**209. Minimum Size Subarray Sum**

Medium

130378FavoriteShare

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

**Example:**

**Input:** s = 7, nums = [2,3,1,2,4,3]

**Output:** 2

**Explanation:** the subarray [4,3] has the minimal length under the problem constraint.

**Follow up:**

If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*).

**Algorithm**

* Initialize \text{left}left pointer to 0 and \text{sum}sum to 0
* Iterate over the \text{nums}nums:
  + Add \text{nums}[i]nums[*i*] to \text{sum}sum
  + While \text{sum}sum is greater than or equal to s*s*:
    - Update \text{ans}=\min(\text{ans},i+1-\text{left})ans=min(ans,*i*+1−left), where i+1-\text{left}*i*+1−left is the size of current subarray
    - It means that the first index can safely be incremented, since, the minimum subarray starting with this index with \text{sum} \geq ssum≥*s*has been achieved
    - Subtract \text{nums[left]}nums[left] from \text{sum}sum and increment \text{left}left

int minSubArrayLen(int s, vector<int>& nums)

{

int n = nums.size();

int ans = INT\_MAX;

int left = 0;

int sum = 0;

for (int i = 0; i < n; i++) {

sum += nums[i];

while (sum >= s) {

ans = min(ans, i + 1 - left);

sum -= nums[left++];

}

}

return (ans != INT\_MAX) ? ans : 0;

}