# 题目

给定一个 长度为 n 的整型数组 nums 和一个数值 k ，返回第 k 小的子序列和。

一个 子序列 是指 非空 且不间断的子数组。  子序列和 则指子序列的和。

示例 1:

输入: nums = [2,1,3], k = 4

输出: 3

解释: [2,1,3] 的子序列为：

- [2] 和为 2

- [1] 和为 1

- [3] 和为 3

- [2,1] 和为 3

- [1,3] 和为 4

- [2,1,3] 和为 6

最小子序列和的升序排序为 1, 2, 3, <u>3</u>, 4, 6。 第4小的子序列和为 3。

Example 2:

Input: nums = [3,3,5,5], k = 7

Output: 10

Explanation: The subarrays of [3,3,5,5] are:

- [3] 和为 3

- [3] 和为 3

- [5] 和为 5

- [5] 和为 5

- [3,3] 和为 6

- [3,5] 和为 8

- [5,5] 和为 10

- [3,3,5], 和为 11

- [3,5,5] 和为 13

- [3,3,5,5] 和为 16

最小子序列和的升序排序为 3, 3, 5, 5, 6, 8, <u>10</u>, 11, 13, 16。第4小的子序列和为 10。

提示:

n == nums.length

1 <= n <= 2 \* 104

1 <= nums[i] <= 5 \* 104

1 <= k <= n \* (n + 1) / 2

# 分析

## 方法一：二分查找 + 滑动窗口

思路和算法

代码

class Solution {

public int kthSmallestSubarraySum(int[] nums, int k) {

int min = Integer.MAX\_VALUE, sum = 0;

for (int num : nums) {

min = Math.min(min, num);

sum += num;

}

int low = min, high = sum;

while (low < high) {

int mid = (high - low) / 2 + low;

int count = countSubarrays(nums, mid);

if (count < k) {

low = mid + 1;

} else {

high = mid;

}

}

return low;

}

public int countSubarrays(int[] nums, int threshold) {

int count = 0;

int sum = 0;

int n = nums.length;

int left = 0, right = 0;

while (right < n) {

sum += nums[right];

while (sum > threshold) {

sum -= nums[left];

left++;

}

count += right - left + 1;

right++;

}

return count;

}

}

复杂度分析

时间复杂度：O(nlogS)，其中n为数组nums的长度，S为数组nums的元素和。二分查找的次数是 O(logS) 次，每次二分查找需要O(n) 的时间计算元素和小于或等于特定阈值的子数组的数量。

空间复杂度：)O(1)。