

## 3 - 1 数组 Array

### No.283 移动零

地址：[🔗 移动零](#)

题目：

#### ● English：

283. Move Zeroes

难度 简单 📖 1109 ☆ 📄 🚫 📌 📄

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

**Note** that you must do this in-place without making a copy of the array.

Example 1:

**Input:** `nums = [0,1,0,3,12]`  
**Output:** `[1,3,12,0,0]`

Example 2:

**Input:** `nums = [0]`  
**Output:** `[0]`

Constraints:

- `1 <= nums.length <= 104`
- `-231 <= nums[i] <= 231 - 1`

#### ● 中文：

283. 移动零

难度 简单 📖 1109 ☆ 📄 🚫 📌 📄

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

**输入:** `[0,1,0,3,12]`  
**输出:** `[1,3,12,0,0]`

说明:

- 必须在原数组上操作，不能拷贝额外的数组。
- 尽量减少操作次数。

思路：利用 `inserting_index` 在原数组上操作

#### 具体分析

- 用变量 `zero_index` 记录非零数将要插入要代替 `0` 元素的位置
- 遍历数组,将非零值赋在要代替 `0` 元素的位置
- 对原非零数位置 赋 `0`

Code：

```
1 # Python3
2 # Did Time : 2021 - 07 - 07
3
4 class Solution:
5
6     def moveZeroes(self, nums: List[int]) -> None:
7         """
```

```

8      Do not return anything, modify nums in-place instead.
9      """
10     zero_index = 0    # position to insert the non zero numbe
11
12     for i in range(0, len(nums)):    # iterate through the nums array
13         if(nums[i] != 0):            # if ith value not equals 0, set its value to zero position
14             nums[zero_index] = nums[i]
15             if(i != zero_index):     # when ith is not the zero index, set ith value as 0
16                 nums[i] = 0
17             zero_index += 1          # move to next zero index

```

```

1  // Java
2  // Did Time : 2021 - 07 - 07
3  class Solution {
4
5      public void moveZeroes(int[] nums) {
6          int j = 0;
7          for (int i = 0; i < nums.length; ++i) {
8              if (nums[i] != 0) {
9                  nums[j] = nums[i];
10                 if (i != j) {
11                     nums[i] = 0;
12                 }
13                 j++;
14             }
15         }
16     }
17 }

```

### 复杂度分析：

1. 时间复杂度： $O(n)$
2. 空间复杂度： $O(1)$

## No.11盛水最多的容器

地址：[🔗 盛水最多的容器](#)

题目：

- English

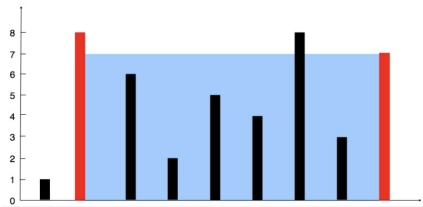
11. Container With Most Water

难度 中等 2588

Given  $n$  non-negative integers  $a_1, a_2, \dots, a_n$ , where each represents a point at coordinate  $(i, a_i)$ .  $n$  vertical lines are drawn such that the two endpoints of the line  $i$  is at  $(i, a_i)$  and  $(i, 0)$ . Find two lines, which, together with the x-axis forms a container, such that the container contains the most water.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]  
Output: 49  
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]  
Output: 1

Example 3:

Input: height = [4,3,2,1,4]  
Output: 16

Example 4:

Input: height = [1,2,1]  
Output: 2

Constraints:

- $n == \text{height.length}$
- $2 \leq n \leq 10^5$
- $0 \leq \text{height}[i] \leq 10^4$

中文：

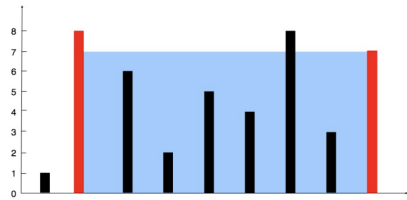
11. 盛最多水的容器

难度 中等 2588

给你  $n$  个非负整数  $a_1, a_2, \dots, a_n$ ，每个数代表坐标中的一个点  $(i, a_i)$ 。在坐标内画  $n$  条垂直线，垂直线  $i$  的两个端点分别为  $(i, a_i)$  和  $(i, 0)$ 。找出其中的两条线，使得它们与  $x$  轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器。

