# Question

Given two arrays, write a function to compute their intersection.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2,2]

**Example 2:**

**Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4]

**Output:** [4,9]

**Note:**

* Each element in the result should appear as many times as it shows in both arrays.
* The result can be in any order.

**Follow up:**

* What if the given array is already sorted? How would you optimize your algorithm?
* What if *nums1*'s size is small compared to *nums2*'s size? Which algorithm is better?
* What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

# Solution

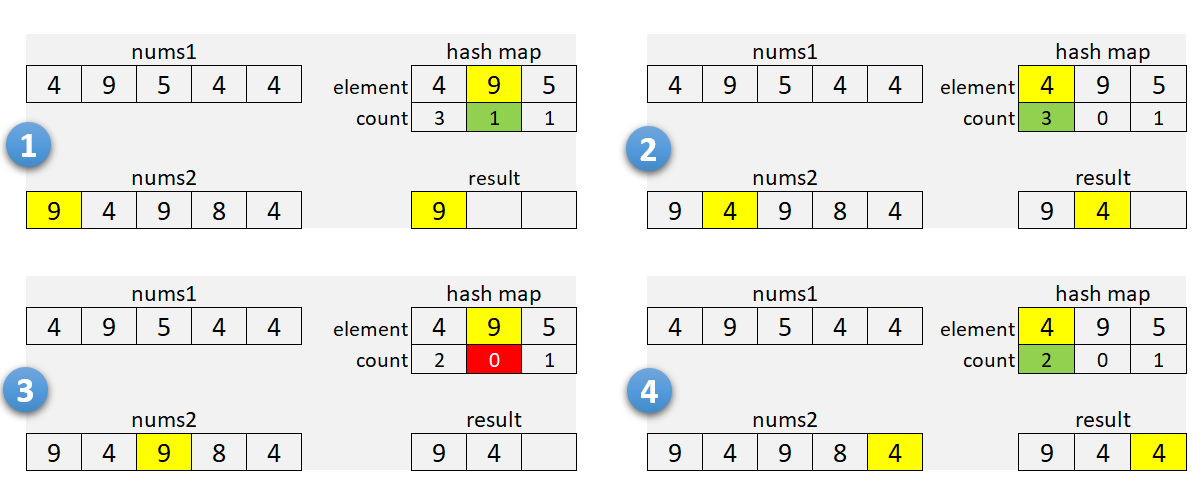
If an interviewer gives you this problem, your first question should be - how should I handle duplicates? Your second question, perhaps, can be about the order of inputs and outputs. Such questions manifest your problem-solving skills, and help you steer to the right solution.

The [solution](https://leetcode.com/problems/intersection-of-two-arrays/solution/) for the previous problem, [349. Intersection of Two Arrays](https://leetcode.com/problems/intersection-of-two-arrays/), talks about approaches when each number in the output must be unique. For this problem, we need to adapt those approaches so that numbers in the result appear as many times as they do in both arrays.

#### **Approach 1: Hash Map**

For the previous problem, we used a hash set to achieve a linear time complexity. Here, we need to use a hash map to track the count for each number.

We collect numbers and their counts from one of the arrays into a hash map. Then, we iterate along the second array, and check if the number exists in the hash map and its count is positive. If so - add the number to the result and decrease its count in the hash map.



It's a good idea to check array sizes and use a hash map for the smaller array. It will reduce memory usage when one of the arrays is very large.

**Algorithm**

1. If nums1 is larger than nums2, swap the arrays.
2. For each element in nums1:
   * Add it to the hash map m.
     + Increment the count if the element is already there.
3. Initialize the insertion pointer (k) with zero.
4. Iterate along nums2:
   * If the current number is in the hash map and count is positive:
     + Copy the number into nums1[k], and increment k.
     + Decrement the count in the hash map.
5. Return first k elements of nums1.

