Discuss

Store ▼

Mock Contest Articles

Note: The solution set must not contain duplicate quadruplets.

Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums such

that a + b + c + d =target ? Find all unique quadruplets in the array which gives the sum of target .

**Example:** 

Explore Problems

LeetCode

Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

## A solution set is:

```
[-1, 0, 0, 1],
   [-2, -1, 1, 2],
   [-2, 0, 0, 2]
Solution
```

This problem is a follow-up of 3Sum, so take a look at that problem first if you haven't. 4Sum and 3Sum are

As you see, 3Sum just wraps Two Sum in an outer loop. As it iterates through each value v, it finds all pairs

very similar; the difference is that we are looking for unique quadruplets instead of triplets.

whose sum is equal to target - v using one of these approaches:

1. Two Sum uses a hash set to check for a matching value.

# 2. Two Sum II uses the two pointers pattern in a sorted array.

catch. If an interviewer asks you to solve 4Sum, they can follow-up with 5Sum, 6Sum, and so on. What they are really expecting at this point is a kSum solution. Therefore, we will focus on a generalized implementation here.

Approach 1: Two Pointers

The two pointers pattern requires the array to be sorted, so we do that first. Also, it's easier to deal with

For 3Sum, we enumerate each value in a single loop, and use the two pointers pattern for the rest of the

target: 42

array. For kSum, we will have k - 2 nested loops to enumerate all combinations of k - 2 values.

duplicates if the array is sorted: repeated values are next to each other and easy to skip.

**Algorithm** 

2 15 20 25 10 k

We can implement k - 2 loops using a recursion. We will pass the starting point and k as the parameters. When k == 2, we will call twoSum, terminating the recursion. 1. For the main function: Sort the input array nums. • Call kSum with start = 0, k = 4, and target, and return the result. 2. For kSum function:

3. For twoSum function: • Set the low pointer lo to start, and high pointer hi to the last index. While low pointer is smaller than high:

Include the current value nums[i] into set.

■ For each returned set of values:

Return the result res.

Python3

res = []

if k == 2:

return res

return res

return twoSum(nums, target)

if i == 0 or nums[i - 1] != nums[i]:

for i in range(len(nums)):

hi -= 1

lo += 1

hi -= 1

else:

Java

3

4

5 6

7

8

9

10

11 12

13

14

15 16

17

18 19

20 21

22 23

24

25

26

27

**Complexity Analysis** 

class Solution:

Add set to the result res.

· Return the result res.

def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

def kSum(nums: List[int], target: int, k: int) -> List[List[int]]:

if len(nums) == 0 or nums[0] \* k > target or target > nums[-1] \* k:

If the sum is greater than target, decrement hi.

sum = nums[lo] + nums[hi] if sum < target or (lo > 0 and nums[lo] == nums[lo - 1]):

res.append([nums[lo], nums[hi]])

res.append([nums[i]] + set)

approach. In 3Sum: Hash Set, we solved the problem without sorting the array. To do that, we needed to sort values within triplets, and track them in a hash set. Doing the same for k values could be impractical. So, for this approach, we will also sort the array and skip duplicates the same way as in the Two Pointers approach above. Thus, the code will only differ in the twoSum implementation.

def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

def kSum(nums: List[int], target: int, k: int) -> List[List[int]]:

if len(nums) == 0 or nums[0] \* k > target or target > nums[-1] \* k:

### 22 s.add(nums[i]) 23 return res 24 25 nums.sort()

Rate this article: \* \* \* \* \*

• Time Complexity:  $\mathcal{O}(n^{k-1})$ , or  $\mathcal{O}(n^3)$  for 4Sum. We have k-2 loops iterating over n elements, and

space? The idea is to compute a + b + c + d, we can precompute all the sum of a + b and store into a map first. We then can do c + d and checking if target - (c + d) is in the map. For a general k-Sum, this could be done in  $O(n^{(k/2)})$  time and  $O(n^{(k/2)})$  space. This code pass all the tests. The sort needed on each result can push an additional k\*logk factor to runtime, i.e. final runtime could be O(n^(k/2) \* k \* Read More

Reputation Reply subhanjansaha 🖈 1 🕗 June 17, 2020 6:35 PM what a grim problem 

Read More 

Following a similar logic, we can implement 4Sum by wrapping 3Sum in another loop. But wait - there is a

M

• Check if the sum of k smallest values is greater than target, or the sum of k largest values is smaller than target . Since the array is sorted, the smallest value is nums[start], and largest the last element in nums. ■ If so, no need to continue - there are no k elements that sum to target. o If k equals 2, call twoSum and return the result. • Iterate i through the array from start: ■ If the current value is the same as the one before, skip it. Recursively call kSum with start = i + 1, k = k - 1, and target - nums[i].

