# 题目

给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数大于 ⌊ n/2 ⌋ 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

**示例 1:**

**输入:** [3,2,3]

**输出:** 3

**示例 2:**

**输入:** [2,2,1,1,1,2,2]

**输出:** 2

# 分析：

## 方法一：排序+中位数

**思路：**

可以将数组先排序，然后出现次数超过一半的必定在数组中间位置。

**代码：**

class Solution {

public:

int majorityElement(vector<int>& nums) {

std::sort(nums.begin(),nums.end());

return nums.at(nums.size()/2);

}

};

## 方法二：哈希表

**思路：**

**代码：**

class Solution {

public:

int majorityElement(vector<int>& nums) {

unordered\_map <int,int> mp;

for(int n:nums)

if(++ mp[n] > nums.size()/2) return n;

//如果当前元素个数大于一半，则表示为多数元素

return -1;

}

};

## 方法三：位运算

**思路：**

**代码：**

class Solution {

public:

int majorityElement(vector<int>& nums) {

int res=0;

for(int i=0;i<32;i++)

{

int ones=0;

for(int n:nums)

ones += (n >> i) & 1;

//位运算法统计每个位置上1出现的次数，每次出现则ones+1

//出现则&后为1，可以作为计数值，否则为0（巧妙！）

res += (ones > nums.size()/2) << i;

//如果1出现次数大于2分之1数组长，1即为这个位置的目标数字

}

return res;

}

};

## 方法四：分治算法

**思路：**

如果数a是数组nums的众数，如果我们将nums分成两部分，那么a必定是至少一部分的众数。

我们可以使用反证法来证明这个结论。假设a既不是左半部分的众数，也不是右半部分的众数，那么a出现的次数少于l / 2 + r / 2次，其中l和r分别是左半部分和右半部分的长度。由于l / 2 + r / 2 <= (l + r) / 2，说明a也不是数组nums的众数，因此出现了矛盾。所以这个结论是正确的。

这样以来，我们就可以使用分治法解决这个问题：将数组分成左右两部分，分别求出左半部分的众数a1以及右半部分的众数a2，随后在a1和a2中选出正确的众数。

**算法：**

我们使用经典的分治算法递归求解，直到所有的子问题都是长度为 1 的数组。长度为 1 的子数组中唯一的数显然是众数，直接返回即可。如果回溯后某区间的长度大于 1，我们必须将左右子区间的值合并。如果它们的众数相同，那么显然这一段区间的众数是它们相同的值。否则，我们需要比较两个众数在整个区间内出现的次数来决定该区间的众数。

**代码：**

class Solution {

int count\_in\_range(vector<int>& nums, int target, int lo, int hi) {

int count = 0;

for (int i = lo; i <= hi; ++i)

if (nums[i] == target)

++count;

return count;

}

int majority\_element\_rec(vector<int>& nums, int lo, int hi) {

if (lo == hi)

return nums[lo];

int mid = (lo + hi) / 2;

int left\_majority = majority\_element\_rec(nums, lo, mid);

int right\_majority = majority\_element\_rec(nums, mid + 1, hi);

if (count\_in\_range(nums, left\_majority, lo, hi) > (hi - lo + 1) / 2)

return left\_majority;

if (count\_in\_range(nums, right\_majority, lo, hi) > (hi - lo + 1) / 2)

return right\_majority;

return -1;

}

public:

int majorityElement(vector<int>& nums) {

return majority\_element\_rec(nums, 0, nums.size() - 1);

}

};

复杂度分析：

时间复杂度：O(nlogn)。

空间复杂度：O(logn)。尽管分治算法没有直接分配额外的数组空间，但在递归的过程中使用了额外的栈空间。算法每次将数组从中间分成两部分，所以数组长度变为 1 之前需要进行 O(logn) 次递归，即空间复杂度为O(logn)。