Sliding Window Handbook (Arrays + Strings)

Arrays Q1: Maximum Sum Subarray of Size K

Problem: Given an array and an integer k, find the maximum sum of any contiguous subarray of size k. Input: arr = [2,1,5,1,3,2], k=3 Output: 9 (subarray [5,1,3])

```
function maxSumSubarray(arr, k) {
    let sum = 0, maxSum = 0;
    for (let i = 0; i < k; i++) sum += arr[i];
    maxSum = sum;
    for (let i = k; i < arr.length; i++) {
        sum += arr[i] - arr[i - k];
        maxSum = Math.max(maxSum, sum);
    }
    return maxSum;
}
```

Arrays Q2: Longest Subarray with Sum ≤ K

Problem: Find the longest subarray length such that its sum ≤ K. Input: arr = [1,2,1,0,1,1,0], k=4 Output: 5

```
function longestSubarraySumK(arr, k) {
    let left = 0, sum = 0, maxLen = 0;
    for (let right = 0; right < arr.length; right++) {
        sum += arr[right];
        while (sum > k) sum -= arr[left++];
        maxLen = Math.max(maxLen, right - left + 1);
    }
    return maxLen;
}
```

Arrays Q3: Longest Subarray Without Forbidden Value

Problem: Find the longest continuous segment without the forbidden value. Input: arr=[1,3,5,3,6,7,3,5,6], forbidden=3 Output: 2

```
function longestSafeSegment(arr, forbidden) {
    let maxLen = 0, currentLen = 0;
    for (let num of arr) {
        if (num === forbidden) currentLen = 0;
        else {
            currentLen++;
            maxLen = Math.max(maxLen, currentLen);
        }
    }
    return maxLen;
}
```

Arrays Q4: Sliding Window Maximum

Problem: Return the maximum for every window of size k. Input: arr=[1,3,-1,-3,5,3,6,7], k=3 Output: [3,3,5,5,6,7]

```
function maxSlidingWindow(nums, k) {
    let deque = [], res = [];
    for (let i = 0; i < nums.length; i++) {
        while (deque.length && deque[0] <= i - k) deque.shift();
        while (deque.length && nums[deque[deque.length - 1]] < nums[i]) deque.pop();
        deque.push(i);
        if (i >= k - 1) res.push(nums[deque[0]]);
    }
    return res;
}
```

Arrays Q5: Smallest Subarray with Sum ≥ K

Problem: Find the length of the smallest contiguous subarray with sum ≥ K. Input: arr=[2,3,1,2,4,3], target=7 Output: 2 ([4,3])

```
function minSubarrayLen(target, arr) {
    let left = 0, sum = 0, minLen = Infinity;
    for (let right = 0; right < arr.length; right++) {
        sum += arr[right];
        while (sum >= target) {
            minLen = Math.min(minLen, right - left + 1);
            sum -= arr[left++];
        }
    }
    return minLen === Infinity ? 0 : minLen;
}
```

Strings Q1: Longest Substring Without Repeating Characters

Problem: Find the length of the longest substring without repeating characters. Input: s='abcabcbb' Output: 3 ('abc')

```
function lengthOfLongestSubstring(s) {
    let set = new Set(), left = 0, maxLen = 0;
    for (let right = 0; right < s.length; right++) {
        while (set.has(s[right])) set.delete(s[left++]);
        set.add(s[right]);
        maxLen = Math.max(maxLen, right - left + 1);
    }
    return maxLen;
}
```

Strings Q2: Longest Substring with At Most K Distinct Characters

Problem: Find the longest substring with at most K distinct characters. Input: s='eceba', k=2 Output: 3 ('ece')

```
function longestSubstringKDistinct(s, k) {
    let map = new Map(), left = 0, maxLen = 0;
    for (let right = 0; right < s.length; right++) {
        map.set(s[right], (map.get(s[right]) || 0) + 1);
        while (map.size > k) {
            map.set(s[left], map.get(s[left]) - 1);
            if (map.get(s[left]) === 0) map.delete(s[left]);
            left++;
        }
        maxLen = Math.max(maxLen, right - left + 1);
    }
    return maxLen;
}
```

Strings Q3: Minimum Window Substring

Problem: Find the smallest substring in s that contains all characters of t. Input: s='ADOBECODEBANC', t='ABC' Output: 'BANC'

```
function minWindow(s, t) {
    let need = new Map(), have = new Map();
    for (let c of t) need.set(c, (need.get(c) || 0) + 1);
    let left = 0, count = 0, minLen = Infinity, res = "";
    for (let right = 0; right < s.length; right++) {
        let c = s[right];
        have.set(c, (have.get(c) || 0) + 1);
        if (need.has(c) && have.get(c) <= need.get(c)) count++;
        while (count === t.length) {
            if (right - left + 1 < minLen) {
                minLen = right - left + 1;
                res = s.substring(left, right + 1);
            }
            let lc = s[left++];
            have.set(lc, have.get(lc) - 1);
            if (need.has(lc) && have.get(lc) < need.get(lc)) count--;
        }
    }
    return res;
}
```

Strings Q4: Longest Repeating Character Replacement

Problem: Replace at most k characters to make the longest repeating substring. Input: s='AABABBA', k=1 Output: 4

```
function characterReplacement(s, k) {
    let count = {}, maxFreq = 0, left = 0, maxLen = 0;
    for (let right = 0; right < s.length; right++) {
        count[s[right]] = (count[s[right]] || 0) + 1;
        maxFreq = Math.max(maxFreq, count[s[right]]);
        while ((right - left + 1) - maxFreq > k) {
            count[s[left]]--;
            left++;
        }
        maxLen = Math.max(maxLen, right - left + 1);
    }
    return maxLen;
}
```

Strings Q5: Find All Anagrams in a String

Problem: Find all start indices of substrings in s that are anagrams of p. Input: s='cbaebabacd', p='abc' Output: [0,6]

```
function findAnagrams(s, p) {
    let res = [], need = {}, window = {};
    for (let c of p) need[c] = (need[c] || 0) + 1;
    let left = 0, right = 0, match = 0;
    while (right < s.length) {
        let c = s[right++];
        window[c] = (window[c] || 0) + 1;
        if (need[c] && window[c] === need[c]) match++;
        while (right - left >= p.length) {
            if (match === Object.keys(need).length) res.push(left);
            let lc = s[left++];
            if (need[lc] && window[lc] === need[lc]) match--;
            window[lc]--;
        }
    }
    return res;
}
```