Min size subarray

Let's explain the code line by line in the simplest way so students can follow.

Code

python Copy Edit

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

        for right in range(n):
            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
```

Step-by-step Explanation

1. `n = len(nums)`
   - Get how many numbers are in the array.
   - We will loop from `0` to `n-1` for the right pointer.
2. `left = 0`
   - This is the start of the sliding window.
   - The window will be `nums[left ... right]`.
3. `curr_sum = 0`
   - Keeps track of the sum of the current window.
   - When we expand the window, we add new numbers to this sum.
4. `min_len = float('inf')`
   - We want the minimum length of a valid window.
   - Start with infinity, meaning "not found yet".
   - Whenever we find a valid window, we update `min_len` with the smaller value.

5. `for right in range(n):`
   - The `right` pointer expands the window step by step.
   - At each step, we include `nums[right]` into the window.
6. `curr_sum += nums[right]`
   - Add the new number into the current window sum.
   - Now the window is `nums[left ... right]`.

---

7. `while curr_sum >= target:`
   - If the sum of the current window meets or exceeds the target, it is valid.
   - But maybe we can make it smaller by shrinking from the left.
   - So, while it's still valid, we keep shrinking.

---

8. `min_len = min(min_len, right - left + 1)`
   - Check if the current window length `(right - left + 1)` is the smallest so far.
   - Update `min_len` if this window is shorter.

---

9. `curr_sum -= nums[left]`
   - Shrink the window from the left by removing `nums[left]` from the sum.
10. `left += 1`
    - Move the left pointer one step right to actually shrink the window.
    - After this, the new window is smaller.
    - Loop continues to check if it is still $\geqslant$ target.

---

11. `return 0 if min_len == float('inf') else min_len`
    - If `min_len` never changed, it means no valid subarray was found → return 0.
    - Otherwise, return the smallest length found.

Concept for Students

- We maintain a window `[left...right]` whose sum we keep updating.
- Expand with `right` → add more numbers.
- If sum is big enough, shrink with `left` to find the smallest valid window.
- Keep track of the minimum length during the process.

This is O(n) because each element is added once (by right) and removed at most once (by left).

Would you like me to also give a step-by-step run with a concrete example (`target=7, nums=[2,3,1,2,4,3]`), showing how each line executes and how variables change at each step?