给定两个数组，写一个方法来计算它们的交集。

例如:

给定 nums1 = [1, 2, 2, 1], nums2 = [2, 2], 返回 [2, 2].

注意：

输出结果中每个元素出现的次数，应与元素在两个数组中出现的次数一致。

我们可以不考虑输出结果的顺序。

跟进:

如果给定的数组已经排好序呢？你将如何优化你的算法？

如果 nums1 的大小比 nums2 小很多，哪种方法更优？

如果nums2的元素存储在磁盘上，内存是有限的，你不能一次加载所有的元素到内存中，你该怎么办？

===========================================================

class Solution {

public:

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

vector<int> nums3;

for (int i = 0; i < nums1.size(); ++i)

for (int j = 0; j < nums2.size(); ++j)

if (nums1[i] == nums2[j])

{

nums3.push\_back(nums1[i]);

nums2.erase(nums2.begin() + j--); // 只删除了2号，导致1号继续 // 判断是否相等

}

return nums3;

}

};

27 / 61 个通过测试用例

状态：解答错误

输入：

[1]

[1,1]

输出：

[1,1]

预期：

[1]

===========================================================

class Solution {

public:

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

vector<int> nums3;

for (int i = 0; i < nums1.size(); ++i)

for (int j = 0; j < nums2.size(); ++j)

if (nums1[i] == nums2[j])

{

nums3.push\_back(nums1[i]);

nums1.erase(nums1.begin() + i--); // 取得交集后不再继续比较

nums2.erase(nums2.begin() + j--); // 删除已取得交集的元素以免重复获取

}

return nums3;

}

};

执行出错信息：

double free or corruption (out): 0x0000000001ba8eb0 \*\*\*

最后执行的输入：

[4,9,5]

[9,4,9,8,4]

错误信息

double free or corruption (out): 0x00000000011abe70 \*\*\*

问题分析

基本上根据题目判定，类型没得跑，内存问题。

所以会有几种情况：

1. 内存重复释放，看程序中是否释放了两次空间（一般不会是这种情况，毕竟。。太明显）

2. 内存越界。（大部分是这种情况，如果你使用了数组，或者开辟了空间，但是在循环的时候越界了，就会出现这种情况）

===========================================================

class Solution {

public:

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

vector<int> nums3;

for (int i = 0; i < nums1.size(); ++i)

for (int j = 0; j < nums2.size(); ++j)

if (nums1[i] == nums2[j])

{

nums3.push\_back(nums1[i]);

nums1.erase(nums1.begin() + i--); // 取得交集后不再继续比较

nums2.erase(nums2.begin() + j--); // 删除已取得交集的元素以免重复获取

break; // 跳出当前循环，直接开始下一轮循环

}

return nums3;

}

};

61 / 61 个通过测试用例

状态：通过

执行用时：12 ms

===========================================================

class Solution {

public:

/\*\*

\* @param nums1 an integer array

\* @param nums2 an integer array

\* @return an integer array

\*/

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

// Write your code here

/\* 1、sort函数可以三个参数也可以两个参数，必须的头文件#include < algorithm>和using namespace std;

2、它使用的排序方法是类似于快排的方法，时间复杂度为n\*log2(n)

3、Sort函数有三个参数：（第三个参数可不写）

（1）第一个是要排序的数组的起始地址。

（2）第二个是结束的地址（最后一位要排序的地址）

（3）第三个参数是排序的方法，可以是从大到小也可是从小到大，还可以不写第三个参数，此时默认的排序方法是从小到大排序。\*/

sort(nums1.begin(), nums1.end());

sort(nums2.begin(), nums2.end());

vector<int> intersect;

vector<int>::iterator it1 = nums1.begin(), it2 = nums2.begin();

while ((it1 != nums1.end()) && (it2 != nums2.end()))

{

// 小的+1，逼近大的

if (\*it1 < \*it2) it1++;

else if (\*it1 > \*it2) it2++;

else // 相等时

{

intersect.push\_back(\*it1);

it1++; it2++;

}

}

return intersect;

}

};