Intersection of Two Arrays II

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

Example:
Given *nums1* = `[1, 2, 2, 1]`, *nums2* = `[2, 2]`, return `[2, 2]`.

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 *num2*'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 1

m: nums1.size n: nums2.size

Hash table solution: Time: O(m + n) Space: O(m + n)

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        unordered_map<int, int> dict;
        vector<int> res;
        for(int i = 0; i < (int)nums1.size(); i++) dict[nums1[i]]++;
        for(int i = 0; i < (int)nums2.size(); i++)
            if(--dict[nums2[i]] >= 0) res.push_back(nums2[i]);
        return res;
    }
};
```

Hash table solution2: Time: O(m + n) Space: O(m)

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        unordered_map<int, int> dict;
        vector<int> res;
        for(int i = 0; i < (int)nums1.size(); i++) dict[nums1[i]]++;
        for(int i = 0; i < (int)nums2.size(); i++)
            if(dict.find(nums2[i]) != dict.end() && --dict[nums2[i]] >= 0) res.pu
sh_back(nums2[i]);
        return res;
    }
};
```

Sort and two pointers Solution: Time: O(max(m, n) log(max(m, n))) Space: O(m + n)

```cpp
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        sort(nums1.begin(), nums1.end());
        sort(nums2.begin(), nums2.end());
        int n1 = (int)nums1.size(), n2 = (int)nums2.size();
        int i1 = 0, i2 = 0;
        vector<int> res;
        while(i1 < n1 && i2 < n2){
            if(nums1[i1] == nums2[i2]) {
                res.push_back(nums1[i1]);
                i1++;
                i2++;
            }
            else if(nums1[i1] > nums2[i2]){
                i2++;
            }
            else{
                i1++;
            }
        }
        return res;
    }
};
```

written by sxycwzwzq original link here

Solution 2

```
public class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        ArrayList<Integer> result = new ArrayList<Integer>();
        for(int i = 0; i < nums1.length; i++)
        {
            if(map.containsKey(nums1[i])) map.put(nums1[i], map.get(nums1[i])+1);
            else map.put(nums1[i], 1);
        }

        for(int i = 0; i < nums2.length; i++)
        {
            if(map.containsKey(nums2[i]) && map.get(nums2[i]) > 0)
            {
                result.add(nums2[i]);
                map.put(nums2[i], map.get(nums2[i])-1);
            }
        }

       int[] r = new int[result.size()];
       for(int i = 0; i < result.size(); i++)
       {
           r[i] = result.get(i);
       }

       return r;
    }
}
```

written by VanillaCoke original link here

Solution 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 only nums2 cannot fit in memory, put all elements of nums1 into a HashMap, read chunks of array that fit into the memory, and record the intersections.
- If both nums1 and nums2 are so huge that neither fit into the memory, sort them individually (external sort), then read 2 elements from each array at a time in memory, record intersections.

written by vishalhemnani original link here