示例 1：



输入：[1,8,6,2,5,4,8,3,7]  
输出：49  
解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例 2：

输入：height = [1,1]  
输出：1

示例 3：

输入：height = [4,3,2,1,4]  
输出：16

示例 4：

输入：height = [1,2,1]  
输出：2

提示：

- $n = \text{height.length}$
- $2 \leq n \leq 3 \times 10^4$
- $0 \leq \text{height}[i] \leq 3 \times 10^4$

思路 1：暴力枚举, 超出时间限制

具体分析

- 从 left bar  $i$  逐个迭代尝试到 right bar  $j$
- $\text{area} = \text{width} * \text{high}$
- $\text{width} = j - i$
- $\text{high} = \min(i - \text{height}, j - \text{height})$  - 最多能装多少水由最小的高度决定

Code：

```
1 // Java
2 // Did Time : 2021 - 07 - 07
3
4 class Solution {
5
6     public int maxArea(int[] height) {
```

```

7      int max = 0;
8
9      // 重要：二次逐个遍历数组，要注意左右边界，不要有反复值，需要牢记
10     for (int i = 0; i < height.length - 1; i++) {
11         for (int j = i + 1; j < height.length; j++) {
12             int area = (j - i) * Math.min(height[i], height[j]);
13             max = Math.max(max, area);
14         }
15     }
16     return max;
17 }
18 }

```

复杂度分析：

- 时间复杂度： $O(n^2)$
- 空间复杂度： $O(1)$

### \* 思路 2：双指针，左右夹逼（重要）

#### 具体分析

双指针，左右边界，左右夹逼，向中间收敛。

左右边界选在最两边，向中间收敛，找到相对高度比较高的两边界，计算面积，直到两边界相遇，返回面积最大者。

说明：左右边界选在最两边，宽度最大，但高度不一定最高；向中间收敛，只关注最高的边界，将其作为新的边界。因为，宽度在缩小，高度若还缩小，面积则一定缩小，就不用考虑了。

### 代码

```

1  // Java
2  // Did Time : 2021 - 07 - 07
3
4  class Solution {
5
6      public int maxArea(int[] height) {
7          int maxArea = 0;
8          for (int i = 0, j = height.length - 1; i < j;) {
9              // i 左边界，向右走；j 右边界，向左走；谁小谁先走，作为迭代条件
10             // height[i++]：当 i 的高度小于 j 的高度时，i++ 既可以使 i 向左走，又可以得到在计算面积时 i 的实际坐标值
11             // height[j--]：当 i 的高度大于 j 的高度时，j-- 使 j 向右走，但在计算面积时 j 的实际坐标值多减了 1
12             // j + 1：计算面积时，为了迭代 j 的实际值多减少了 1
13             int minHeight = height[i] < height[j] ? height[i++] : height[j--];
14
15             // area = (j + 1 - i) * minHeight;
16             maxArea = Math.max(maxArea, (j + 1 - i) * minHeight);
17         }
18         return maxArea;
19     }
20 }

```

### 复杂度分析

- 时间复杂度： $O(n)$
- 空间复杂度： $O(1)$

## No.70 爬楼梯

地址：[爬楼梯](#)

题目：

- English：

70. Climbing Stairs

难度 简单 1733 ☆ 1733 1733 1733 1733 1733

You are climbing a staircase. It takes  $n$  steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

**Input:**  $n = 2$   
**Output:** 2  
**Explanation:** There are two ways to climb to the top.  
1. 1 step + 1 step  
2. 2 steps

Example 2:

**Input:**  $n = 3$   
**Output:** 3  
**Explanation:** There are three ways to climb to the top.  
1. 1 step + 1 step + 1 step  
2. 1 step + 2 steps  
3. 2 steps + 1 step

Constraints:

- $1 \leq n \leq 45$

- 中文：

70. 爬楼梯

难度 简单 1733 ☆ 1733 1733 1733 1733 1733

假设你正在爬楼梯。需要  $n$  阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

注意：给定  $n$  是一个正整数。

示例 1：

**输入：** 2  
**输出：** 2  
**解释：** 有两种方法可以爬到楼顶。  
1. 1 阶 + 1 阶  
2. 2 阶

示例 2：

**输入：** 3  
**输出：** 3  
**解释：** 有三种方法可以爬到楼顶。  
1. 1 阶 + 1 阶 + 1 阶  
2. 1 阶 + 2 阶  
3. 2 阶 + 1 阶

## 思路

懵逼时候：

- 是否能暴力解题？
- 都有什么基本情况？
- 找最近重复逻辑
- if else, for, while, recursing
- 不要用人脑分析 recursing 步骤

