# 题目

两个数对 (a, b) 和 (c, d) 之间的 乘积差 定义为 (a \* b) - (c \* d) 。

例如，(5, 6) 和 (2, 7) 之间的乘积差是 (5 \* 6) - (2 \* 7) = 16 。

给你一个整数数组 nums ，选出四个 不同的 下标 w、x、y 和 z ，使数对 (nums[w], nums[x]) 和 (nums[y], nums[z]) 之间的乘积差取到最大值。

返回以这种方式取得的乘积差中的最大值。

示例 1：

输入：nums = [5,6,2,7,4]

输出：34

解释：可以选出下标为 1 和 3 的元素构成第一个数对 (6, 7) 以及下标 2 和 4 构成第二个数对 (2, 4)

乘积差是 (6 \* 7) - (2 \* 4) = 34

示例 2：

输入：nums = [4,2,5,9,7,4,8]

输出：64

解释：可以选出下标为 3 和 6 的元素构成第一个数对 (9, 8) 以及下标 1 和 5 构成第二个数对 (2, 4)

乘积差是 (9 \* 8) - (2 \* 4) = 64

提示：

4 <= nums.length <= 10^4

1 <= nums[i] <= 10^4

# 分析

## 方法一：排序

思路：

代码：

class Solution {

public:

int maxProductDifference(vector<int>& nums) {

// 对数组进行排序

sort(nums.begin(), nums.end());

// 获取最大和最小的两个数的乘积差

int maxProduct = nums.back() \* nums[nums.size() - 2];

int minProduct = nums[0] \* nums[1];

// 返回乘积差

return maxProduct - minProduct;

}

};

上述算法的时间复杂度是O(n log n)，空间复杂度在最坏情况下是O(n)，但可能通过优化达到O(log n)或O(1)，这取决于std::sort的具体实现和所使用的排序算法。在实际应用中，我们通常假设std::sort的空间复杂度是O(log n)或O(n)，除非有特别说明或优化。