# 209. Minimum Size Subarray Sum

Given an array of positive integers `nums` and a positive integer `target`, return *the* ***minimal length*** *of a subarray whose sum is greater than or equal to* `target`. If there is no such subarray, return `0` instead.

**Example 1:**

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

**Example 2:**

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

**Example 3:**

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

**Constraints:**

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

**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))`.

# 209. Minimum Size Subarray Sum

```
/*
        Sliding window
        Time compelxity: O(n)
        Space complexity: O(1)
*/
class Solution {
public:
    int minSubArrayLen(int target, vector<int>& nums) {
        int n=nums.size();
        int l=0,r=0,sum=0,ans=INT_MAX;
        while(r<n){
            sum+=nums[r];
            while(sum>=target){
                sum-=nums[l];
                ans=std::min(ans,r-l+1);
                l++;
            }
            r++;
        }
        return ans==INT_MAX?0:ans;
    }
};
```

# 862. Shortest Subarray with Sum at Least K

Given an integer array `nums` and an integer `k`, return *the length of the shortest non-empty **subarray** of* `nums` *with a sum of at least* `k`. If there is no such **subarray**, return `-1`.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

```
Input: nums = [1], k = 1
Output: 1
```

**Example 2:**

```
Input: nums = [1,2], k = 4
Output: -1
```

**Example 3:**

```
Input: nums = [2,-1,2], k = 3
Output: 3
```

**Constraints:**

- $1 <= nums.length <= 10^5$
- $-10^5 <= nums[i] <= 10^5$
- $1 <= k <= 10^9$

# 862. Shortest Subarray with Sum at Least K

```
/*
    Brute force
    Time complexity: O(n^2)
    Space complexity: O(1)
*/
class Solution {
    public:
        int shortestSubarray(std::vector<int>& nums, int k) {
            int n=nums.size();

            int ans=INT_MAX;

            for (int i=0;i<n;++i){
                long long s=0;
                int j=i;
                while(j<n && s<k){
                    s+=nums[j];
                    j++;
                }
                if(s>=k) ans=std::min(ans,j-i);
            }
            return (ans != INT_MAX)?ans:-1;
        }
};
```

# 862. Shortest Subarray with Sum at Least K

```
/*
  Min heap
  Time complexity: O(nlog n)
  Space complexity: O(n)
*/
typedef std::pair<long long,int> ii;
typedef std::vector<ii> vii;

class Solution {
  public:
    int shortestSubarray(std::vector<int>& nums, int k) {
      int n=nums.size();

      std::priority_queue<ii,vii,std::greater<ii>> min_heap;

      int ans=INT_MAX;
      long long s=0;
      for (int i=0;i<n;++i){
        s+=nums[i];
        // If s>=k, minimize the actual length with previous lengths
        if(s>=k) ans=std::min(ans,i+1);
        // Always take the minimum prefix sum before i, trying to hold ps[i]-ps[x]>=k
        while(!min_heap.empty() && s-min_heap.top().first>=k){
          ans=std::min(ans,i-min_heap.top().second);
          min_heap.pop(); // Not need to use this index for incoming indexes
        }

        min_heap.push({s,i});
      }
      return (ans != INT_MAX)?ans:-1;
    }
};
```

# 862. Shortest Subarray with Sum at Least K

```
/*
  Monotonic deque
  Time complexity: O(n)
  Space complexity: O(n)
*/
typedef std::pair<long long,int> ii;
typedef std::vector<ii> vii;

class Solution {
  public:
    int shortestSubarray(std::vector<int>& nums, int k) {
      int n=nums.size();

      // Create a deque
      std::deque<ii> q;

      int ans=INT_MAX;
      long long s=0;
      for (int i=0;i<n;++i){
       s+=nums[i];
       // If s>=k, minimize the actual length with previous lengths
       if(s>=k) ans=std::min(ans,i+1);

       // Always take the minimum prefix sum before i, trying to hold ps[i]-ps[x]>=k
       while(!q.empty() && s-q.front().first>=k){
         ans=std::min(ans,i-q.front().second);
         q.pop_front();
       }

       // Remove any prefix sum > s, to ensure that the deque stay
       // monotonically increasing
       while(!q.empty() && q.back().first>=s) q.pop_back();

       // Here, we are sure that s is the greatest value
       q.push_back({s,i});
      }

      return (ans != INT_MAX)?ans:-1;
    }
};
```