Sliding Window Concept - Beginner Friendly Guide

The Sliding Window technique is used to efficiently process subarrays or substrings in an array or string.
Instead of using slow methods (like nested loops), we move a "window" over the data, making our code much faster.

Why Use Sliding Window?
- If the problem asks to find or optimize a subarray (continuous part of an array).
- If a brute-force (nested loops) solution is too slow.
- If you need to find max/min/length/sum of a subarray or substring.

Types of Sliding Window:
1. Fixed-Size Sliding Window - The window size stays the same throughout.
   Example: "Find the maximum sum of a subarray of size K".
2. Variable-Size Sliding Window - The window size changes dynamically.
   Example: "Find the smallest subarray with sum >= S".

Step-by-Step Guide to Implement Sliding Window:

Example 1: Fixed-Size Sliding Window (Max Sum of K Elements)
Brute Force Approach ($O(N^2)$):

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

Optimized Approach (Sliding Window) - O(N):
Instead of recalculating the sum for each window, we:
1. First, calculate the sum of the first K elements.

2. Then slide the window forward by adding the new element and removing the first one.
3. Keep updating the max sum.

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

Example 2: Variable-Size Sliding Window (Smallest Subarray with Sum >= S)
1. Expand the window until the sum >= S.
2. Then, shrink the window from the left to find the smallest possible subarray.

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

List of Problems Solved Using Sliding Window:

Easy Problems:
1. Maximum sum of K-sized subarray
2. Find the average of all K-sized subarrays
3. Check if a string contains a permutation of another string
4. Longest substring without repeating characters

Medium Problems:

5. Smallest subarray with sum >= S
6. Longest substring with at most K distinct characters
7. Longest repeating character replacement
8. Sliding window maximum

Hard Problems:

9. Substring with concatenation of all words
10. Minimum window substring
11. Find all anagrams of a pattern in a string
12. Longest subarray with sum <= K

Summary:

- Sliding Window helps optimize problems involving contiguous subarrays/substrings.
- Fixed-size window: Used when a subarray size is given.
- Variable-size window: Used when constraints like sum >= S exist.
- Reduces complexity from O(N^2) to O(N) in many cases.