在只给一个数组的N数之和问题总结

第15题. 三数之和

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]]

> 本题中如果用暴力枚举通过两层for循环则无法满足“所有满足条件且不重复的三元组”的要求**（其中使用map数组解决则会需要耗费大量时间实现去重问题）**

此时采用双指针的方法

前提条件：该nums数组是排序过的，

> a=nums[i],b=nums[left],c=nums[right]
> 初始时:Left的为i+1,right为数组末尾
> **接下来如何移动left 和right呢， 如果nums[i] + nums[left] + nums[right] > 0 就说明 此时三数之和大了，因为数组是排序后了，所以right下表就应该向左移动，这样才能让三数之和小一些。**
> **如果 nums[i] + nums[left] + nums[right] < 0 说明 此时 三数之和小了，left 就向右移动，才能让三数之和大一些，直到left与right相遇为止。**
> **时间复杂度：O(n^2)。**

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
 vector<vector<int>> ans;
 sort(nums.begin(),nums.end());
 for(int i=0;i<nums.size();i++){
 if(nums[i]>0) return ans;//因为排过序了,后面的元素都比前面的元素大或者等于
 if(i>0&&nums[i]==nums[i-1]) continue;
 int left=i+1;
  int right=nums.size()-1;
  while(left<right){
 //去掉相同的元素
 //while(left<right&&nums[left]==nums[left+1]) left++;
 //while(left<right&&nums[righ]==nums[right-1])right--;
```

```
15
16   if(nums[i]+nums[left]+nums[right]>0){
17   right--;
18   }else if(nums[i]+nums[left]+nums[right]<0){
19   left++;
20   }else{//nums[i]+nums[left]+nums[right]==0
21   ans.push_back(vector<int>{nums[i],nums[right],nums[left]});
22   //去重逻辑应该是在找到一个三元组之后
23   while(left<right&&nums[right]==nums[right-1]) right--;
24   while(left<right&&nums[left]==nums[left+1])left++;
25   right--;
26   left++;
27   }
28   }
29   }
30   return ans;
31       }
32  };
```

双指针法解决四数之和或者五数之和、六数之和、N数之和呢?

第18题. 四数之和

题意：给定一个包含 n 个整数的数组 nums 和一个目标值 target，判断 nums 中是否存在四个元素 a，b，c 和 d ，使得 a + b + c + d 的值与 target 相等？找出所有满足条件且不重复的四元组。

「注意：」

答案中不可以包含重复的四元组。

示例：
给定数组 nums = [1, 0, -1, 0, -2, 2]，和 target = 0。
满足要求的四元组集合为：
[[-1, 0, 0, 1], [-2, -1, 1, 2], [-2, 0, 0, 2]]

> 思路：三数之和的基础上加一个for循环

```
1  class Solution {
2  public:
3      vector<vector<int>> fourSum(vector<int>& nums, int target) {
4   vector<vector<int>> ans;
5   sort(nums.begin(),nums.end());
6   for(int i=0;i<nums.size();i++){
7   //不重复四元组
```

```
8   if(i>0&&nums[i]==nums[i-1]) continue;//之前如果有满足条件的四元组已经记录了
9   for(int j=i+1;j<nums.size();j++){
10   if(j>i+1&&nums[j]==nums[j-1]) continue;
11   int left=j+1;
12   int right=nums.size()-1;
13   while(left<right){
14   int sum=nums[i]+nums[j]+nums[left]+nums[right];
15   if(sum>target){
16   right--;
17   }
18   if(sum<target){
19   left++;
20   }
21   if(sum==target){
22   ans.push_back(vector<int>{nums[i],nums[j],nums[left],nums[right]});
23   //往后去重
24   while(left<right&&nums[right]==nums[right-1])right--;
25   while(left<right&&nums[left]==nums[left+1]) left++;
26   left++;
27   right--;
28   }
29   }
30   }
31   }
32   return ans;
33      }
34  };
```

总结：

四数之和是在同一个数组里面可以用双指针法：

四数之和的双指针解法是两层for循环nums[k] + nums[i]为确定值，依然是循环内有left和right下表作为双指针，找出nums[k] + nums[i] + nums[left] + nums[right] == target的情况，三数之和的时间复杂度是O(n^2)，四数之和的时间复杂度是O(n^3) 。

那么一样的道理，五数之和、六数之和等等都采用这种解法。

对于三数之和双指针法就是将原本暴力O(n^3)的解法，降为O(n^2)的解法，四数之和的双指针解法就是将原本暴力O(n^4)的解法，降为O(n^3)的解法。

如果四数之和在不同数组里面则用哈希法