3Sum

3Sum

Medium

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` where `nums[i] + nums[j] + nums[k] == 0`, and the indices `i`, `j` and `k` are all distinct.

The output should *not* contain any duplicate triplets. You may return the output and the triplets in any order.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]
```

Copy

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]`.

Example 2:

```
Input: nums = [0,1,1]

Output: []
```

Copy

Explanation: The only possible triplet does not sum up to 0.

Example 3:

```
Input: nums = [0,0,0]

Output: [[0,0,0]]
```

Copy

Explanation: The only possible triplet sums up to 0.

Constraints:

`3 <= nums.length <= 1000-10^5 <= nums[i] <= 10^5`

- `3 <= nums.length <= 1000`
- `10^5 <= nums[i] <= 10^5`

Initial thought :

Input array can be sorted into increasing order

[j] and [k] pairs can be sorted such that when we calculate [j] + [k] we check to see the reciprocal of the equating number so maybe -1 + 0 = -1, we check to see if there is a -1 that is not the pointed index

Using a 2 pointer

So final thought, while we iterate through a sorted array of index i, we use a left and right point for every i.

the sorted array helps removing duplicates

```
from typing import List

class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # Sort the input list
        sort_nums = sorted(nums)
        result = []

        # Iterate through the sorted list
        for i in range(len(sort_nums) - 2):
            # Skip duplicate elements
            if i > 0 and sort_nums[i] == sort_nums[i - 1]:
                continue

            # Two-pointer approach
            j, k = i + 1, len(sort_nums) - 1
            while j < k:
                target = sort_nums[i] + sort_nums[j] + sort_nums
                if target > 0:
                    k -= 1
                elif target < 0:
                    j += 1
                else:
                    result.append([sort_nums[i], sort_nums[j], s
                    # Move pointers to avoid duplicates
                    while j < k and sort_nums[j] == sort_nums[j
                        j += 1
                    while j < k and sort_nums[k] == sort_nums[k
                        k -= 1
                    j += 1
                    k -= 1

        return result
```

break down of the code and reasoning

```
sort_nums = sorted(nums)
        result = []
```

Using the inbuilt python sort, this to eliminate duplicates, this is because repeated elements will just be next to each other and you can easily just not include them

```
   for i in range(len(sort_nums) - 2):
            # Skip duplicate elements
            if i > 0 and sort_nums[i] == sort_nums[i - 1]:
                continue
```

```
 j, k = i + 1, len(sort_nums) - 1
```

used to initialise the two pointers

```
while j < k:
                target = sort_nums[i] + sort_nums[j] + sort_nums
                if target > 0:
                    k -= 1
                elif target < 0:
                    j += 1
                else:
                    result.append([sort_nums[i], sort_nums[j], s
                    # Move pointers to avoid duplicates
                    while j < k and sort_nums[j] == sort_nums[j
                        j += 1
                    while j < k and sort_nums[k] == sort_nums[k
                        k -= 1
```

```
                    j += 1
                    k -= 1

        return result
```

On this two pointer approach, j starts after i, and k starts at the list

the target sum guides the pointr movement, k moves left if its too large and j moves right if the sum is too small

two pointer :

if the sum is too large, theres no way incrementing the left pointer would work, as it would still make it too large

just like if the sum is too small, decrementing the right pointer wouldn’t work as it still makes it too small

two pointer is logical and deterministic