LeetCode 209. Minimum Size Subarray Sum

1. Problem Title & Link

- **209. Minimum Size Subarray Sum**
- https://leetcode.com/problems/minimum-size-subarray-sum/

2. Problem Statement (Short Summary)

Given an array of positive integers nums and a positive integer target,
find the **minimal length** of a **contiguous subarray** whose sum ≥ target.
If no such subarray exists, return 0.

3. Examples (Input → Output)

Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has sum 7 and length 2.

Input: target = 4, nums = [1,4,4]
Output: 1

Input: target = 11, nums = [1,1,1,1,1,1,1]
Output: 0

4. Constraints

- 1 <= target <= 10^9
- 1 <= nums.length <= 10^5
- 1 <= nums[i] <= 10^4

5. Thought Process (Step by Step)

Since all numbers are **positive**, we can use a **sliding window** approach efficiently.

This is a textbook "grow and shrink window" problem 💡

Step 1: Sliding Window Intuition

We maintain a window [left, right] whose sum ≥ target.
Algorithm flow:

1. Expand right to include more elements until sum ≥ target.
2. Then shrink left as much as possible while sum ≥ target (to minimize window size).
3. Keep updating the smallest window length seen.

Step 2: Why Sliding Window Works

- Since all numbers are positive, when we move left++, the sum *decreases*.
- So, no need to revisit prior elements — ensuring **O(n)** complexity.

6. Pseudocode

```
left = 0
sum = 0
min_len = ∞

for right in range(0, n):
   sum += nums[right]

   while sum >= target:
      min_len = min(min_len, right - left + 1)
      sum -= nums[left]
      left += 1

if min_len == ∞:
   return 0
else:
   return min_len
```

7. Code Implementation

✅ **Python**

```
class Solution:
   def minSubArrayLen(self, target: int, nums: List[int]) -> int:
      left = 0
      curr_sum = 0
      min_len = float('inf')

      for right in range(len(nums)):
         curr_sum += nums[right]

         while curr_sum >= target:
            min_len = min(min_len, right - left + 1)
            curr_sum -= nums[left]
            left += 1

      return 0 if min_len == float('inf') else min_len
```

✅ **Java**

```
class Solution {
   public int minSubArrayLen(int target, int[] nums) {
      int left = 0, sum = 0, minLen = Integer.MAX_VALUE;

      for (int right = 0; right < nums.length; right++) {
         sum += nums[right];
         while (sum >= target) {
            minLen = Math.min(minLen, right - left + 1);
            sum -= nums[left];
            left++;
```

```
        }
      }
      return (minLen == Integer.MAX_VALUE) ? 0 : minLen;
    }
}
```

8. Time & Space Complexity

- **Time:** O(n) — each element visited at most twice (once by right, once by left).
- **Space:** O(1) — constant extra memory.

9. Dry Run (Step-by-Step Execution)

👉 Input: target = 7, nums = [2,3,1,2,4,3]

Step	left	right	nums[right]	sum	Condition	Action	min_len
1	0	0	2	2	sum < 7	expand right	∞
2	0	1	3	5	sum < 7	expand right	∞
3	0	2	1	6	sum < 7	expand right	∞
4	0	3	2	8	sum ≥ 7	shrink left → sum=6	4
5	1	4	4	10	sum ≥ 7	shrink left → sum=7	4→3
6	2	4	—	7≥7	shrink left → sum=6	3→2	
7	3	5	3	9	sum ≥ 7	shrink left → sum=7→4	✅ min_len=2

✅ Output: 2 (subarray [4,3])

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Sliding Window (Variable Size) — dynamically adjust window bounds based on cumulative condition.	- Minimum/maximum subarray length - Window-based sum/ product problems - Streaming computations	- Forgetting to shrink window inside loop - Not handling “no valid subarray” → should return 0 - Using sliding window when negatives are allowed (breaks logic)	🔷 Builds to **LeetCode 3 (Longest Substring Without Repeating Characters)** 🔷 Connects to **LC 76 (Minimum Window Substring)** 🔷 Reinforces two-pointer + window shrinking mastery

11. Common Mistakes / Edge Cases

- Forgetting to reset min_len when no subarray found → must return 0.
- Using sliding window on arrays with negative numbers (not valid for this logic).
- Off-by-one errors in window length (right - left + 1).

12. Variations / Follow-Ups

- **LC 904 (Fruits into Baskets)** — longest window with ≤2 distinct elements.
- **LC 76 (Minimum Window Substring)** — character-based version of same logic.
- **LC 862 (Shortest Subarray with Sum ≥ K)** — similar, but allows negatives → requires deque-based prefix sums.