# 题目

给定两个整数数组，请交换一对数值（每个数组中取一个数值），使得两个数组所有元素的和相等。

返回一个数组，第一个元素是第一个数组中要交换的元素，第二个元素是第二个数组中要交换的元素。若有多个答案，返回任意一个均可。若无满足条件的数值，返回空数组。

示例:

输入: array1 = [4, 1, 2, 1, 1, 2], array2 = [3, 6, 3, 3]

输出: [1, 3]

示例:

输入: array1 = [1, 2, 3], array2 = [4, 5, 6]

输出: []

提示：

1 <= array1.length, array2.length <= 100000

# 分析

## 方法一：哈希表

思路：

对于这个问题，我们可以先计算两个数组的和，然后计算它们之间的差值 diff。如果diff是奇数，那么两个数组无法交换元素使得和相等，直接返回空数组。如果diff是偶数，则说明可以找到一对数进行交换。

我们可以在一个数组中查找是否存在一个数，使得它与另一个数组中的某个数的差值等于diff / 2。这样一对数的交换就可以使得两个数组的和相等。

具体算法如下：

1. 计算数组array1和array2的和分别为sum1和sum2。

2. 计算差值diff = sum2 - sum1。

3. 如果diff是奇数，返回空数组[]。

4. 遍历数组array1，对于每个元素num1，计算目标值target = num1 + diff / 2。

5. 在数组array2中查找是否存在值等于target的元素，如果存在，返回 [num1, target]。

6. 如果遍历结束仍未找到符合条件的交换元素，则返回空数组[]。

代码：

class Solution {

public:

vector<int> findSwapValues(vector<int>& array1, vector<int>& array2) {

int sum1 = 0, sum2 = 0;

unordered\_set<int> set2;

for (int num : array1) {

sum1 += num;

}

for (int num : array2) {

sum2 += num;

set2.insert(num);

}

int diff = sum2 - sum1;

if (diff % 2 != 0) {

return {};

}

int target = diff / 2;

for (int num1 : array1) {

int num2 = num1 + target;

if (set2.count(num2) > 0) {

return {num1, num2};

}

}

return {};

}

};

## 方法二：排序

思路：

为了解决这个问题，我们可以首先计算两个数组的总和，并检查它们的差是否可以被2整除。如果不能，那么无论如何交换都不可能使得两个数组的和相等，因此直接返回空数组。

如果差可以被2整除，那么我们需要找到两个数组中的一个元素，使得从第一个数组中减去这个元素并从第二个数组加上这个元素后，两个数组的和相等。

一个简单的方法是，首先对所有元素进行排序，然后从两端开始搜索可能的交换元素。如果两个数组的长度不同，那么我们应该从长度较长的那个数组开始搜索，因为它有更多的元素可供选择。

代码：

class Solution {

public:

std::vector<int> findSwapValues(std::vector<int>& array1, std::vector<int>& array2) {

std::vector<int> result;

int sum1 = 0, sum2 = 0;

// 计算两个数组的总和

for (int num : array1) sum1 += num;

for (int num : array2) sum2 += num;

// 如果总和之差不能被2整除，则无法交换使得和相等

if ((sum1 - sum2) % 2 != 0) return result;

int targetDiff = (sum1 - sum2) / 2;

// 对数组进行排序

std::sort(array1.begin(), array1.end());

std::sort(array2.begin(), array2.end());

int i = 0, j = 0;

int n1 = array1.size(), n2 = array2.size();

// 从两个数组的两端开始搜索

while (i < n1 && j < n2) {

int diff = array1[i] - array2[j];

if (diff == targetDiff) {

// 找到了一对满足条件的元素

result.push\_back(array1[i]);

result.push\_back(array2[j]);

return result;

} else if (diff < targetDiff) {

// 当前差值太小，增加array1的元素

i++;

} else {

// 当前差值太大，增加array2的元素

j++;

}

}

// 没有找到满足条件的元素对

return result;

}

};

请注意，这个算法的时间复杂度主要取决于排序操作，因此整体时间复杂度为O(n log n)，其中n是数组长度之和。如果两个数组的长度相差很大，并且较长的数组中有多个元素可以与较短的数组中的某个元素交换使得和相等，那么上面的算法可能不是最优的，因为它只找到了一对满足条件的元素。如果需要找到所有可能的交换对，那么算法需要相应地进行调整。然而，根据题目要求，返回任意一个满足条件的交换对即可，因此上述算法是符合要求的。