# 题目

给定一个无序的数组nums，返回数组在排序之后，相邻元素之间最大的差值。如果数组元素个数小于2，则返回0。

您必须编写一个在「线性时间」内运行并使用「线性额外空间」的算法。

示例 1:

输入: nums = [3,6,9,1]

输出: 3

解释: 排序后的数组是 [1,3,6,9], 其中相邻元素 (3,6) 和 (6,9) 之间都存在最大差值 3。

示例 2:

输入: nums = [10]

输出: 0

解释: 数组元素个数小于 2，因此返回 0。

提示:

1 <= nums.length <= 10^5

0 <= nums[i] <= 10^9

# 分析

## 方法一：基数排序

思路：

一种最简单的思路是将数组排序后再找出最大间距，但传统的基于比较的排序算法（快速排序、归并排序等）均需要O(NlogN)的时间复杂度。如果要将时间复杂度降到O(N)，我们就必须使用其他的排序算法。例如，基数排序可以在O(N)的时间内完成整数之间的排序。

代码：

## 方法二：桶排序

思路：

这里要求的是在线性时间内完成，因此可以选择桶排序。

首先找到数组中的最小值和最大值，然后计算桶的大小和桶的数量。接着，创建桶并将元素放入对应的桶中，维护每个桶的最小值和最大值。最后，遍历桶，计算相邻桶之间的最大差值。这个算法保证了线性时间和线性额外空间的要求。

class Solution {

public:

int maximumGap(vector<int>& nums) {

if (nums.size() < 2) {

return 0;

}

// 找到数组中的最小值和最大值

int minVal = \*min\_element(nums.begin(), nums.end());

int maxVal = \*max\_element(nums.begin(), nums.end());

// 计算桶的大小和桶的数量

int bucketSize = std::max(1, (maxVal - minVal) / std::max(int(nums.size() - 1), 1));

int bucketCount = (maxVal - minVal) / bucketSize + 1;

// 创建桶，每个桶维护最小值和最大值

vector<pair<int, int>> buckets(bucketCount, {INT\_MAX, INT\_MIN});

// 将元素放入桶中，并维护桶的最小值和最大值

for (int num : nums) {

int index = (num - minVal) / bucketSize;

buckets[index].first = std::min(buckets[index].first, num);

buckets[index].second = std::max(buckets[index].second, num);

}

// 计算相邻桶之间的最大差值

int maxGap = 0;

int prevMax = minVal;

for (const auto& bucket : buckets) {

// 跳过空桶

if (bucket.first == INT\_MAX) {

continue;

}

maxGap = std::max(maxGap, bucket.first - prevMax);

prevMax = bucket.second;

}

return maxGap;

}

};