15. 3Sum

⏱ Created	@July 15, 2020 3:07 AM
⊘ Difficulty	Medium
☰ LC Url	https://leetcode.com/problems/3sum/
⊘ Importance	
☰ Tag	Two pointers
☰ Video	

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

Constraints:

- `3 <= nums.length <= 3000`

- `10 5  <= nums[i] <= 10 5`

Solution

排序+双指针

详细讲解：

> 力扣
>
> <https://leetcode.cn/problems/3sum/solution/san-shu-zhi-he-by-leetcode-solution/>

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # 双指针
        # 链接：https://leetcode.cn/problems/3sum/solution/san-shu-zhi-he-by-leetcode-solution/
        n = len(nums)
        nums.sort()
        ans = list()

        # 枚举 a
        for first in range(n):
            # 需要和上一次枚举的数不相同
            if first > 0 and nums[first] == nums[first - 1]:
                continue
            # c 对应的指针初始指向数组的最右端
            third = n - 1
            target = -nums[first]
            # 枚举 b
            for second in range(first + 1, n):
                # 需要和上一次枚举的数不相同
                if second > first + 1 and nums[second] == nums[second - 1]:
                    continue
                # 需要保证 b 的指针在 c 的指针的左侧
                while second < third and nums[second] + nums[third] > target:
                    third -= 1
                # 如果指针重合，随着 b 后续的增加
                # 就不会有满足 a+b+c=0 并且 b<c 的 c 了，可以退出循环
                if second == third:
                    break
                if nums[second] + nums[third] == target:
                    ans.append([nums[first], nums[second], nums[third]])

        return ans
```

复杂度分析

- 时间复杂度：$O(N^2)$，其中 N 是数组 $nums$ 的长度。
- 空间复杂度：$O(\log N)$。我们忽略存储答案的空间，额外的排序的空间复杂度为 $O(\log N)$。然而我们修改了输入的数组 $nums$，在实际情况下不一定允许，因此也可以看成使用了一个额外的数组存储了 $nums$ 的副本并进行排序，空间复杂度为 $O(N)$。

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        ArrayList<List<Integer>> res = new ArrayList<List<Integer>>();
        if (nums == null || nums.length <= 2) return res;
        int n = nums.length;
        int i = 0;
        Arrays.sort(nums);
        while (i < n-2) {
            int base = nums[i];
            int left = i + 1;
            int right = n - 1;

            while (left < right) {
                int sum = base + nums[left] + nums[right];
                if (sum == 0) {
                    LinkedList<Integer> list = new LinkedList<Integer>();
                    list.add(base);
                    list.add(nums[left]);
                    list.add(nums[right]);
                    res.add(list);
                    left = moveRight(nums, left+1);
                    right = moveLeft(nums, right-1);
                } else if (sum > 0) {
                    right = moveLeft(nums, right-1);
                } else {
                    left = moveRight(nums, left+1);
                }
            }
            i = moveRight(nums, i+1);
        }
        return res;
    }

    public int moveLeft(int[] nums, int right) {
        while (right == nums.length-1 || (right >= 0 && nums[right] == nums[right+1])) {
            right--;
        }
        return right;
    }
```
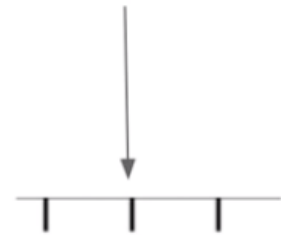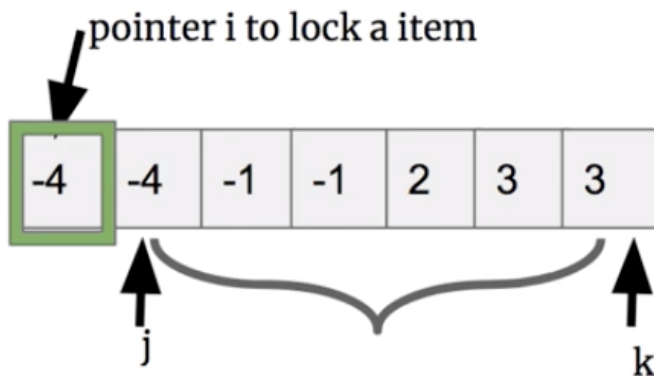
```
    public int moveRight(int[] nums, int left) {
        while (left == 0 || left < nums.length && nums[left] == nums[left-1]) {
            left++;
        }
        return left;
    }
}
```

-1	3	2	-4	-1	-4	3

target = 0

pointer i to lock a item

-4	-4	-1	-1	2	3	3

j   k

Solution: Two Pointers
1. Sort the array
2. Lock one pointer and do two sum with the other two

```java
public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
    int target = 0;
    Arrays.sort(nums);
    // corner case
    if (nums.length < 3) return result;

    for (int i = 0; i < nums.length; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;  // skip duplicates
        int j = i + 1;
        int k = nums.length - 1;
        while (j < k) {
            if (nums[i] + nums[j] + nums[k] < target) {
                j++;
                while (j < k && nums[j] == nums[j - 1]) j++; // skip duplicates
            } else if (nums[i] + nums[j] + nums[k] > target) {
                k--;
                while (j < k && nums[k] == nums[k + 1]) k--; // skip duplicates
            } else {
                result.add(Arrays.asList(nums[i], nums[j], nums[k]));
                j++;
                k--;
                while (j < k && nums[j] == nums[j - 1]) j++; // skip duplicates
                while (j < k && nums[k] == nums[k + 1]) k--; // skip duplicates
            }
        }
    }
    return result;
}
```
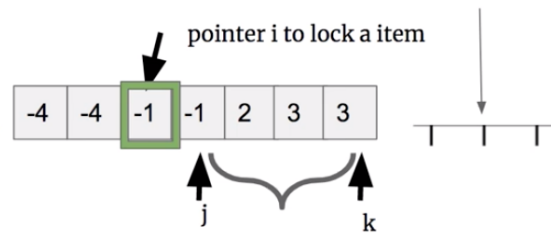
target = 0

pointer i to lock a item

| -4 | -4 | -1 | -1 | 2 | 3 | 3 |

j   k

TIME: O(n^2)
SPACE: O(1)