350. Intersection of Two Arrays II

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

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]
Explanation: [9,4] is also accepted.

```
def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        res = []
        freq1 = collections.Counter(nums1)
        freq2 = collections.Counter(nums2)
        for key in freq1.keys():
            if key in freq2:
                temp = min(freq1[key],freq2[key])
                res = res+[key]*temp
        return res
```

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?

```
def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        nums1.sort()
        nums2.sort()
        res = []
        a = 0
        b = 0
```

```
        while a < len(nums1) and b < len(nums2):
            if nums1[a] == nums2[b]:
                res.append(nums1[a])
                a += 1
                b += 1
            elif nums1[a] < nums2[b]:
                a += 1
            else:
                b += 1
        return res
```

- What if `nums1`'s size is small compared to `nums2`'s size? Which algorithm is better?

Context:

A very frequent Facebook follow up: what if only one of the inputs are super large? e.g. len(nums1) = 3, len(nums2) = 1000. Can we do better than 2-pointers which run in O(n+m) time? Answer is yes: we achieve O(nlogm) in this case, n < m.

Assumption:

nums1 and nums2 are already sorted.

Explanation:

This approach is efficient when only one of the input is really large. Therefore, we first figure out which is the shorter array and loop through it, for every element, we binary search it on the other array. One thing to note is that the question asks to include duplicates, therefore when we binary search, we need to find the left-most matching number. Since the inputs are sorted, next time we perform a binary search, the low should start the previously found index+1.

- 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?

External Sorting