

Monotonicity & Two-Pointers — 1-Page Interview Cheat Sheet

🔑 What does *Monotonic* mean?

Monotonic = moves in only ONE direction (never reverses)

- **Monotonic Increasing**: always goes up or stays same
 - **Monotonic Decreasing**: always goes down or stays same
 - **Not Monotonic**: goes up *and* down
-

Why Monotonicity Matters in Interviews

Many O(n) solutions rely on **predictable behavior**.

If expanding or shrinking a window changes the state in **one direction only**, we can use:
- Two pointers / sliding window
- Monotonic stack or queue

If not → these techniques **fail**.

🔗 When Two-Pointers WORK (Important Distinction)

⌚ Case 1: Classic Two-Pointers (Array Ends)

Requires sorted input (or sorting first)

Used when pointer movement depends on **value comparison**.

Examples: - Move left → value decreases - Move right → value increases

Array must be sorted to make this true.

Examples: - Two Sum II (167) - 3Sum / 4Sum (after sorting) - Squares of a Sorted Array

⌚ Case 2: Sliding Window

Does NOT require sorted input

Used when:
- Working with **contiguous subarrays / substrings**
- Pointer movement depends on **window state**, not value order

 Sorting would actually break the problem.

Examples: - Minimum Window Substring - Longest Repeating Character Replacement - Subarray Product Less Than K

When Two-Pointers FAIL

Negative numbers present (for sum problems)

- Adding may decrease sum
- Removing may increase sum

► Breaks monotonic behavior

Counting ALL valid subarrays

- Multiple valid windows can end at the same index
 - Sliding window tracks only one window
-

Need memory of earlier states

- Sliding window forgets previous configurations
-

Non-contiguous selection required

- Two pointers assume contiguous windows or sorted order
-

Sliding Window vs Prefix Sum

Scenario	Sliding Window	Prefix Sum
All nums ≥ 0		
Negative nums		
Count all subarrays		
Find one window		

Monotonic Stack

A stack that stays **sorted in one direction**.

Monotonic Increasing Stack

- Bottom → Top increases
- Used for:
- Next smaller element
- Remove K Digits

Monotonic Decreasing Stack

- Bottom → Top decreases
- Used for:
- Next greater element
- Car Fleet
- 132 Pattern

Key Interview Rules (Memorize)

- **Classic two-pointers** → needs sorted input
- **Sliding window** → needs monotonic window behavior
- **Negative numbers** → sliding window usually fails

🧪 Classic Failure Example (Sliding Window)

```
nums = [1, -1, 1], k = 1
```

Valid subarrays: - [1] - [1, -1, 1] - [1]

Sliding window finds only **2**. Prefix sum finds **3**.

📌 LeetCode Problem Mappings

🔗 Two-Pointers (Sorted Required)

- 1. Two Sum II
- 1. 3Sum
- 1. 3Sum Closest
- 1. 4Sum
- 1. Squares of a Sorted Array

🔗 Sliding Window (No Sorting)

- 1. Minimum Window Substring
- 1. Minimum Size Subarray Sum
- 1. Longest Repeating Character Replacement
- 1. Subarray Product Less Than K

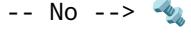
Sliding Window → Prefix Sum

- 1. Subarray Sum Equals K
- 1. Subarray Sums Divisible by K
- 1. Count Number of Nice Subarrays

Monotonic Stack

- 1. Largest Rectangle in Histogram
- 1. Daily Temperatures
- 1. Car Fleet
- 1. 132 Pattern
- 1. Remove K Digits

Decision Flowchart (Text Version)

```
Start
|
v
Is the problem about a contiguous subarray / substring?
| -- No --> Is array sorted?
|           | -- Yes -->  Use Two Pointers
|           | -- No  -->  Sort or use Hashmap
|
v
Does expanding/shrinking the window change the state monotonically?
| -- No -->  Use Prefix Sum / Hashmap / DP
|
v
Are all numbers non-negative?
| -- No -->  Use Prefix Sum
|
v
Do you need to count ALL valid windows?
| -- Yes -->  Use Prefix Sum
|
v
 Use Sliding Window
```

One-Line Interview Summary

"Classic two-pointer techniques require sorted input, while sliding window techniques require monotonic window behavior. If monotonicity breaks, prefix sums or other approaches are needed."