Do：

- $n = 1 \rightarrow 1$  种：1 阶
  - $n = 2 \rightarrow 2$  种：1 阶，
  - $n = 3 \rightarrow 3$  种：1 阶，1 阶 + 2 阶，2 阶 + 1 阶
- .....
- $f(n) = f(n - 1) + f(n - 2)$  : Fibonacci

递归会超时，将递归改为 for 循环。

## Code：

```
1  # python3
2
3  class Solution:
4      def climbStairs(self, n: int) -> int:
5          # if n == 0:
6          #     return 0
7          # if n == 1:
8          #     return 1
9          # if n == 2:
10         #     return 2
11
12         if n <= 2:
13             return n
14
15         prepre, pre, cur = 1, 2, 3
16
17         for i in range(3, n + 1):
18             cur = pre + prepre
19             prepre = pre
20             pre = cur
21
22         return cur
23
24
25     # 递归代码
26     # return self.climbStairs(n-1) + self.climbStairs(n-2)
```

复杂度分析

- 时间复杂度：O(n)
- 空间复杂度：O(1)

No.4 两数之和

地址：[🔗 两数之和](#)

题目：

- English：

1. Two Sum

难度 简单👤 11511 ☆📄🔍🔔📌

Given an array of integers `nums` and an integer `target`, return *indices* of the two numbers such that they add up to `target`.

You may assume that each input would have **exactly one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

**Input:** `nums = [2,7,11,15], target = 9`  
**Output:** `[0,1]`  
**Output:** Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Example 2:

**Input:** `nums = [3,2,4], target = 6`  
**Output:** `[1,2]`

Example 3:

**Input:** `nums = [3,3], target = 6`  
**Output:** `[0,1]`

Constraints:

- `2 <= nums.length <= 104`
- `-109 <= nums[i] <= 109`
- `-109 <= target <= 109`
- Only one valid answer exists.

- 中文：

## 1. 两数之和

难度 简单 11511 ☆ 0 0 0 0

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值 `target` 的那两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

输入：nums = [2,7,11,15], target = 9  
输出：[0,1]  
解释：因为 nums[0] + nums[1] == 9，返回 [0, 1]。

示例 2：

输入：nums = [3,2,4], target = 6  
输出：[1,2]

示例 3：

输入：nums = [3,3], target = 6  
输出：[0,1]

提示：

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- 只会存在一个有效答案

## 思路：暴力枚举，两层循环

### 具体分析

两层循环，枚举不同下标,将前一个数与后一个数相加，和等于目标数，返回这两个数值。

### 代码

```
1 // Java
2 class Solution {
3
4     public int[] twoSum(int[] nums, int target) {
5         int[] a = new int[2];
6         int numsSize = nums.length;
7         for (int i = 0; i < numsSize - 1; i++) {
8             for (int j = i + 1; j < numsSize; j++) {
9                 if (nums[i] + nums[j] == target) {
10                     a[0] = i;
11                     a[1] = j;
12                     return a;
13                 }
14             }
15         }
16         return new int[0];
17     }
18 }
```

## 复杂度分析：

- 时间复杂度： $O(n^2)$
- 空间复杂度： $O(1)$



No.15 三数之和（高频老题）

地址：🔗三数之和

题目：

- English：

15. 三数之和

难度 中等👤 3476★📄🔍🔔🔖

给你一个包含  $n$  个整数的数组 `nums`，判断 `nums` 中是否存在三个元素  $a, b, c$ ，使得  $a + b + c = 0$ ？请你找出所有和为  $0$  且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

输入：nums = [-1,0,1,2,-1,-4]

输出：[[-1,-1,2],[-1,0,1]]

示例 2：

输入：nums = []

输出：[]

示例 3：

输入：nums = [0]

输出：[]

提示：

- $0 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

- 中文：

15. 三数之和

难度 中等👤 3476★📄🔍🔔🔖

给你一个包含  $n$  个整数的数组 `nums`，判断 `nums` 中是否存在三个元素  $a, b, c$ ，使得  $a + b + c = 0$ ？请你找出所有和为  $0$  且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

