# 题目

给你两个非递增的整数数组nums1和nums2，数组下标均 从0开始计数。

下标对(i, j)中0 <= i < nums1.length 且 0 <= j < nums2.length 。如果该下标对同时满足i <= j且nums1[i] <= nums2[j]，则称之为有效下标对，该下标对的 距离为j – i。

返回所有有效下标对(i, j)中的最大距离。如果不存在有效下标对，返回0。

一个数组arr，如果每个1 <= i < arr.length均有arr[i-1] >= arr[i]成立，那么该数组是一个非递增数组。

示例 1：

输入：nums1 = [55,30,5,4,2], nums2 = [100,20,10,10,5]

输出：2

解释：有效下标对是 (0,0), (2,2), (2,3), (2,4), (3,3), (3,4) 和 (4,4) 。

最大距离是 2 ，对应下标对 (2,4) 。

示例 2：

输入：nums1 = [2,2,2], nums2 = [10,10,1]

输出：1

解释：有效下标对是 (0,0), (0,1) 和 (1,1) 。

最大距离是 1 ，对应下标对 (0,1) 。

示例 3：

输入：nums1 = [30,29,19,5], nums2 = [25,25,25,25,25]

输出：2

解释：有效下标对是(2,2), (2,3), (2,4), (3,3)和(3,4)。

最大距离是2，对应下标对(2,4)。

提示：

1 <= nums1.length <= 105

1 <= nums2.length <= 105

1 <= nums1[i], nums2[j] <= 105

nums1和nums2都是非递增数组

# 分析

## 方法一：双指针

思路：

代码：

class Solution {

public:

int maxDistance(vector<int>& nums1, vector<int>& nums2) {

int n1 = nums1.size();

int n2 = nums2.size();

int res = 0;

int j = 0;

for (int i = 0; i < n1; i++) {

// 因为是非递增的，因此这里不需要再从头开始遍历nums2

while (j < n2 && nums1[i] <= nums2[j]) {

res = max(res, j - i);

j++;

}

}

return res;

}

};

**复杂度分析：**

时间复杂度：O(n1+n2)，其中n1,n2分别为nums1与nums2的长度。在双指针寻找最大值的过程中，我们最多会遍历两个数组各一次。

空间复杂度：O(1)，我们使用了常数个变量进行遍历。

## 方法二：二分查找