1. Minimum Size Subarray Sum

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*).

**解法1** 暴力搜索

找出所有的$\_{k=i}^j a\_k s j-i+1$最小的

class Solution {  
public:  
 int minSubArrayLen(int s, vector<int>& nums) {  
 if(nums.size() == 0)return 0;  
 int ans = INT\_MAX;  
 for(int i = 0; i < nums.size(); ++i){  
 int tmp\_sum = 0;  
 for(int j = i; j < nums.size() && j - i <= ans; ++j){  
 tmp\_sum += nums[j];  
 if(tmp\_sum >= s)ans = min(ans, j-i+1);  
 }  
 }  
 return ans == INT\_MAX ? 0 : ans;  
 }  
};

Note :

* 在内层循环中，一定要加j - i <= ans的判断条件，否则会超时
* 为了避免在内层循环中重复求和，可以先计算nums的前n项和，放到数组sum中

**解法2** 二分查找。前n项和数组sum一定是单调递增的，原问题可以转换为：

查找在sum中第一次出现的位置，即找对应的最小长度

直接调用c++ stl中的lower\_bound()函数

class Solution {  
public:  
 int minSubArrayLen(int s, vector<int>& nums) {  
 if(nums.size() == 0)return 0;  
 vector<int>sum(nums.size() + 1, 0);  
 for(int i = 0; i < nums.size(); ++i)sum[i+1] = sum[i]+nums[i];  
 int ans = INT\_MAX;  
 for(int i = 1; i <= nums.size(); ++i){  
 int to\_find = s + sum[i-1];  
 auto bound = lower\_bound(sum.begin(), sum.end(), to\_find);  
 if(bound != sum.end())ans = min(ans, int(bound - sum.begin()) - i + 1);  
 }  
 return ans == INT\_MAX ? 0 : ans;  
 }  
};

**解法3** one-pass。记录满足的子串的起始位置，在找到一个符合条件的子串后，不断收缩子串

class Solution {  
public:  
 int minSubArrayLen(int s, vector<int>& nums) {  
 if(nums.size() == 0)return 0;  
 int ans = INT\_MAX, left = 0, sum = 0;  
 for(int i = 0; i < nums.size(); ++i){  
 sum += nums[i];  
 while(sum >= s){  
 ans = min(ans, i - left + 1);  
 sum -= nums[left++];  
 }  
 }  
 return ans == INT\_MAX ? 0 : ans;  
 }  
};