# 题目

给你一个下标从 0 开始的整数数组 nums 和一个整数 pivot 。请你将 nums 重新排列，使得以下条件均成立：

* 所有小于 pivot 的元素都出现在所有大于 pivot 的元素 之前 。
* 所有等于 pivot 的元素都出现在小于和大于 pivot 的元素 中间 。
* 小于 pivot 的元素之间和大于 pivot 的元素之间的 相对顺序 不发生改变。

更正式的，考虑每一对 pi，pj ，pi 是初始时位置 i 元素的新位置，pj 是初始时位置 j 元素的新位置。对于小于 pivot 的元素，如果 i < j 且 nums[i] < pivot 和 nums[j] < pivot 都成立，那么 pi < pj 也成立。类似的，对于大于 pivot 的元素，如果 i < j 且 nums[i] > pivot 和 nums[j] > pivot 都成立，那么 pi < pj 。

请你返回重新排列 nums 数组后的结果数组。

示例 1：

输入：nums = [9,12,5,10,14,3,10], pivot = 10

输出：[9,5,3,10,10,12,14]

解释：

元素 9 ，5 和 3 小于 pivot ，所以它们在数组的最左边。

元素 12 和 14 大于 pivot ，所以它们在数组的最右边。

小于 pivot 的元素的相对位置和大于 pivot 的元素的相对位置分别为 [9, 5, 3] 和 [12, 14] ，它们在结果数组中的相对顺序需要保留。

示例 2：

输入：nums = [-3,4,3,2], pivot = 2

输出：[-3,2,4,3]

解释：

元素 -3 小于 pivot ，所以在数组的最左边。

元素 4 和 3 大于 pivot ，所以它们在数组的最右边。

小于 pivot 的元素的相对位置和大于 pivot 的元素的相对位置分别为 [-3] 和 [4, 3] ，它们在结果数组中的相对顺序需要保留。

提示：

1 <= nums.length <= 10^5

-10^6 <= nums[i] <= 10^6

pivot 等于 nums 中的一个元素。

# 分析

## 方法一：双指针

思路：

我们可以对数组nums进行一次遍历，并使用两个指针left和right来更新答案数组。答案数组长度与nums的长度相同，初始时，left和right分别指向答案数组的最左端和最右端。

在对nums进行遍历时，记当前遍历到的数为 x。如果x<pivot，我们就将x放入左指针的位置，并将左指针向右移动一个位置；如果x>pivot，我们就将x放入右指针的位置，并将右指针向左移动一个位置。

在遍历结束之后，我们需要注意两点：

对于所有与pivot相等的元素，我们并没有进行处理。我们可以在初始化答案数组时，直接将所有元素赋值为pivot来避免额外的处理；

对于严格大于pivot的元素，题目中要求它们的「相对顺序」不能发生改变，因此需要将这些元素在答案数组中对应的段进行反转。

代码：

class Solution {

public:

vector<int> pivotArray(vector<int>& nums, int pivot) {

int n = nums.size();

vector<int> ans(n, pivot);

int left = 0, right = n - 1;

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

if (nums[i] < pivot) {

ans[left] = nums[i];

++left;

}

else if (nums[i] > pivot) {

ans[right] = nums[i];

--right;

}

}

reverse(ans.begin() + right + 1, ans.end());

return ans;

}

};

复杂度分析

时间复杂度：O(n)，其中n是数组nums的长度。

空间复杂度：O(1)。这里不考虑答案数组需要使用的空间。