<https://leetcode.com/problems/find-all-duplicates-in-an-array/>

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears* ***twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

**Example 1:**

Input: nums = [4,3,2,7,8,2,3,1]

Output: [2,3]

**Example 2:**

Input: nums = [1,1,2]

Output: [1]

**Example 3:**

Input: nums = [1]

Output: []

**Constraints:**

* n == nums.length
* 1 <= n <= 105
* 1 <= nums[i] <= n
* Each element in nums appears **once** or **twice**.

**Attempt 1: 2023-09-11**

**Solution 1: Brute Force (10 min, TLE, 26/28)**

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

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

for(int j = i + 1; j < nums.length; j++) {

if(nums[i] == nums[j]) {

result.add(nums[i]);

}

}

}

return result;

}

}

Time Complexcity: O(N\*N)

Space Complexcity: O(1)

**Refer to**

<https://leetcode.com/problems/find-all-duplicates-in-an-array/solutions/775798/c-four-solution-o-n-n-to-o-n-explain-all/>

**1. Brute force Approach**

Idea - Do Check Double Time For Each element

Time Complexcity - O(N\*N) **<=Give You TLE**

Space Complexcity - O(1)

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

if(nums.empty())return {};

vector<int>ans;

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

for(int j=i+1;j<nums.size();j++){

if(nums[i]!=nums[j])continue;

else{

ans.push\_back(nums[i]);

break;

}

}

}

return ans;

}

};

**Solution 2: Sorting array first (10 min)**

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

if(nums.length == 1) {

return result;

}

Arrays.sort(nums);

for(int i = 1; i < nums.length; i++) {

if(nums[i - 1] == nums[i]) {

result.add(nums[i]);

}

}

return result;

}

}

Time Complexcity: O(N\*logN)

Space Complexcity: O(1)

**Refer to**

<https://leetcode.com/problems/find-all-duplicates-in-an-array/solutions/775798/c-four-solution-o-n-n-to-o-n-explain-all/>

**2.Using Sort Solution**

Idea - Do **sort The array** First Then Track Adjacent Element Is Same Or Not [Can be Done By XOR or Have an count Element]

Time Complexcity - O(N\*Log N)

Space Complexcity - O(1)

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

if(nums.empty())return {};

vector<int>ans;

sort(begin(nums),end(nums));

int s = nums[0];

for(int i=1;i<nums.size();i++){

if(!(s^nums[i])){

ans.push\_back(nums[i]),i+=1;

if(i<nums.size())s=nums[i];

else break;

}

else s = nums[i];

}

return ans;

}

};

**Solution 3: Hash Table (10 min)**

Use HashMap for frequency

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

Map<Integer, Integer> freq = new HashMap<>();

for(int num : nums) {

freq.put(num, freq.getOrDefault(num, 0) + 1);

}

for(Map.Entry<Integer, Integer> entry : freq.entrySet()) {

if(entry.getValue() == 2) {

result.add(entry.getKey());

}

}

return result;

}

}

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

Use int[] array for frequency

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

int n = nums.length;

int[] freq = new int[n + 1];

for(int num : nums) {

freq[num]++;

}

for(int i = 0; i <= n; i++) {

if(freq[i] == 2) {

result.add(i);

}

}

return result;

}

}

Time Complexcity: O(N)

Space Complexcity: O(N)

**Refer to**

<https://leetcode.com/problems/find-all-duplicates-in-an-array/solutions/775798/c-four-solution-o-n-n-to-o-n-explain-all/>

**3. Using Unordered map**

Idea - Take An unordered\_map To store frequency Of each Element And Return those Having Frequency **2**

Time Complexcity - O(N)

Space Complexcity - O(N)

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

if(nums.empty())return {};

vector<int>ans;

unordered\_map<int,int>mp;

for(int no:nums)mp[no]++;

for(auto it:mp)if(it.second==2)ans.push\_back(it.first);

return ans;

}

};

**Solution 4: Efficient way to flip original element to negative (30 min)**

class Solution {

public List<Integer> findDuplicates(int[] nums) {

// As condition for input 1 <= nums[i] <= n, we can use strategy:

// When find a number i, flip the number at position i-1 to negative.

