# 题目

有一个整数数组 nums ，和一个查询数组 requests ，其中 requests[i] = [starti, endi] 。第 i 个查询求 nums[starti] + nums[starti + 1] + ... + nums[endi - 1] + nums[endi] 的结果 ，starti 和 endi 数组索引都是 从 0 开始 的。

你可以任意排列 nums 中的数字，请你返回所有查询结果之和的最大值。

由于答案可能会很大，请你将它对 109 + 7 取余 后返回。

示例 1：

输入：nums = [1,2,3,4,5], requests = [[1,3],[0,1]]

输出：19

解释：一个可行的 nums 排列为 [2,1,3,4,5]，并有如下结果：

requests[0] -> nums[1] + nums[2] + nums[3] = 1 + 3 + 4 = 8

requests[1] -> nums[0] + nums[1] = 2 + 1 = 3

总和为：8 + 3 = 11。

一个总和更大的排列为 [3,5,4,2,1]，并有如下结果：

requests[0] -> nums[1] + nums[2] + nums[3] = 5 + 4 + 2 = 11

requests[1] -> nums[0] + nums[1] = 3 + 5 = 8

总和为： 11 + 8 = 19，这个方案是所有排列中查询之和最大的结果。

示例 2：

输入：nums = [1,2,3,4,5,6], requests = [[0,1]]

输出：11

解释：一个总和最大的排列为 [6,5,4,3,2,1] ，查询和为 [11]。

示例 3：

输入：nums = [1,2,3,4,5,10], requests = [[0,2],[1,3],[1,1]]

输出：47

解释：一个和最大的排列为 [4,10,5,3,2,1] ，查询结果分别为 [19,18,10]。

提示：

n == nums.length

1 <= n <= 105

0 <= nums[i] <= 105

1 <= requests.length <= 105

requests[i].length == 2

0 <= starti <= endi < n

# 分析

要解决“通过排列数组元素最大化查询结果总和”的问题，核心思路是统计每个索引被查询覆盖的次数，再将最大的数组元素分配给被覆盖次数最多的索引，从而最大化总和。

解题思路

1、核心观察：

每个查询[start, end]会累加区间内所有索引的元素值。若某索引被k个查询覆盖，则该索引的元素值会被计入总和k次。因此，要最大化总和，需将数组中最大的元素分配给被覆盖次数最多的索引，次大元素分配给次多覆盖次数的索引，以此类推（贪心策略）。

2、步骤拆解：

- 统计每个索引的覆盖次数：利用差分数组高效计算每个索引被查询区间覆盖的次数（避免O(n×q)的暴力统计，n为数组长度，q为查询数）。

- 排序与匹配：将数组元素从大到小排序，将覆盖次数从大到小排序，然后按顺序相乘并累加（最大元素×最大次数 + 次大元素×次大次数 + ...），得到最大总和。

代码：

class Solution {

public:

const int MOD = 1e9 + 7;

int maxSumRangeQuery(vector<int>& nums, vector<vector<int>>& requests) {

int n = nums.size();

// 差分数组：用于高效统计每个索引的覆盖次数

vector<int> diff(n + 1, 0);

// 处理所有查询，更新差分数组

for (auto& req : requests) {

int start = req[0];

int end = req[1];

diff[start]++; // 区间[start, ...]覆盖次数+1

if (end + 1 < n) {

diff[end + 1]--; // 区间[end+1, ...]覆盖次数-1（若未越界）

}

}

// 计算前缀和，得到每个索引的实际覆盖次数

vector<int> counts(n);

counts[0] = diff[0];

for (int i = 1; i < n; ++i) {

counts[i] = counts[i - 1] + diff[i];

}

// 排序：数组元素从大到小，覆盖次数从大到小

sort(nums.rbegin(), nums.rend());

sort(counts.rbegin(), counts.rend());

// 计算最大总和：大元素匹配大次数

long long sum = 0;

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

sum = (sum + (long long)nums[i] \* counts[i]) % MOD;

}

return (int)sum;

}

};

代码解释

1、差分数组统计覆盖次数：

- 差分数组diff的作用是高效记录区间更新：对查询[start, end]，执行diff[start]++和diff[end+1]--（若end+1 < n）。

- 通过计算diff的前缀和得到counts数组，counts[i]即为索引i被查询覆盖的总次数。此过程时间复杂度为O(n + q)（q为查询数），远优于暴力统计的O(n×q)。

2、排序与匹配：

- 将nums从大到小排序（最大元素优先），将counts从大到小排序（覆盖次数最多的索引优先）。

- 按顺序相乘并累加，确保最大的元素被最多次数的查询计入，从而最大化总和。

3、取模处理：

由于结果可能很大，每次累加后对10^9 + 7取模，避免溢出。

复杂度分析

- 时间复杂度：O(n log n + q)，其中n是数组长度，q是查询数。

差分数组更新：O(q)。

前缀和计算：O(n)。

排序nums和counts：O(n log n)（主导复杂度）。

- 空间复杂度：O(n)，用于存储差分数组和覆盖次数数组。