Merge Sort

```
function mergeSort(arr) {
    if (arr.length > 1) {
        const mid = Math.floor(arr.length / 2);
        let left = arr.slice(0, mid);
        let right = arr.slice(mid);
        console.log(left, "ff");
        left = mergeSort(left);
        console.log(left);
        right = mergeSort(right);
        let k = 0, i = 0, j = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) {
                arr[k] = left[i];
                i += 1;
            } else {
                arr[k] = right[j];
                j += 1;
            }
            k += 1;
        }
        while (i < left.length) {
            arr[k] = left[i];
            i += 1;
            k += 1;
        }
        while (j < right.length) {
            arr[k] = right[j];
            j += 1;
            k += 1;
        }
    }
    return arr;
}

console.log(mergeSort([4,2,3,1,5]));
```

349. Intersection of Two Arrays

```
var intersection = function(nums1, nums2) {
    const set1 = new Set(nums1);
    const result = new Set();
    for (let num of nums2) {
      if (set1.has(num)) {
        result.add(num);
      }
    }
    return Array.from(result);
};
```

94. Binary Tree Inorder Traversal

We will traverse our tree in orderly. TC is O(n), SC is O(n)

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[]}
 */
var inorderTraversal = function(root) {
    result = [];
    stack = [];
    current = root;
    while (stack.length > 0 || current) {
        while (current) {
            stack.push(current);
            current = current.left;
        }
        current = stack.pop();
        console.log(current.val);
        result.push(current.val);
        current = current.right;
    }
    return result;
};
```

100. Same Tree

We will check whether p and q are same trees recursively. TC is O(n), SC is O(n)

```
var isSameTree = function(p, q) {
    if (!p || !q) {
        return p === q;
    }
    return p.val === q.val && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
};
```

102. Binary Tree Level Order Traversal

We will traverse all nodes layer by layer. TC is O(n), SC is O(n)

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[][]}
 */
var levelOrder = function(root) {
    const result = [];
    let next_ite;
    let cur_level_vals;
    let cur;
    if (!root) {
        return result;
    } else {
        cur = [root];
    }
    while (cur.length > 0) {
        next_ite = [];
        cur_level_vals = [];
        for (let node of cur) {
            if (node.left) {
                next_ite.push(node.left)
            }
            if (node.right) {
                next_ite.push(node.right)
            }
            cur_level_vals.push(node.val);
        }
        result.push(cur_level_vals)
        cur = next_ite;
    }
    return result;
};
```