<https://leetcode.com/problems/intersection-of-two-arrays/>

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

**Example 1:**

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

Output: [2]

**Example 2:**

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

Output: [9,4]

Explanation: [4,9] is also accepted.

**Constraints:**

* 1 <= nums1.length, nums2.length <= 1000
* 0 <= nums1[i], nums2[i] <= 1000

**Attempt 1: 2023-02-28**

**Solution 1:  Hash Table (10 min)**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<Integer>();

for(int num : nums1) {

set.add(num);

}

Set<Integer> tmp = new HashSet<Integer>();

for(int num : nums2) {

if(set.contains(num)) {

tmp.add(num);

}

}

return tmp.stream().mapToInt(i -> i).toArray();

}

}

Time Complexity:O(n)

Space Complexity:O(n)

**Refer to**

<https://leetcode.com/problems/intersection-of-two-arrays/solutions/81969/three-java-solutions/>

Use two hash sets

Time complexity: O(n)

public class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<>();

Set<Integer> intersect = new HashSet<>();

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

set.add(nums1[i]);

}

for (int i = 0; i < nums2.length; i++) {

if (set.contains(nums2[i])) {

intersect.add(nums2[i]);

}

}

int[] result = new int[intersect.size()];

int i = 0;

for (Integer num : intersect) {

result[i++] = num;

}

return result;

}

}

**Solution 2: Two Pointers (10 min)**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<Integer>();

Arrays.sort(nums1);

Arrays.sort(nums2);

int i = 0;

int j = 0;

while(i < nums1.length && j < nums2.length) {

if(nums1[i] > nums2[j]) {

j++;

} else if(nums1[i] < nums2[j]) {

i++;

} else {

set.add(nums1[i]);

i++;

j++;

}

}

return set.stream().mapToInt(k -> k).toArray();

}

}

Time Complexity:O(nlogn)

Space Complexity:O(n)

**Refer to**

<https://leetcode.com/problems/intersection-of-two-arrays/solutions/81969/three-java-solutions/>

Sort both arrays, use two pointers

Time complexity: O(nlogn)

public class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<>();

Arrays.sort(nums1);

Arrays.sort(nums2);

int i = 0;

int j = 0;

while (i < nums1.length && j < nums2.length) {

if (nums1[i] < nums2[j]) {

i++;

} else if (nums1[i] > nums2[j]) {

j++;

} else {

set.add(nums1[i]);

i++;

j++;

}

}

int[] result = new int[set.size()];

int k = 0;

for (Integer num : set) {

result[k++] = num;

}

return result;

}

}

**Solution 3: Binary Search (10 min)**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<Integer>();

Arrays.sort(nums2);

for(int num : nums1) {

if(binarySearch(num, nums2)) {

set.add(num);

}

}

return set.stream().mapToInt(k -> k).toArray();

}

private boolean binarySearch(int num, int[] nums) {

int lo = 0;

int hi = nums.length - 1;

while(lo <= hi) {

int mid = lo + (hi - lo) / 2;

if(nums[mid] == num) {

return true;

} else if(nums[mid] > num) {

hi = mid - 1;

} else {

lo = mid + 1;

}

}

return false;

}

}

Time Complexity:O(nlogn)

Space Complexity:O(n)

**Refer to**

<https://leetcode.com/problems/intersection-of-two-arrays/solutions/81969/three-java-solutions/>

Binary search

Time complexity: O(nlogn)

public class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

Set<Integer> set = new HashSet<>();

Arrays.sort(nums2);

for (Integer num : nums1) {

if (binarySearch(nums2, num)) {

set.add(num);

}

}

int i = 0;

int[] result = new int[set.size()];

for (Integer num : set) {

result[i++] = num;

}

return result;

}

public boolean binarySearch(int[] nums, int target) {

int low = 0;

int high = nums.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (nums[mid] == target) {

return true;

}

if (nums[mid] > target) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return false;

}

}