JavaScript Logical Set Basic to Hard

1. **Reverse an array:**

```
function reverseArray(arr) {
    let start = []
    for (let i = arr.length-1;i>=1;i--){
        start.push(arr[i])
    }
   return start
}
```

2. **Find the maximum element in an array:**

```
function findMaxElement(arr) {
    let max = arr[0];
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

3. **Find the minimum element in an array:**

```
function findMinElement(arr) {
    let min = arr[0];
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    return min;
}
```

4. **Find the sum of all elements in an array:**

```
function findArraySum(arr) {
    let sum = 0;
    for (let i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
    return sum;
}
```

5. **Calculate the average of elements in an array:**

```
function calculateArrayAverage(arr) {
    let sum = 0;
    for (let i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
    return sum / arr.length;
}
```

6. **Find the second largest element in an array:**

```
function findSecondLargest(arr) {
    let firstMax = arr[0];
    let secondMax = null;

    for (let i = 1; i < arr.length; i++) {
        if (arr[i] > firstMax) {
            secondMax = firstMax;
            firstMax = arr[i];
        } else if (arr[i] !== firstMax && (secondMax === null || arr[i] >
secondMax)) {
            secondMax = arr[i];
        }
    }

    return secondMax;
}
```

7. **Find the second smallest element in an array:**

```
function findSecondSmallest(arr) {
    let firstMin = arr[0];
    let secondMin = null;

    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < firstMin) {
            secondMin = firstMin;
            firstMin = arr[i];
        } else if (arr[i] !== firstMin && (secondMin === null || arr[i] <
secondMin)) {
            secondMin = arr[i];
        }
    }

    return secondMin;
}
```

8. **Count the number of even elements in an array:**

```
function countEvenNumbers(arr) {
    let count = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] % 2 === 0) {
            count++;
        }
    }
    return count;
}
```

9. **Count the number of odd elements in an array:**

```
function countOddNumbers(arr) {
    let count = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] % 2 !== 0) {
            count++;
        }
    }
    return count;
}
```

10. **Check if an array is sorted in ascending order:**

```
function isSortedAscending(arr) {
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < arr[i - 1]) {
            return false;
        }
    }
    return true;
}
```

11. **Check if an array is sorted in descending order:**

```
function isSortedDescending(arr) {
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] > arr[i - 1]) {
            return false;
        }
    }
    return true;
}
```

12. **Remove duplicates from an array:**

```
function removeDuplicates(arr) {
    let uniqueArr = [];
    for (let i = 0; i < arr.length; i++) {
        if (!uniqueArr.includes(arr[i])) {
            uniqueArr.push(arr[i]);
        }
    }
    return uniqueArr;
}
```

13. **Find the intersection of two arrays:**

```
function findIntersection(arr1, arr2) {
    let intersection = [];
    for (let i = 0; i < arr1.length; i++) {
        if (arr2.includes(arr1[i]) && !intersection.includes(arr1[i])) {
            intersection.push(arr1[i]);
        }
    }
    return intersection;
}
```

14. **Find the union of two arrays:**

```
function findUnion(arr1, arr2) {
    let union = [...arr1];
    for (let i = 0; i < arr2.length; i++) {
        if (!union.includes(arr2[i])) {
            union.push(arr2[i]);
        }
    }
    return union;
}
```

15. **Find the missing number in an array of 1 to N:**

```
function findMissingNumber(arr) {
    const n = arr.length + 1; // N is the length of the array + 1
    const expectedSum = (n * (n + 1)) / 2;
    const actualSum = arr.reduce((sum, num) => sum + num, 0);
    return expectedSum - actualSum;
}
```

16. **Move all zeros to the end of an array:**

```
function moveZerosToEnd(arr) {
    let nonZeros = arr.filter(num => num !== 0);
    let zeros = Array(arr.length - nonZeros.length).fill(0);
    return nonZeros.concat(zeros);
}
```

17. **Rotate an array to the right by K positions:**

