# 题目

给你一个按 非递减顺序 排列的数组 nums ，返回正整数数目和负整数数目中的最大值。

换句话讲，如果 nums 中正整数的数目是 pos ，而负整数的数目是 neg ，返回 pos 和 neg二者中的最大值。

注意：0 既不是正整数也不是负整数。

示例 1：

输入：nums = [-2,-1,-1,1,2,3]

输出：3

解释：共有 3 个正整数和 3 个负整数。计数得到的最大值是 3 。

示例 2：

输入：nums = [-3,-2,-1,0,0,1,2]

输出：3

解释：共有 2 个正整数和 3 个负整数。计数得到的最大值是 3 。

示例 3：

输入：nums = [5,20,66,1314]

输出：4

解释：共有 4 个正整数和 0 个负整数。计数得到的最大值是 4 。

提示：

1 <= nums.length <= 2000

-2000 <= nums[i] <= 2000

nums 按非递减顺序排列。

进阶：你可以设计并实现时间复杂度为 O(log(n)) 的算法解决此问题吗？

# 分析

## 方法一：遍历

遍历数组，用两个变量统计。

代码：

class Solution {

public:

int maximumCount(vector<int> &nums) {

int less = 0, great = 0;

for (int x : nums) {

if (x < 0) {

less++;

} else if (x > 0) {

great++;

}

}

return max(less, great);

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n为nums的长度。

空间复杂度：O(1)，仅用到若干额外变量。

## 方法二：二分查找

class Solution {

public:

int maximumCount(vector<int> &nums) {

int less = lower\_bound(nums.begin(), nums.end(), 0) - nums.begin();

int great = nums.end() - upper\_bound(nums.begin(), nums.end(), 0);

return max(less, great);

}

};

**复杂度分析：**

时间复杂度：O(logn)，其中n为nums的长度。

空间复杂度：O(1)，仅用到若干额外变量。