LeetCode 第 15 号问题：三数之和

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

题目来源于 LeetCode 上第 15 号问题：三数之和。

题目描述

给定一个包含 *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？找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例

```
给定数组 nums = [-1, 0, 1, 2, -1, -4],

满足要求的三元组集合为:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

题目解析

最容易想到的就是三重循环暴力法搜索，时间复杂度为 `O(n^3)`. 有点高啊，优化一下.

通过题目我们了解到，主要问题在于 搜索所有满足条件的情况 和 避免重复项，那么我们可以使用 升序数组 + 双指针 有效处理问题并降低时间复杂度.

你可能想知道为啥会选择使用这个方案？

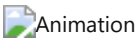
首先数组排序时间复杂度可以达到 `O(NlogN)`，这点时间消耗我们是能接受的，另外根据有序数组的特性，数组重复项会挨在一起，不需要额外的空间存储就能跳过重复项，由于是升序，当发现最左边的数值大于0，就可以及时跳出来结束运算.

双指针可以用来 降维 . 通过遍历数组，取当前下标值为 定值，双指针代表 定值 后面子数组的 首尾数值，通过不断靠近双指针来判断三个值的和。

具体算法流程如下：

1. 特判：对于数组长度 `n`，如果数组为 `null` 或者数组长度小于 `3`，返回 `[ ]`；
2. 数组升序排序；
3. 遍历数组：
    - 若 `num[i] > 0`：因为是升序，所以结果不可能等于0，直接返回结果；
    - 令左指针 `L = i + 1`，右指针 `R = n - 1`，当 `L < R` 时，执行循环：
        - 当 `nums[i] + nums[L] + nums[R] == 0` ，执行循环，判断左指针和右指针是否和下一位置重复， 去除重复解 。并同时将 `L,R` 移到下一位置，寻找新的解；
        - 若 和 大于 `0`，说明 `nums[R]` 太大， `R`指针 左移
        - 若 和 小于 `0`，说明 `nums[L]` 太小， `L`指针 右移

动画描述

 Animation

参考代码

```
 // lang = JavaScript
var threeSum = function(nums) {
    let res = [];
    if (nums == null || nums.length < 3) {
        return res;
    }
    const len = nums.length;
    nums.sort((a, b) => a - b); // 升序
    for (let i = 0; i < len - 2;) {
        const element = nums[i];
        if (element > 0) {
            // 如果当前数字大于0，则三数之和一定大于0，所以结束循环
            break;
        }
        let L = i + 1,
            R = len - 1;
        while (L < R) {
            const sum = element + nums[L] + nums[R];
            if (sum == 0) {
                res.push([element, nums[L], nums[R]]);
                // 左右指针去重 & L+1 & R-1
                while (L < R && nums[L] == nums[++L]);
                while (L < R && nums[R] == nums[--R]);
            }else if (sum < 0) {
                while (L < R && nums[L] == nums[++L]);
            }else {
                while (L < R && nums[R] == nums[--R]);
            }
        }
        //  定值去重
        while (nums[i] == nums[++i]);
    }
    return res;
};
```

复杂度分析

- 时间复杂度：`O(n^2)`

  数组排序`O(NlogN)`，遍历数组`O(n)`，双指针遍历`O(n)`，总体复杂度为`O(NlogN) + O(n) * O(n)`，`O(n^2)`

- 空间复杂度：`O(1)`