**Solution@18-082023**

**349. Intersection of Two Arrays**

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        Set<Integer> set = new HashSet<>();
        for (int num : nums1) set.add(num);
        Set<Integer> rst = new HashSet<>();
        for (int num: nums2) {
            if (set.contains(num)) rst.add(num);
        }
        int i = 0;
        int[] result = new int[rst.size()];
        for (int num : rst) result[i++] = num;
        return result;
    }
}
```

```
public class Codec {
    private final String DELI = ",";

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuffer sb = new StringBuffer();
        appendString(root, sb);
        return sb.toString();
    }
    private void appendString(TreeNode node, StringBuffer sb) {
        if (node == null) return;
        sb.append(node.val).append(DELI);
        appendString(node.left, sb);
```

```
        appendString(node.right, sb);
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if (data.length() == 0) return null;
        Queue<String> queue = new LinkedList<>(Arrays.asList(data.split(DELI)));
        return buildTree(queue, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    private TreeNode buildTree(Queue<String> queue, int min, int max) {
        if (queue.isEmpty()) return null;
        String s = queue.peek();
        int currVal = Integer.parseInt(queue.peek());
        if (currVal < min || currVal > max) return null;
        queue.poll();
        TreeNode node = new TreeNode(currVal);
        node.left = buildTree(queue, min, currVal);
        node.right = buildTree(queue, currVal, max);
        return node;
    }
}

// DFS w/o utilizing Binary Search Tree
public class Codec {
    private final String DELI = ",";
    private final String NULL = "#";

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuffer sb = new StringBuffer();
```

```
        appendString(root, sb);
        return sb.toString();
    }
    private void appendString(TreeNode node, StringBuffer sb) {
        if (node == null) {
            sb.append(NULL).append(DELI);
        } else {
            sb.append(node.val).append(DELI);
            appendString(node.left, sb);
            appendString(node.right, sb);
        }
    }
```

**449. Serialize and Deserialize BST**

```
    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        Queue<String> queue = new LinkedList<>(Arrays.asList(data.split(DELI)));
        return buildTree(queue);
    }

    private TreeNode buildTree(Queue<String> queue) {
        String val = queue.poll();
        if (val.equals(NULL)) return null;
        TreeNode node = new TreeNode(Integer.parseInt(val));
        node.left = buildTree(queue);
        node.right = buildTree(queue);
        return node;
    }
}
```