# 题目

给你一个整数数组 nums 和一个整数 x 。每一次操作时，你应当移除数组 nums 最左边或最右边的元素，然后从 x 中减去该元素的值。请注意，需要 修改 数组以供接下来的操作使用。

如果可以将 x 恰好 减到 0 ，返回 最小操作数 ；否则，返回 -1 。

示例 1：

输入：nums = [1,1,4,2,3], x = 5

输出：2

解释：最佳解决方案是移除后两个元素，将 x 减到 0 。

示例 2：

输入：nums = [5,6,7,8,9], x = 4

输出：-1

示例 3：

输入：nums = [3,2,20,1,1,3], x = 10

输出：5

解释：最佳解决方案是移除后三个元素和前两个元素（总共 5 次操作），将 x 减到 0 。

提示：

1 <= nums.length <= 105

1 <= nums[i] <= 104

1 <= x <= 109

# 分析

要解决从数组两端移除元素使x恰好减到0的问题，我们可以转换思路，通过寻找数组中最长的连续子数组（中间部分），其和等于数组总和减去x，从而间接求出最小操作数。

解题思路：

1、问题转换：

- 设数组总和为total，目标是从两端移除元素的和为x，等价于数组中间存在一个连续子数组，其和为total - x。

- 若total < x，则不可能完成，直接返回-1。

- 若total == x，则需移除所有元素，返回数组长度。

2、滑动窗口找最长子数组：

- 用滑动窗口在数组中寻找和为target = total - x的最长连续子数组。

- 窗口的左右指针初始都在数组头部，通过移动右指针扩大窗口，当窗口和超过`target`时，移动左指针缩小窗口。

- 记录窗口和等于target时的最大长度。

3、计算最小操作数：

- 若找到最长子数组的长度为max\_len，则最小操作数为数组长度 - max\_len。

- 若未找到这样的子数组，返回-1。

代码：

class Solution {

public:

int minOperations(vector<int>& nums, int x) {

int total = accumulate(nums.begin(), nums.end(), 0);

int target = total - x;

int n = nums.size();

// 总和小于x，不可能完成

if (target < 0) return -1;

// 总和等于x，需移除所有元素

if (target == 0) return n;

int left = 0;

int current\_sum = 0;

int max\_len = -1;

// 滑动窗口寻找和为target的最长子数组

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

current\_sum += nums[right];

// 窗口和超过target，缩小左边界

while (current\_sum > target && left <= right) {

current\_sum -= nums[left];

left++;

}

// 找到和为target的子数组，更新最大长度

if (current\_sum == target) {

max\_len = max(max\_len, right - left + 1);

}

}

// 若找到有效子数组，计算最小操作数；否则返回-1

return max\_len == -1 ? -1 : n - max\_len;

}

};

代码解释：

1、总和与目标计算：先计算数组总和total，再得到目标子数组的和target = total - x。若target为负，直接返回-1；若target为0，返回数组长度（需移除所有元素）。

2、滑动窗口逻辑：

- 右指针right遍历数组，将元素加入当前窗口和current\_sum。

- 若current\_sum超过target，移动左指针left并从current\_sum中减去对应元素，直到current\_sum不超过target。

- 当current\_sum等于target时，更新最长子数组长度max\_len。

3、结果计算：若存在有效子数组，最小操作数为数组长度减去最长子数组长度（即两端需要移除的元素数）；否则返回-1。

复杂度分析

- 时间复杂度：O(n)，其中n是数组长度。滑动窗口的左右指针各遍历数组一次，每个元素最多被访问两次。

- 空间复杂度：O(1)，仅使用常数个变量存储中间结果，无需额外空间。