输入：nums = [-1,0,1,2,-1,-4]

输出：[[-1,-1,2],[-1,0,1]]

示例 2：

输入：nums = []

输出：[]

示例 3：

输入：nums = [0]

输出：[]

提示：

- $0 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

审题

- 返回不重复的三元组
- 会有复数、无序

3. 可能不存在，实际要求返回空数组
4. 将问题  $a + b + c = 0$  转换成  $a + b = -c$  考虑
5. 数组内有重复数字，结果可能会有重复

## 思路：夹逼法

### 具体分析

- 双指针，因为不需要下标，可以先排序后夹逼；
- 外循环：固定指针  $k$  循环
  - $k$  针指向 目标值，范围  $0 \sim$  倒数第二个；
- 设置双指针  $i, j$  为  $k$  指针所指目标值右边所查找数据的左右边界；
- 内循环：双指针求和循环
  - $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] = 0 \rightarrow \text{nums}[i] + \text{nums}[j] = -\text{nums}[k]$ ;
  - $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] < 0 \rightarrow \text{nums}[i] + \text{nums}[j] < -\text{nums}[k]$  :
    - 小于目标值， $i++$ , 左指针向右移动，找更大的和；
  - $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] > 0 \rightarrow \text{nums}[i] + \text{nums}[j] > -\text{nums}[k]$  :
    - 大于目标值， $j--$ , 右指针向左移动，找更小的和；
  - $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] == 0$  :
    - 等于目标值，找到结果，记录  $i, j, k$  所指向的值
- 内循环结束条件：
  - $i == j$ ，即，当  $i$  指针与  $j$  指针相遇时，双指针求和遍历完毕，换  $k$  指针所指向的目标值，再次开始内循环，查找满足下一个目标值的和；
- 外循环结束，返回所有符合条件的值；
- 注：可以通过一些边界条件，加速代码。

## 复杂度分析

- 时间复杂度 ( $O^2$ )：固定指针  $k$  循环复杂度  $O(N)$ ，双指针  $i, j$  复杂度为  $O(N)$ ，因为两者嵌套的，所以符合乘法法则，总时间复杂度是  $O(N^2)$ ；
- 空间复杂度  $O(1)$ ：指针使用常数大小的额外空间。

## 代码

```
1 class Solution {
2
3     public List<List<Integer>> threeSum(int[] nums) {
4
5         // 先对数组进行排序
6         Arrays.sort(nums);
7
8         // list 用于存储 结果值
9         List<List<Integer>> res = new ArrayList<>();
10
11         // 外循环：k 指针 指向 目标值，范围是 0 ~ 倒数第二个，当 k = 倒数第二个时，i == j, end
12         for (int k = 0; k < nums.length - 2; k++) {
```

```

13
14 // 如果 nums[k] > 0 : 意味着不存在与目标值相加为0的另外两个数, 停止循环, 进行下一个目标值的查找
15 if (nums[k] > 0) break;
16
17 // 避免重复值再次查找 (排序后相同值会连续排在一起)
18 if (k > 0 && nums[k] == nums[k - 1]) continue;
19
20 // i - 左边界, j - 右边界
21 int i = k + 1, j = nums.length - 1;
22
23 // 内循环 : 双指针求和
24 while (i < j) {
25
26     // 求三者之和
27     int sum = nums[k] + nums[i] + nums[j];
28
29     if (sum < 0) { // 当 和 小于 0 时, 小于目标值
30         while (i < j && nums[i] == nums[++i]); // 左边界跳过重复值, 且右移至下一个不重复的值
31     } else if (sum > 0) { // 当 和 大于 0 时, 大于目标值
32         while (i < j && nums[j] == nums[--j]); // 右边界跳过重复值, 且左移至下一个不重复的值
33     } else { // 当 和 等于 0 时, 等于目标值, 找到结果, 并记录到 result 链表中
34         res.add(
35             new ArrayList<Integer>(Arrays.asList(nums[i], nums[j], nums[k]))
36         );
37         while (i < j && nums[i] == nums[++i]); // 左边界跳过重复值, 且右移至下一个不重复的值
38         while (i < j && nums[j] == nums[--j]); // 右边界跳过重复值, 且左移至下一个不重复的值
39     }
40 }
41 }
42 return res;
43 }
44 }

```

笔记时间：2021-07-08 总结《算法训练营 25期》谭超专栏 --- Benjamin Song