<https://leetcode.com/problems/majority-element-ii/description/>

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

**Example 1:**

**Input:** nums = [3,2,3]

**Output:** [3]

**Example 2:**

**Input:** nums = [1]

**Output:** [1]

**Example 3:**

**Input:** nums = [1,2]

**Output:** [1,2]

**Constraints:**

1 <= nums.length <= 5 \* 10^4

-10^9 <= nums[i] <= 10^9

**Follow up:** Could you solve the problem in linear time and in O(1) space?

**Attempt 1: 2025-04-09**

**Refer to Deepseek**

**Problem Description**

Given an integer array of size n, find all elements that appear more than ⌊n/3⌋ times. The solution should run in linear time and use O(1) space.

**Approaches**

**Boyer-Moore Voting Algorithm (Extended)**

This is an extension of the Boyer-Moore algorithm we used for the standard majority element problem. Since we're looking for elements that appear more than n/3 times, there can be at most two such elements.

**Solution Code**

class Solution {

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

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

if (nums == null || nums.length == 0) {

return result;

}

// Initialize candidates and their counts

int candidate1 = nums[0], candidate2 = nums[0];

int count1 = 0, count2 = 0;

// First pass to find potential candidates

for (int num : nums) {

if (num == candidate1) {

count1++;

} else if (num == candidate2) {

count2++;

} else if (count1 == 0) {

candidate1 = num;

count1 = 1;

} else if (count2 == 0) {

candidate2 = num;

count2 = 1;

} else {

count1--;

count2--;

}

}

// Second pass to verify the candidates

count1 = 0;

count2 = 0;

for (int num : nums) {

if (num == candidate1) {

count1++;

} else if (num == candidate2) {

count2++;

}

}

// Check if candidates meet the n/3 requirement

if (count1 > nums.length / 3) {

result.add(candidate1);

}

if (count2 > nums.length / 3 && candidate2 != candidate1) {

result.add(candidate2);

}

return result;

}

}

**Explanation**

Initialization: We start with two candidate variables and their respective counts set to zero.

First Pass:

For each number in the array:

If it matches either candidate, increment that candidate's count

If a count reaches zero, replace that candidate with the current number

If it doesn't match either candidate, decrement both counts

Second Pass:

Reset the counts

Count occurrences of each candidate in the original array

Verification:

Add candidates to the result if they appear more than n/3 times

Ensure we don't add duplicates (when all elements are the same)

**Why It Works**

There can be at most two elements that appear more than n/3 times

The first pass identifies potential candidates

The second pass verifies if these candidates actually meet the frequency requirement

**Time and Space Complexity**

Time Complexity: O(n) - Two passes through the array

Space Complexity: O(1) - Uses constant extra space

**Alternative Solutions**

**Hash Map Solution (Not O(1) space)**

class Solution {

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

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

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

for (int num : nums) {

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

}

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

if (entry.getValue() > nums.length / 3) {

result.add(entry.getKey());

}

}

return result;

}

}

The Boyer-Moore approach is preferred for its O(1) space complexity, while the hash map solution uses O(n) space. The voting algorithm is more efficient for large input sizes.

**Refer to**

<https://www.cnblogs.com/grandyang/p/4606822.html>

这道题让我们求出现次数大于 n/3 的数字，而且限定了时间和空间复杂度，那么就不能排序，也不能使用 HashMap，这么苛刻的限制条件只有一种方法能解了，那就是摩尔投票法 Moore Voting，这种方法在之前那道题 [Majority Element](http://www.cnblogs.com/grandyang/p/4233501.html) 中也使用了。题目中给了一条很重要的提示，让先考虑可能会有多少个这样的数字，经过举了很多例子分析得出，任意一个数组出现次数大于 n/3 的数最多有两个，具体的证明博主就不会了，博主也不是数学专业的（热心网友[用手走路](https://www.cnblogs.com/grandyang/p/4606822.html#3277653)提供了证明：如果有超过两个，也就是至少三个数字满足“出现的次数大于 n/3”，那么就意味着数组里总共有超过 3\*(n/3) = n 个数字，这与已知的数组大小矛盾，所以，只可能有两个或者更少）。那么有了这个信息，使用投票法的核心是找出两个候选数进行投票，需要两遍遍历，第一遍历找出两个候选数，第二遍遍历重新投票验证这两个候选数是否为符合题意的数即可，选候选数方法和前面那篇 [Majority Element](http://www.cnblogs.com/grandyang/p/4233501.html) 一样，由于之前那题题目中限定了一定会有大多数存在，故而省略了验证候选众数的步骤，这道题却没有这种限定，即满足要求的大多数可能不存在，所以要有验证，参加代码如下：

class Solution {

public:

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

vector<int> res;

int a = 0, b = 0, cnt1 = 0, cnt2 = 0, n = nums.size();

for (int num : nums) {

if (num == a) ++cnt1;

else if (num == b) ++cnt2;

else if (cnt1 == 0) { a = num; cnt1 = 1; }

else if (cnt2 == 0) { b = num; cnt2 = 1; }

else { --cnt1; --cnt2; }

}

cnt1 = cnt2 = 0;

for (int num : nums) {

if (num == a) ++cnt1;

else if (num == b) ++cnt2;

}

if (cnt1 > n / 3) res.push\_back(a);

if (cnt2 > n / 3) res.push\_back(b);

return res;

}

};

<https://segmentfault.com/a/1190000003740925>

上一题中，超过一半的数只可能有一个，所以我们只要投票出一个数就行了。而这题中，超过n/3的数最多可能有两个，所以我们要记录出现最多的两个数。同样的两个candidate和对应的两个counter，如果遍历时，某个候选数和到当前数相等，则给相应计数器加1。如果两个计数器都不为0，则两个计数器都被抵消掉1。如果某个计数器为0了，则将当前数替换相应的候选数，并将计数器初始化为1。最后我们还要遍历一遍数组，确定这两个出现最多的数，是否都是众数。

public class Solution {

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

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

if(nums.length == 0) return res;

int c1 = 1, c2 = 0, n1 = nums[0], n2 = 0;

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

// 如果和某个候选数相等，将其计数器加1

if(nums[i] == n1){

c1++;

} else if(nums[i] == n2){

c2++;

// 如果都不相等，而且计数器都不为0，则计数器都减1

} else if(c1 != 0 && c2 != 0){

c1--;

c2--;

// 如果某个计数器为0，则更新相应的候选数

} else {

if(c1 == 0){

n1 = nums[i];

c1 = 1;

} else {

n2 = nums[i];

c2 = 1;

}

}

}

c1 = 0;

c2 = 0;

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

if(nums[i] == n1) c1++;

else if(nums[i] == n2) c2++;

}

if(c1 > nums.length / 3) res.add(n1);

if(c2 > nums.length / 3) res.add(n2);

return res;

}

}

**Refer to**

[L169.Majority Element (Ref.L229,L2404)](note://WEBdb3b313d0dac3393a975cdd3b9a9abfa)