// if the number at position i-1 is already negative,

// i is the number that occurs twice.

List<Integer> result = new ArrayList<>();

int n = nums.length;

for(int num : nums) {

// Since its an inplace flip number to negative value on original

// 'nums' array, we have to add Math.abs(num) to restore it back

// to positive number when calcuating index

// int index = num - 1; -> ArrayIndexOutofBoundary exception

int index = Math.abs(num) - 1;

if(nums[index] < 0) {

result.add(index + 1);

}

nums[index] = -nums[index];

}

return result;

}

}

Time Complexcity: O(N)

Space Complexcity: O(1)

**Refer to**

<https://grandyang.com/leetcode/442/>

这道题给了我们一个数组，数组中的数字可能出现一次或两次，让我们找出所有出现两次的数字，由于之前做过一道类似的题目[Find the Duplicate Number](http://www.cnblogs.com/grandyang/p/4843654.html)，所以不是完全无从下手。**这类问题的一个重要条件就是1 ≤ a[i] ≤ n (n = size of array)，不然很难在O(1)空间和O(n)时间内完成。**首先来看一种正负替换的方法，**这类问题的核心是就是找nums[i]和nums[nums[i] - 1]的关系**，我们的做法是，对于每个nums[i]，我们将其对应的nums[nums[i] - 1]取相反数，如果其已经是负数了，说明之前存在过，我们将其加入结果res中即可，参见代码如下：

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

vector<int> res;

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

int idx = abs(nums[i]) - 1;

if (nums[idx] < 0) res.push\_back(idx + 1);

nums[idx] = -nums[idx];

}

return res;

}

};

**Refer to**

<https://leetcode.com/problems/find-all-duplicates-in-an-array/solutions/775798/c-four-solution-o-n-n-to-o-n-explain-all/>

Assume I/O Array A -[ 4, 3, 2, 7, 8, 2, 3, 1 ]

index - 0 1 2 3 4 5 6 7

index element Todo

0 A[0] = 4 A[4-1] Not negative

do it negetive mean we visited 4

array - [4,3,2,-7,8,2,3,1]

1 A[1] = 3 A[3-1] Not negative

do it negetive mean we visited 3

array - [4,3,-2,-7,8,2,3,1]

2 A[2] = 2 A[2-1] Not negative

do it negetive mean we visited 2

array - [4,-3,-2,-7,8,2,3,1]

3 A[3] = 7 A[7-1] Not negative

do it negetive mean we visited 7

Array- [4,-3,-2,-7,8,2,-3,1]

4 A[4] = 8 A[8-1] Not negative

do it negetive mean we visited 8

Array- [4,-3,-2,-7,8,2,-3,-1]

5 A[5] = 2 A[2-1] is Negative Mean It is A

duplicate ele so Put it into ans array

Array- [4,-3,-2,-7,8,2,-3,-1]

6 A[6] = 3 A[3-1] is Negative Mean It is A

duplicate ele so Put it into ans array

Array- [4,-3,-2,-7,8,2,-3,-1]

7 A[7] = 1 A[1-1] Not negative

do it negetive mean we visited 1

Array- [-4,-3,-2,-7,8,2,-3,-1]

So we Have {2 , 3 } <= Here For ans

**Refer to**

<https://leetcode.com/problems/find-all-duplicates-in-an-array/solutions/92387/java-simple-solution/>

public class Solution {

// when find a number i, flip the number at position i-1 to negative.