For our solutions here, we use one of the arrays to store the result. As we find common numbers, we copy them to the first array starting from the beginning. This idea is from [this solution](https://leetcode.com/problems/intersection-of-two-arrays-ii/discuss/82405/Simple-Java-Solution) by [sankitgupta](https://leetcode.com/sankitgupta/).

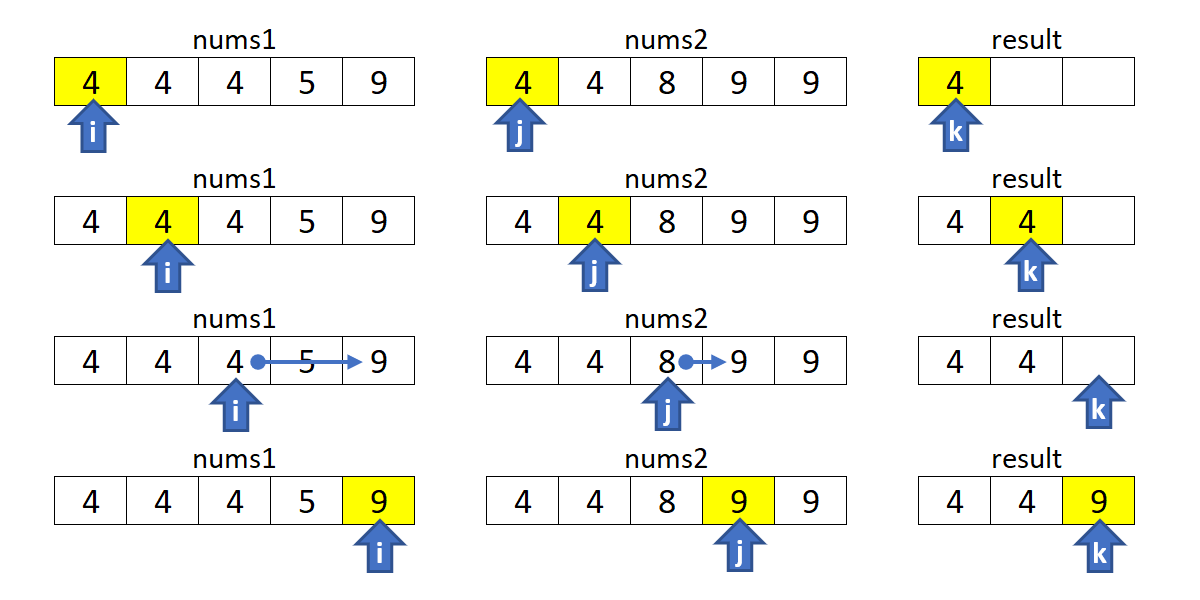
|  |
| --- |
| **public int[] intersect(int[] nums1, int[] nums2) {**  **Arrays.sort(nums1);**  **Arrays.sort(nums2);**  **int i = 0, j = 0, k = 0;**  **while (i < nums1.length && j < nums2.length) {**  **if (nums1[i] < nums2[j]) {**  **++i;**  **} else if (nums1[i] > nums2[j]) {**  **++j;**  **} else {**  **nums1[k++] = nums1[i++];**  **++j;**  **}**  **}**  **return Arrays.copyOfRange(nums1, 0, k);**  **}** |

**Complexity Analysis**

* Time Complexity: \mathcal{O}(n + m)O(*n*+*m*), where n*n* and m*m* are the lengths of the arrays. We iterate through the first, and then through the second array; insert and lookup operations in the hash map take a constant time.
* Space Complexity: \mathcal{O}(\min(n, m))O(min(*n*,*m*)). We use hash map to store numbers (and their counts) from the smaller array.

#### **Approach 2: Sort**

You can recommend this method when the input is sorted, or when the output needs to be sorted. Here, we sort both arrays (assuming they are not sorted) and use two pointers to find common numbers in a single scan.