```
function rotateArray(arr, k) {
    k = k % arr.length; // Handle cases where k is greater than array length
    let rotatedPart = arr.splice(-k);
    return rotatedPart.concat(arr);
}
```

18. **Find the "Kth" largest element in an array:**

```
function findKthLargest(arr, k) {
    arr.sort((a, b) => b - a); // Sort in descending order
    return arr[k - 1];
}
```

19. **Find the "Kth" smallest element in an array:**

```
function findKthSmallest(arr, k) {
    arr.sort((a, b) => a - b); // Sort in ascending order
    return arr[k - 1];
}
```

20. **Implement a linear search algorithm:**

```
function linearSearch(arr, target) {
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === target) {
            return i; // Return the index if found
        }
    }
    return -1; // Return -1 if not found
}
```

Certainly! Here are the JavaScript implementations for the additional array-related questions without using built-in methods:

21. **Implement a binary search algorithm:**

```
function binarySearch(arr, target) {
    let low = 0;
    let high = arr.length - 1;

    while (low <= high) {
        let mid = Math.floor((low + high) / 2);
        if (arr[mid] === target) {
            return mid; // Element found
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1; // Element not found
}
```

22. **Count occurrences of an element in an array:**

```
function countOccurrences(arr, target) {
    let count = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === target) {
            count++;
        }
    }
    return count;
}
```

23. **Find the majority element (element that appears more than n/2 times):**

```
function findMajorityElement(arr) {
    let candidate = null;
    let count = 0;

    for (let i = 0; i < arr.length; i++) {
        if (count === 0) {
            candidate = arr[i];
            count = 1;
        } else if (arr[i] === candidate) {
            count++;
        } else {
            count--;
        }
    }

    // Validate if the candidate is the majority element
    count = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === candidate) {
            count++;
        }
    }

    return count > arr.length / 2 ? candidate : null;
}
```

24. **Find the leaders in an array (elements with no element greater to its right):**

```
function findLeaders(arr) {
    const leaders = [];
    let maxRight = arr[arr.length - 1];
    leaders.push(maxRight);

    for (let i = arr.length - 2; i >= 0; i--) {
        if (arr[i] > maxRight) {
            maxRight = arr[i];
            leaders.unshift(maxRight);
        }
    }

    return leaders;
}
```

25. **Find the equilibrium index of an array (sum of elements on the left equals sum on the right):**

```
function findEquilibriumIndex(arr) {
    let totalSum = arr.reduce((sum, num) => sum + num, 0);
    let leftSum = 0;

    for (let i = 0; i < arr.length; i++) {
        totalSum -= arr[i];
        if (leftSum === totalSum) {
            return i;
        }
        leftSum += arr[i];
    }

    return -1; // Equilibrium index not found
}
```

26. **Implement a stack using an array:**

```
function Stack() {
    this.items = [];

    this.push = function (element) {
        this.items.push(element);
    };

    this.pop = function () {
        if (this.items.length === 0) {
            return null;
        }
        return this.items.pop();
    };

    this.peek = function () {
        return this.items[this.items.length - 1];
    };

    this.isEmpty = function () {
        return this.items.length === 0;
    };

    this.size = function () {
        return this.items.length;
    };
}
```

27. **Implement a queue using an array:**

```
function Queue() {
    this.items = [];

    this.enqueue = function (element) {
        this.items.push(element);
    };

    this.dequeue = function () {
        if (this.items.length === 0) {
            return null;
        }
        return this.items.shift();
    };

    this.front = function () {
        return this.items[0];
    };
    this.isEmpty = function () {
        return this.items.length === 0;
    };

    this.size = function () {
        return this.items.length;
    };
}
```

28. **Implement two stacks in an array:**

```
function TwoStacks() {
    this.items = [];
    this.top1 = -1;
    this.top2 = this.items.length;

    this.push1 = function (element) {
        this.items[++this.top1] = element;
    };

    this.push2 = function (element) {
        this.items[--this.top2] = element;
    };

    this.pop1 = function () {
        if (this.top1 === -1) {
            return null;
        }
        return this.items[this.top1--];
    };

    this.pop2 = function () {
        if (this.top2 === this.items.length) {
            return null;
        }
        return this.items[this.top2++];
    };
}
```

29. **Implement a circular queue:**

