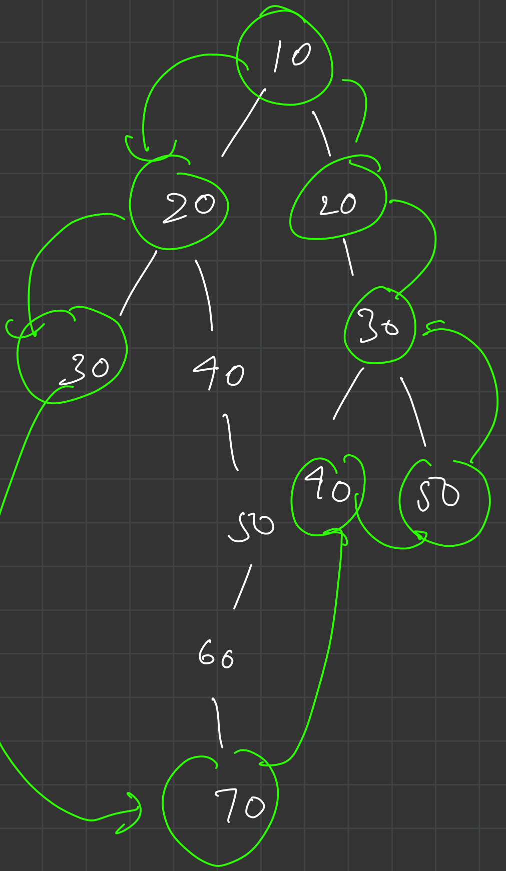
0 1 2 3 4 5 6 7 8

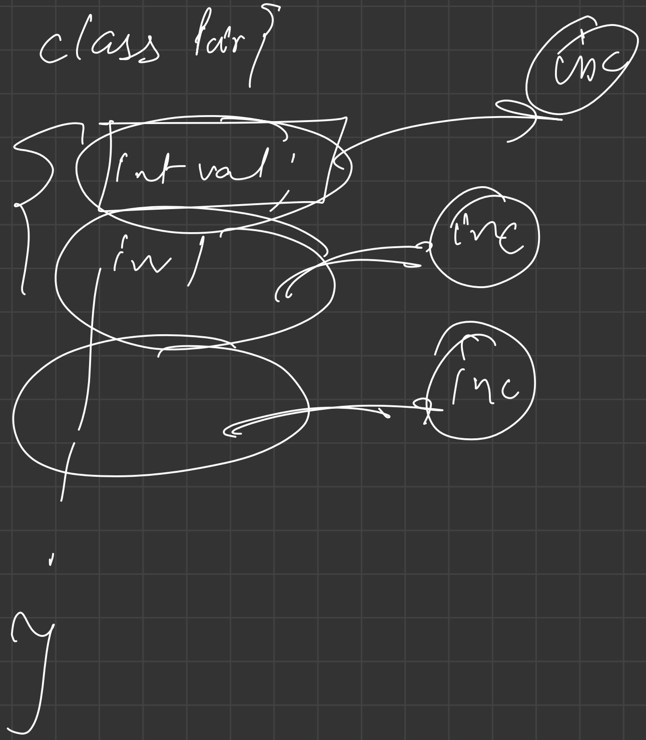
pre: 10, 20, 40, 50, 80, 30, 60, 90, 70

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

1 2 3 4




10
20
20
30
40
30
50
40
50
60
70


class Pair{
int val;
inv)
inc
inc
inc
}