**Algorithm**

1. Sort nums1 and nums2.
2. Initialize i, j and k with zero.
3. Move indices i along nums1, and j through nums2:
   * Increment i if nums1[i] is smaller.
   * Increment j if nums2[j] is smaller.
   * If numbers are the same, copy the number into nums1[k], and increment i, j and k.
4. Return first k elements of nums1.

|  |
| --- |
| **public int[] intersect(int[] nums1, int[] nums2) {**  **Arrays.sort(nums1);**  **Arrays.sort(nums2);**  **int i = 0, j = 0, k = 0;**  **while (i < nums1.length && j < nums2.length) {**  **if (nums1[i] < nums2[j]) {**  **++i;**  **} else if (nums1[i] > nums2[j]) {**  **++j;**  **} else {**  **nums1[k++] = nums1[i++];**  **++j;**  **}**  **}**  **return Arrays.copyOfRange(nums1, 0, k);**  **}** |

**Complexity Analysis**

* Time Complexity: \mathcal{O}(n\log{n} + m\log{m})O(*n*log*n*+*m*log*m*), where n*n* and m*m* are the lengths of the arrays. We sort two arrays independently, and then do a linear scan.
* Space Complexity: from \mathcal{O}(\log{n} + \log{m})O(log*n*+log*m*) to \mathcal{O}(n + m)O(*n*+*m*), depending on the implementation of the sorting algorithm. For the complexity analysis purposes, we ignore the memory required by inputs and outputs.

#### **Approach 3: Built-in Intersection**

This is similar to [Approach 2](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-2-sort). Instead of iterating with two pointers, we use a built-in function to find common elements. In C++, we can use set\_intersection for sorted arrays (or multisets).

The [retainAll](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractCollection.html#retainAll-java.util.Collection-) method in Java, unfortunately, does not care how many times an element occurs in the other collection. You can use the [retainOccurrences](https://guava.dev/releases/23.0/api/docs/com/google/common/collect/Multisets.html#retainOccurrences-com.google.common.collect.Multiset-com.google.common.collect.Multiset-) method of the multiset implementation in [Guava](https://guava.dev/releases/16.0/api/docs/com/google/common/collect/Multiset.html).

**Algorithm**

Note that set\_intersection returns the position past the end of the produced range, so it can be used as an input for the erase function. The idea is from [this solution](https://leetcode.com/problems/intersection-of-two-arrays-ii/discuss/82269/Short-Python-C%2B%2B) by [StefanPochmann](https://leetcode.com/stefanpochmann/).

|  |
| --- |
| **vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {**  **sort(begin(nums1), end(nums1));**  **sort(begin(nums2), end(nums2));**  **nums1.erase(set\_intersection(begin(nums1), end(nums1),**  **begin(nums2), end(nums2), begin(nums1)), end(nums1));**  **return nums1;**  **}** |

**Complexity Analysis**

* Same as for [approach 2](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach2complexity) above.

#### **Follow-up Questions**

1. What if the given array is already sorted? How would you optimize your algorithm?
   * We can use either [Approach 2](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-2-sort) or [Approach 3](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-3-built-in-intersection), dropping the sort of course. It will give us linear time and constant memory complexity.
2. What if nums1's size is small compared to nums2's size? Which algorithm is better?
   * [Approach 1](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-1-hash-map) is a good choice here as we use a hash map for the smaller array.
3. What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?
   * If nums1 fits into the memory, we can use [Approach 1](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-1-hash-map) to collect counts for nums1 into a hash map. Then, we can sequentially load and process nums2.
   * If neither of the arrays fit into the memory, we can apply some partial processing strategies:
     + Split the numeric range into subranges that fits into the memory. Modify [Approach 1](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-1-hash-map) to collect counts only within a given subrange, and call the method multiple times (for each subrange).
     + Use an external sort for both arrays. Modify [Approach 2](https://leetcode.com/problems/intersection-of-two-arrays-ii/solution/#approach-2-sort) to load and process arrays sequentially.