// if the number at position i-1 is already negative, i is the number that occurs twice.

public List<Integer> findDuplicates(int[] nums) {

List<Integer> res = new ArrayList<>();

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

int index = Math.abs(nums[i])-1;

if (nums[index] < 0)

res.add(Math.abs(index+1));

nums[index] = -nums[index];

}

return res;

}

}

**Solution 5: Efficient way to keep swapping if nums[i] != nums[nums[i] - 1] (30 min)**

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

int n = nums.length;

for(int i = 0; i < n; i++) {

if(nums[i] != nums[nums[i] - 1]) {

swap(nums, i, nums[i] - 1);

// The i-- here is very likely the strategy

// used in L84.Largest Rectangle in Histogram

// which keep holding the position until

// necessary move

// e.g nums = [4,3,2,7,8,2,3,1]

// round 1: swap(nums,0,3) -> [7,3,2,4,8,2,3,1]

// round 2: swap(nums,0,6) -> [3,3,2,4,8,2,7,1]

// round 3: swap(nums,0,2) -> [2,3,3,4,8,2,7,1]

// round 4: swap(nums,0,1) -> [3,2,3,4,8,2,7,1]

// round 5: nums[0] = nums[nums[0] - 1] = 3

// till round 5, the i keep as 0, then round 6

// increase i from 0 to 1

// round 6: nums[1] = nums[nums[1] - 1] = 2

// then round 7 increase i from 1 to 2

// round 7: nums[2] = nums[nums[2] - 1] = 3

// then round 8 increase i from 2 to 3

// round 8: nums[3] = nums[nums[3] - 1] = 4

// then round 9 increase i from 3 to 4

// ...

// after all nums = [1,2,3,4,3,2,7,8]

i--;

}

}

for(int i = 0; i < n; i++) {

if(nums[i] != i + 1) {

result.add(nums[i]);

}

}

return result;

}

private void swap(int[] nums, int i, int j) {

int tmp = nums[i];

nums[i] = nums[j];

nums[j] = tmp;

}

}

Time Complexcity: O(N)

Space Complexcity: O(1)

**Refer to**

<https://grandyang.com/leetcode/442/>

下面这种方法是将nums[i]置换到其对应的位置nums[nums[i]-1]上去，比如对于没有重复项的正确的顺序应该是[1, 2, 3, 4, 5, 6, 7, 8]，而我们现在却是[4,3,2,7,8,2,3,1]，我们需要把数字移动到正确的位置上去，比如第一个4就应该和7先交换个位置，以此类推，最后得到的顺序应该是[1, 2, 3, 4, 3, 2, 7, 8]，我们最后在对应位置检验，如果nums[i]和i+1不等，那么我们将nums[i]存入结果res中即可，参见代码如下：

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

vector<int> res;

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

if (nums[i] != nums[nums[i] - 1]) {

swap(nums[i], nums[nums[i] - 1]);

--i;

}

}

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

if (nums[i] != i + 1) res.push\_back(nums[i]);

}

return res;

}

};

**Solution 6: Efficient way to find number over boundary after +n (60 min)**

class Solution {

public List<Integer> findDuplicates(int[] nums) {

List<Integer> result = new ArrayList<>();

int n = nums.length;

for(int i = 0; i < n; i++) {

nums[(nums[i] - 1) % n] += n;

}

for(int i = 0; i < n; i++) {

if(nums[i] > 2 \* n) {

result.add(i + 1);

}

}

return result;

}

}

Time Complexcity: O(N)

Space Complexcity: O(1)

**Refer to**

<https://grandyang.com/leetcode/442/>

下面这种方法是在**nums[nums[i]-1]位置累加数组长度n，注意nums[i]-1有可能越界，所以我们需要对n取余，最后要找出现两次的数只需要看nums[i]的值是否大于2n即可**，最后遍历完nums[i]数组为[12, 19, 18, 15, 8, 2, 11, 9]，我们发现有两个数字19和18大于2n，那么就可以通过i+1来得到正确的结果2和3了，参见代码如下

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

vector<int> res;

int n = nums.size();

for (int i = 0; i < n; ++i) {

nums[(nums[i] - 1) % n] += n;

}

for (int i = 0; i < n; ++i) {

if (nums[i] > 2 \* n) res.push\_back(i + 1);

}

return res;

}

};