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 <= nums.length <= 3000`
- `-10^5 <= nums[i] <= 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 <= nums.length <= 3000`
- `-10^5 <= nums[i] <= 10^5`

审题

1. 返回不重复的三元组
2. 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1：夹逼法

- ⭐ 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 ~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
  - ⭐ nums[i] + nums[j] + nums[k] = 0 → nums[i] + nums[j] = - num[k];
  - ⭐ nums[i] + nums[j] + nums[k] < 0 →nums[i] + nums[j] < - nums[k]：
    - ⭐ 小于目标值，i ++, 左指针向右移动, 找更大的和；
  - ⭐ nums[i] + nums[j] + nums[k] > 0 →nums[i] + nums[j] > - nums[k]:
    - ⭐ 大于目标值，j --, 右指针向左移动，找更小的和；
  - ⭐ nums[i] + nums[j] + 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

```
class Solution {


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


    // 先对数组进行排序
    Arrays.sort(nums);


```

```java
    // list 用于存储 结果值
    List<List<Integer>> res = new ArrayList<>();

    // 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
    for (int k = 0; k < nums.length - 2; k++) {


      // 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
      if (nums[k] > 0) break;

      // 避免重复值再次查找 （排序后相同值会连续排在一起）
      if (k > 0 && nums[k] == nums[k - 1]) continue;


      // i - 左边界，j - 右边界
      int i = k + 1, j = nums.length - 1;


      // 内循环 : 双指针求和
      while (i < j) {

        // 求三者之和
        int sum = nums[k] + nums[i] + nums[j];

        if (sum < 0) {                                // 当 和 小于 0 时，小于目标值
          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
        } else if (sum > 0) {                         // 当 和 大于 0 时，大于目标值
          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
        } else {                                      // 当 和 等于 0 时，等于目标值，找到结果，并记录到 result 链表中
          res.add(
            new ArrayList<Integer>(Arrays.asList(nums[i], nums[j], nums[k]))
            );
          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
          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
        }
      }
    }
    return res;
  }
}
```