**Sliding Window Technique**

**1. Fixed Sliding Window Problems**

function maxSumSubarray(arr, k) {

let maxSum = 0;

let windowSum = 0;

// Calculate the sum of the first k elements

for (let i = 0; i < k; i++) {

windowSum += arr[i];

}

maxSum = windowSum;

// Slide the window from start to end

for (let i = k; i < arr.length; i++) {

windowSum += arr[i] - arr[i - k];

maxSum = Math.max(maxSum, windowSum);

}

return maxSum;

}

// Example usage

console.log(maxSumSubarray([1, 2, 3, 4, 5, 6, 7, 8], 3)); // Output: 21

* Maximum Sum Subarray of Size K
* First Negative Integer in Every Window of Size K
* Average of Subarrays of Size K

**2. Dynamic Sliding Window Problems**

function minSubarrayLen(arr, target) {

let minLength = Infinity;

let windowSum = 0;

let start = 0;

for (let end = 0; end < arr.length; end++) {

windowSum += arr[end];

while (windowSum >= target) {

minLength = Math.min(minLength, end - start + 1);

windowSum -= arr[start];

start++;

}

}

return minLength === Infinity ? 0 : minLength;

}

// Example usage

console.log(minSubarrayLen([2, 3, 1, 2, 4, 3], 7)); // Output: 2

* Smallest Subarray with a Given Sum
* Longest Substring with K Distinct Characters
* Fruits into Baskets (LeetCode)

**3. Caterpillar Method Problems**

function countSubarrays(arr, target) {

let count = 0;

let windowSum = 0;

let start = 0;

for (let end = 0; end < arr.length; end++) {

windowSum += arr[end];

while (windowSum > target) {

windowSum -= arr[start];

start++;

}

if (windowSum === target) {

count++;

}

}

return count;

}

// Example usage

console.log(countSubarrays([1, 2, 3, 4, 2, 3], 6)); // Output: 3

* Number of Subarrays with Sum Equals K
* Longest Subarray with Sum Less than or Equal to K
* Subarray Product Less than K (LeetCode)

**4. Expanding/Shrinking Sliding Window Problems**

function lengthOfLongestSubstring(s) {

const charSet = new Set();

let maxLength = 0;

let start = 0;

for (let end = 0; end < s.length; end++) {

while (charSet.has(s[end])) {

charSet.delete(s[start]);

start++;

}

charSet.add(s[end]);

maxLength = Math.max(maxLength, end - start + 1);

}

return maxLength;

}

// Example usage

console.log(lengthOfLongestSubstring("abcabcbb")); // Output: 3

* Longest Substring Without Repeating Characters
* Longest Substring with At Most Two Distinct Characters
* Permutation in String (LeetCode)

***Important LeetCode Problems***

**Fixed Sliding Window:**

* Maximum Sum of K Consecutive Elements
* First Negative Integer in Every Window of Size K
* Average of Subarrays of Size K

**Dynamic Sliding Window:**

* Smallest Subarray with a Given Sum
* Longest Substring with K Distinct Characters
* Fruits into Baskets

**Caterpillar Method:**

* Number of Subarrays with Sum Equals K
* Longest Subarray with Sum Less than or Equal to K
* Subarray Product Less than K

**Expanding/Shrinking Sliding Window:**

* Longest Substring Without Repeating Characters
* Longest Substring with At Most Two Distinct Characters
* Permutation in String
* Minimum Window Substring
* Longest Repeating Character Replacement
* Find All Anagrams in a String