Sliding Window Technique in DSA: Beginner to Advanced Guide

Subtitle: Master the sliding window approach to efficiently solve subarray and substring problems in linear time.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

The Sliding Window technique is used to solve problems involving **contiguous subarrays or substrings** efficiently. Instead of checking all possible windows ($O(n^2)$), this approach often reduces complexity to **O(n)**. This guide covers patterns, examples, and exercises.

Step 1: What is Sliding Window?

- Maintain a **window (subarray/subsequence)** with start and end indices
- Slide the window across the array/string while maintaining required properties
- Often used with **sums, counts, or max/min conditions**

Example: Maximum sum of subarray of size k

```
let arr = [1,2,3,4,5], k = 3;

let maxSum = 0, sum = 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);

}

console.log(maxSum); // 12
```

Step 2: Types of Sliding Window

1. Fixed Window

- Window size is fixed (e.g., subarray of size k)
- Maintain sum, max, or other properties

2. Variable / Dynamic Window

- Window size adjusts based on condition
- Example: longest substring with K distinct characters

Step 3: Common Problems

1. Maximum/Minimum Sum Subarray of Size k

- Use fixed window, update sum efficiently

2. Longest Substring Without Repeating Characters

- Dynamic window with set/hash map

```
let s = "abcabcbb";

let set = new Set();

let left = 0, maxLen = 0;



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

  while (set.has(s[right])) {

    set.delete(s[left]);

    left++;

  }

  set.add(s[right]);

  maxLen = Math.max(maxLen, right - left + 1);

}

console.log(maxLen); // 3
```

3. Minimum Size Subarray Sum ≥ target

- Dynamic window, shrink from left when sum ≥ target

4. Sliding Window Maximum

- Maintain deque for O(n) maximums in subarrays of size k

Step 4: Advantages of Sliding Window

- Reduces complexity from **$O(n^2) \rightarrow O(n)$**
- Efficient for problems on **contiguous sequences**
- Works well with **arrays and strings**

Step 5: Common Mistakes

- Forgetting to **shrink window** in dynamic cases
- Mismanaging indices → off-by-one errors
- Not updating max/min during sliding
- Using hash sets incorrectly for duplicates

Step 6: Practice Plan for Sliding Window

Day 1:

- Fixed size sum/max subarray → 5 problems

Day 2:

- Longest substring without repeating characters → 5 problems

Day 3:

- Minimum window substring → 5 problems

Day 4:

- Sliding window maximum / deque problems → 5 problems

Day 5–6:

- Mixed sliding window problems → 10 problems

Mini Exercises

1. Count number of subarrays with sum = k
2. Longest substring with at most k distinct characters
3. Maximum of all contiguous subarrays of size k
4. Smallest subarray with sum ≥ target

Key Takeaways

- Sliding window is **efficient for contiguous sequences**
- Fixed or dynamic window depending on problem

- Combine with **hash maps, sets, or deque** for optimization
- Reduces time complexity significantly
- Practice improves **intuition on window expansion/shrinking**

Visit haas.dev for more DSA sliding window guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app