# 题目

给定一个长度为 n 的整数数组和一个目标值 target，寻找能够使条件 nums[i] + nums[j] + nums[k] < target 成立的三元组 i, j, k 个数（0 <= i < j < k < n）。

示例：

输入: nums = [-2,0,1,3], target = 2

输出: 2

解释: 因为一共有两个三元组满足累加和小于 2:

  [-2,0,1]

[-2,0,3]

进阶：是否能在 O(n2) 的时间复杂度内解决？

# 分析

## 方法一：二分查找

我们首先考虑更简单的情况。题目中要求的是三元组(i,j,k)，我们可以首先尝试一个简单的版本，求出二元组(i,j)。即

给定一个长度为n的数组nums，找出所有的二元组(i,j) 的数目，满足0≤i<j<n 且nums[i]+nums[j]<target。

在这个简化的问题中，我们首先对数组进行排序，随后从小到大枚举 ii，并用二分查找，找出最大的满足nums[i]+nums[j]<target的j，那么我们就可以知道对于当前的i，有j−i对二元组是满足要求的。

解决了二元组的问题后，我们可以发现，在问题变成三元组时，只要在二元组的算法框架外部再套上一层循环就行了。也就是说，我们从小到大枚举i和j，规定i<j，随后在数组上二分查找，找出最大的满足nums[i]+nums[j]+nums[k]<target的k，那么对于当前的i和j，有k−j对三元组是满足要求的。

public int threeSumSmaller(int[] nums, int target) {

Arrays.sort(nums);

int sum = 0;

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

sum += twoSumSmaller(nums, i + 1, target - nums[i]);

}

return sum;

}

private int twoSumSmaller(int[] nums, int startIndex, int target) {

int sum = 0;

for (int i = startIndex; i < nums.length - 1; i++) {

int j = binarySearch(nums, i, target - nums[i]);

sum += j - i;

}

return sum;

}

private int binarySearch(int[] nums, int startIndex, int target) {

int left = startIndex;

int right = nums.length - 1;

while (left < right) {

int mid = (left + right + 1) / 2;

if (nums[mid] < target) {

left = mid;

} else {

right = mid - 1;

}

}

return left;

}

复杂度分析

时间复杂度：O(n^2 log n)。枚举i和j各需要O(n)的时间复杂度，二分查找 kk 需要)O(logn)的时间复杂度，因此总的时间复杂度为O(n2logn)。

空间复杂度：O(1)。

## 方法二：双指针

在方法二中，我们首先解决了一个简化版的问题，随后在外部套上一层循环，就可以解决当前的问题。如果我们能找出更好的解决简化版的问题的方法，就能在更低的时间复杂度内解决当前的问题。

我们首先仍然对数组进行排序，例如{nums} = [3,5,2,8,1]nums=[3,5,2,8,1] 排序后变成[1,2,3,5,8]。target 的值为 7。

[1, 2, 3, 5, 8]

↑ ↑

left right

我们用两个指针left和right分别指向数组中的第一个和最后一个元素，它们的和为1+8=9>target，这说明right不能出现在二元组中（因为left只向左移动，如果此时二者的和已经大于target，那么在left 移动的过程中，二者的和就不可能小于target了），因此需要将right 向左移动一位。

[1, 2, 3, 5, 8]

↑ ↑

left right

现在二者的和为1+5=6<target，那么对于当前的left，应当有right−left=3 对二元组满足要求，它们分别为(1,2),(1,3),(1,5)。在这之后，我们将left向右移动一位。

public int threeSumSmaller(int[] nums, int target) {

Arrays.sort(nums);

int sum = 0;

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

sum += twoSumSmaller(nums, i + 1, target - nums[i]);

}

return sum;

}

private int twoSumSmaller(int[] nums, int startIndex, int target) {

int sum = 0;

int left = startIndex;

int right = nums.length - 1;

while (left < right) {

if (nums[left] + nums[right] < target) {

sum += right - left;

left++;

} else {

right--;

}

}

return sum;

}

复杂度分析

时间复杂度：O(n^2)。在每一步中，要么left向右移动一位，要么right向左移动一位。当left≥right时循环结束，因此它的时间复杂度为O(n)。加上外层的循环，总的时间复杂度为O(n^2)。

空间复杂度：O(1)。

C++：

class Solution

{

public:

int func(vector<int> & nums, int L, int target) //两个数的和小于target

{

int res = 0;

int R = nums.size() - 1;

while (L < R)

{

if (nums[L] + nums[R] < target)

{

res += (R - L);

L += 1;

}

else

R -= 1;

}

return res;

}

int threeSumSmaller(vector<int>& nums, int target)

{

int n = nums.size();

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

int res = 0;

for (int i = 0; i < n - 2; i ++)

{

res += func(nums, i + 1, target - nums[i]);

}

return res;

}

};