```
function CircularQueue(capacity) {
    this.items = new Array(capacity);
    this.front = -1;
    this.rear = -1;
    this.size = 0;

    this.enqueue = function (element) {
        if (this.isEmpty()) {
            this.front = 0;
            this.rear = 0;
        } else {
            this.rear = (this.rear + 1) % capacity;
        }

        this.items[this.rear] = element;
        this.size++;
    };

    this.dequeue = function () {
        if (this.isEmpty()) {
            return null;
        }

        const removed = this.items[this.front];
        this.items[this.front] = null;

        if (this.front === this.rear) {
            this.front = -1;
            this.rear = -1;
        } else {
            this.front = (this.front + 1) % capacity;
        }

        this.size--;
        return removed;
    };

    this.frontValue = function () {
        return this.isEmpty() ? null : this.items[this.front];
    };

    this.isEmpty = function () {
        return this.size === 0;
    };

    this.isFull = function () {
        return this.size === capacity;
    };
}
```

30. **Implement a dynamic array (resizeable array):**

```
function DynamicArray() {
    this.capacity = 1;
    this.size = 0;
    this.items = new Array(this.capacity);

    this.resize = function () {
        this.capacity *= 2;
        const newArray = new Array(this.capacity);
        for (let i = 0; i < this.size; i++) {
            newArray[i] = this.items[i];
        }
        this.items = newArray;
    };

    this.push = function (element) {
        if (this.size === this.capacity) {
            this.resize();
        }
        this.items[this.size++] = element;
    };

    this.pop = function () {
        if (this.size === 0) {
            return null;
        }
        const popped = this.items[--this.size];
        this.items[this.size] = null; // Optional: Clear the last element
        return popped;
    };

    this.get = function (index) {
        if (index < 0 || index >= this.size) {
            return

 null;
        }
        return this.items[index];
    };
}
```

Certainly! Here are the JavaScript implementations for the specified array-related questions without using built-in methods:

31. **Find the largest subarray with equal number of 0s and 1s (Binary Subarray with Equal 0s and 1s):**

```
function findMaxLengthSubarray(arr) {
    const n = arr.length;
    const hashMap = {0: -1};
    let maxLength = 0;
    let count = 0;

    for (let i = 0; i < n; i++) {
        count += arr[i] === 0 ? -1 : 1;

        if (hashMap[count] !== undefined) {
            maxLength = Math.max(maxLength, i - hashMap[count]);
        } else {
            hashMap[count] = i;
        }
    }

    return maxLength;
}
```

32. **Implement an algorithm to rotate an array:**

```
function rotateArray(arr, k) {
    const n = arr.length;
    k = k % n; // Handle cases where k is greater than array length

    reverseArray(arr, 0, n - 1);
    reverseArray(arr, 0, k - 1);
    reverseArray(arr, k, n - 1);

    return arr;
}

function reverseArray(arr, start, end) {
    while (start < end) {
        let temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

33. **Implement an algorithm to sort an array using Bubble Sort:**

```
function bubbleSort(arr) {
    const n = arr.length;

    for (let i = 0; i < n - 1; i++) {
        for (let j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap if the element is greater than the next element
                let temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    return arr;
}
```

34. **Implement an algorithm to sort an array using Selection Sort:**

```
function selectionSort(arr) {
    const n = arr.length;

    for (let i = 0; i < n - 1; i++) {
        let minIndex = i;

        for (let j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                // Update minIndex if a smaller element is found
                minIndex = j;
            }
        }

        // Swap the found minimum element with the element at index i
        let temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }

    return arr;
}
```

35. **Implement an algorithm to sort an array using Insertion Sort:**

```
function insertionSort(arr) {
    const n = arr.length;

    for (let i = 1; i < n; i++) {
        let key = arr[i];
        let j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }

    return arr;
}
```

