📘 Minimum Size Subarray Sum — Leetcode 209 (Medium)

> 🧠 Problem Statement:

Given an array `nums` of **positive integers** and a **positive integer** `target`, return the **minimal length** of a **contiguous subarray** of which the sum is **greater than or equal to** `target`.

If there is **no such subarray**, return `0`.

🔍 Example:

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

🚀 Approach: Sliding Window Technique 🪟

🤔 Intuition:

Hamare paas ek array hai aur hume dekhna hai ki **minimum length ka subarray** konsa hoga jiska **sum** `target` **se bada ya barabar ho**.

Toh socha jaye toh yeh ek **window** hai jo kabhi expand hogi aur kabhi shrink — jismein hamesha dekhna hai ki **target poora ho raha hai ya nahi**.

🧩 Step-by-Step Breakdown:

1. **Initialize Pointers:**
   - `i` (start of window)
   - `j` (end of window)
   - `sum` (current sum of window)

- `minlen` (answer, initially set to infinity)

2. **Expand the Window ➡️**
   - Move `j` forward and keep adding `nums[j]` to `sum`.
3. **Shrink the Window ⬅️**
   - Jab `sum >= target` ho jaye, toh:
     - Length calculate karo → `j - i + 1`
     - Compare with `minlen` → Update if smaller
     - Shrink from left → `sum -= nums[i]` and `i++`
4. **Repeat** till `j` reaches end.
5. **Final Check:**
   - If `minlen` was never updated → return `0`
   - Else → return `minlen`

👨‍💻 Code with ❤️ by Priyansh:

```
class Solution {
public:
    int minSubArrayLen(int target, vector<int>& nums) {
        int n = nums.size();
        int j = 0;  // End of window
        int i = 0;  // Start of window
        int sum = 0;
        int minlen = INT_MAX;

        while (j < n) {
            sum += nums[j];  // Expand window

            // Shrink while condition is satisfied
            while (sum >= target) {
                int len = j - i + 1;
```

```
                minlen = min(minlen, len);
                sum -= nums[i];
                i++;  // Shrink window
            }
            j++;  // Move window forward
        }

        // No valid subarray found
        if (minlen == INT_MAX) return 0;

        return minlen;
    }
};
```

💡 Time & Space Complexity:

Type	Complexity
⏱️ Time	O(n)
🧠 Space	O(1) (in-place)

📌 Notes by Priyansh:

✅ Ye technique kaafi powerful hai jab bhi **contiguous subarray** ke sath **sum / product / max / min** jaisa kuch kaam ho.

✅ Always think of **sliding window** when dealing with "minimum length", "longest subarray", or "fixed sum".

📎 Real-World Analogy:

Imagine a scanner (window) moving over a document (array). You're trying to find the **smallest possible area** (subarray) where the scanner covers words (elements) that **meet your target score** (sum). As soon as you get that score, you try to **shrink** the scanned area — because we're searching for the **smallest** one. 📄🔍

⬅️ END Final Thoughts:

> "Window ko tab tak bada karo jab tak target na mil jaye,
>
> milne ke baad window ko chhota karo jab tak target bana rahe."
>
> – *Priyansh ke Algorithms* 😎