Otherwise, we found a pair: Add it to the result res. Decrement hi and increment lo.

for \_, set in enumerate(kSum(nums[i + 1:], target - nums[i], k - 1)):

Copy

Copy

• If the sum of nums[10] and nums[hi] is less than target, increment 10.

Also increment 10 if the value is the same as for 10 - 1.

Also decrement hi if the value is the same as for hi + 1.

def twoSum(nums: List[int], target: int) -> List[List[int]]: res = []lo, hi = 0, len(nums) - 1while (lo < hi):

elif sum > target or (hi < len(nums) - 1 and nums[hi] == nums[hi + 1]):

Note that for k > 2, sorting the array does not change the overall time complexity. • Space Complexity:  $\mathcal{O}(n)$ . We need  $\mathcal{O}(k)$  space for the recursion. k can be the same as n in the worst case for the generalized algorithm. Note that, for the purpose of complexity analysis, we ignore the memory required for the output. Approach 2: Hash Set Since elements must sum up to the exact target value, we can also use the Two Sum: One-pass Hash Table

twoSum implementation here is almost the same as in Two Sum: One-pass Hash Table. The only difference is

the check to avoid duplicates. Since the array is sorted, we can just compare the found pair with the last one

for \_, set in enumerate(kSum(nums[i + 1:], target - nums[i], k - 1)):

• Time Complexity:  $\mathcal{O}(n^{k-1})$ , or  $\mathcal{O}(n^3)$  for 4Sum. We have k-2 loops, and twoSum is  $\mathcal{O}(n)$ .

## 11 12 13 return res 14 15 def twoSum(nums: List[int], target: int) -> List[List[int]]:

**Complexity Analysis** 

twoSum is  $\mathcal{O}(n)$ .

**Algorithm** 

C++

1

2 3

4

5

6

7

8

9 10

16

17

18 19

20

21

26

in the result res.

Java

class Solution:

Python3

return []

return twoSum(nums, target)

if i == 0 or nums[i - 1] != nums[i]:

res.append([nums[i]] + set)

if len(res) == 0 or res[-1][1] != nums[i]:

res.append([target - nums[i], nums[i]])

if target - nums[i] in s:

for i in range(len(nums)):

for i in range(len(nums)):

return kSum(nums, target, 4)

svella 🛊 2 🗿 June 17, 2020 6:02 AM

if k == 2:

res = []

res = []

s = set()

ullet Space Complexity:  $\mathcal{O}(n)$  for the hash set. The space needed for the recursion will not exceed  $\mathcal{O}(n)$ .

Note that for k>2, sorting the array does not change the overall time complexity.

O Previous Next 👀 Comments: 6 Sort By ▼ Type comment here... (Markdown is supported) Preview Post dryadd44651 \* 24 • June 19, 2020 8:34 AM what's next? 5 sum? 11 A V C Share Reply **SHOW 2 REPLIES** hohaidang \* 7 ② June 16, 2020 10:44 AM I think this should be hard problem, especially if you built a generic program for 3, 4, 5... SUM 13 A V C Share Reply SHOW 1 REPLY luannv \* 12 ② June 9, 2020 11:21 PM How about a solution with better runtime at the cost of extra space. e.g. O(n^2) time and O(n^2)

**SHOW 2 REPLIES** Rahul-Chauhan21 \* 0 \* June 24, 2020 5:14 PM A simple improvement to reduce the running time is for (int i = start; i < nums.size() - k + 1; ++i) // before: for(int i = start; i < nums.size(); ++i) if (i == start || nums[i - 1] != nums[i]) for (auto &set: kSum(nums, target - nums[i], i + 1, k - 1)) { res.push back({nums[i]}); Read More A Report

Approach 1 algorithm doesn't match implementation in several respects. Check if the sum of k smallest values is greater than target, or the sum of k largest values is smaller than target. but the implementation is multiplying the lowest and highest single values by four instead of summing