36. **Implement an algorithm to sort an array using Merge Sort:**

```
function mergeSort(arr) {
    if (arr.length <= 1) {
        return arr;
    }
    const mid = Math.floor(arr.length / 2);
    const left = mergeSort(arr.slice(0, mid));
    const right = mergeSort(arr.slice(mid));

    return merge(left, right);
}

function merge(left, right) {
    let result = [];
    let i = 0;
    let j = 0;

    while (i < left.length && j < right.length) {
        if (left[i] < right[j]) {
            result.push(left[i]);
            i++;
        } else {
            result.push(right[j]);
            j++;
        }
    }

    return result.concat(left.slice(i), right.slice(j));
}
```

37. **Implement an algorithm to sort an array using Quick Sort:**

```
function quickSort(arr) {
    if (arr.length <= 1) {
        return arr;
    }

    const pivot = arr[0];
    const left = [];
    const right = [];

    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < pivot) {
            left.push(arr[i]);
        } else {
            right.push(arr[i]);
        }
    }

    return quickSort(left).concat(pivot, quickSort(right));
}
```

38. **Implement an algorithm to sort an array using Heap Sort:**

```
function heapSort(arr) {
    const n = arr.length;

    // Build a max heap
    for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // Extract elements from the heap one by one
    for (let i = n - 1; i > 0; i--) {
        // Swap the root (maximum element) with the last element
        let temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // Call heapify on the reduced heap
        heapify(arr, i, 0);
    }

    return arr;
}

function heapify(arr, n, i) {
    let largest = i;
    let left = 2 * i + 1;
    let right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {


        largest = left;
    }

    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest !== i) {
        // Swap arr[i] and arr[largest]
        let temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}
```

39. **Find the maximum product subarray:**

```
function maxProductSubarray(arr) {
    const n = arr.length;

    if (n === 0) {
        return 0;
    }

    let maxProduct = arr[0];
    let minProduct = arr[0];
    let result = arr[0];

    for (let i = 1; i < n; i++) {
        if (arr[i] < 0) {
            // Swap max and min products when encountering a negative number
            [maxProduct, minProduct] = [minProduct, maxProduct];
        }

        maxProduct = Math.max(arr[i], maxProduct * arr[i]);
        minProduct = Math.min(arr[i], minProduct * arr[i]);

        result = Math.max(result, maxProduct);
    }

    return result;
}
```

40. **Find the longest increasing subsequence in an array:**

```
function longestIncreasingSubsequence(arr) {
    const n = arr.length;
    const lis = new Array(n).fill(1);

    for (let i = 1; i < n; i++) {
        for (let j = 0; j < i; j++) {
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1) {
                lis[i] = lis[j] + 1;
            }
        }
    }

    return Math.max(...lis);
}
```

Certainly! Here are the JavaScript implementations for the specified array-related questions without using built-in methods:

41. **Find the "Kth" largest and "Kth" smallest element in an unsorted array:**

```
function findKthLargestAndSmallest(arr, k) {
    arr.sort((a, b) => a - b);

    const kthSmallest = arr[k - 1];
    const kthLargest = arr[arr.length - k];

    return { kthSmallest, kthLargest };
}
```

42. **Implement an algorithm to reverse a group of "K" elements in an array:**

```
function reverseGroups(arr, k) {
    const n = arr.length;

    for (let i = 0; i < n; i += k) {
        let start = i;
        let end = Math.min(i + k - 1, n - 1);

        while (start < end) {
            // Swap elements at start and end indices
            let temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;

            start++;
            end--;
        }
    }

    return arr;
}
```

43. **Implement an algorithm to find the median of two sorted arrays:**

```
function findMedianSortedArrays(nums1, nums2) {
    const merged = [...nums1, ...nums2].sort((a, b) => a - b);
    const n = merged.length;

    if (n % 2 === 0) {
        // If the length is even, return the average of the middle elements
        return (merged[n / 2 - 1] + merged[n / 2]) / 2;
    } else {
        // If the length is odd, return the middle element
        return merged[Math.floor(n / 2)];
    }
}
```

44. **Implement an algorithm to rearrange positive and negative numbers alternatively:**

```
function rearrangePositiveNegative(arr) {
    const n = arr.length;
    let positiveIndex = 0;

    for (let i = 0; i < n; i++) {
        if (arr[i] < 0) {
            // Rotate the array to move negative numbers to the front
            const temp = arr[i];
            arr.splice(i, 1);
            arr.unshift(temp);

            positiveIndex++;
        }
    }

    // Rearrange positive and negative numbers alternatively
    for (let i = 0; i < n - 1 && positiveIndex < n; i += 2) {
        if (arr[i] < 0) {
            // Swap negative and positive numbers
            const temp = arr[i];
            arr[i] = arr[positiveIndex];
            arr[positiveIndex] = temp;

            positiveIndex++;
        }
    }

    return arr;
}
```

45. **Implement an algorithm to find the contiguous subarray with the largest sum (Kadane's Algorithm):**

```
function maxSubarraySum(arr) {
    let maxSum = arr[0];
    let currentSum = arr[0];

    for (let i = 1; i < arr.length; i++) {
        currentSum = Math.max(arr[i], currentSum + arr[i]);
        maxSum = Math.max(maxSum, currentSum);
    }

    return maxSum;
}
```

46. **Implement an algorithm to rotate a 2D array (matrix) by 90 degrees:**

```
function rotateMatrix(matrix) {
    const n = matrix.length;

    // Transpose the matrix
    for (let i = 0; i < n; i++) {
        for (let j = i; j < n; j++) {
            [matrix[i][j], matrix[j][i]] = [matrix[j][i], matrix[i][j]];
        }
    }

    // Reverse each row to get the rotated matrix
    for (let i = 0; i < n; i++) {
        matrix[i].reverse();
    }

    return matrix;
}
```

47. **Implement an algorithm to find the common elements in three sorted arrays:**

```
function findCommonElements(arr1, arr2, arr3) {
    const commonElements = [];
    let i = 0, j = 0, k = 0;

    while (i < arr1.length && j < arr2.length && k < arr3.length) {
        if (arr1[i] === arr2[j] && arr2[j] === arr3[k]) {
            commonElements.push(arr1[i]);
            i++;
            j++;
            k++;
        } else if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr2[j] < arr3[k]) {
            j++;
        } else {
            k++;
        }
    }

    return commonElements;
}
```

48. **Implement an algorithm to search in a rotated sorted array:**

```
function searchInRotatedArray(arr, target) {
    let low = 0;
    let high = arr.length - 1;

    while (low <= high) {
        let mid = Math.floor((low + high) / 2);

        if (arr[mid] === target) {
            return mid; // Element found
        }

        // Check which half is sorted, and perform binary search accordingly
        if (arr[low] <= arr[mid]) {
            if (target >= arr[low] &&

 target < arr[mid]) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        } else {
            if (target > arr[mid] && target <= arr[high]) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }

    return -1; // Element not found
}
```

49. **Implement an algorithm to segregate 0s and 1s in an array:**

```
function segregateZerosAndOnes(arr) {
    let low = 0;
    let high = arr.length - 1;

    while (low < high) {
        while (arr[low] === 0 && low < high) {
            low++;
        }

        while (arr[high] === 1 && low < high) {
            high--;
        }

        if (low < high) {
            // Swap arr[low] and arr[high]
            let temp = arr[low];
            arr[low] = arr[high];
            arr[high] = temp;

            low++;
            high--;
        }
    }
    return arr;
}
```

50. **Implement an algorithm to find the first repeating element in an array:**

```
function findFirstRepeatingElement(arr) {
    const seen = new Set();

    for (let i = 0; i < arr.length; i++) {
        if (seen.has(arr[i])) {
            return arr[i];
        }

        seen.add(arr[i]);
    }

    return -1; // No repeating element found
}
```

These implementations should cover the specified array-related questions without using built-in methods. Feel free to test them with different arrays!

These implementations should cover the specified array-related questions without using built-in methods. Feel free to test them with different arrays!