3Sum

Difficulty	Medium
Category	Two Pointers
Question	https://leetcode.com/problems/3sum/
Solution	https://youtu.be/jzZsG8n2R9A
Status	Done

Question

> 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

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.
```

Idea

💡 Sort input, for each first element, find next two where -a = b+c, if a=prev a, skip a, if b=prev b skip b to delete duplicates; to find b,c use two pointers, left/right on remaining list;

Solution

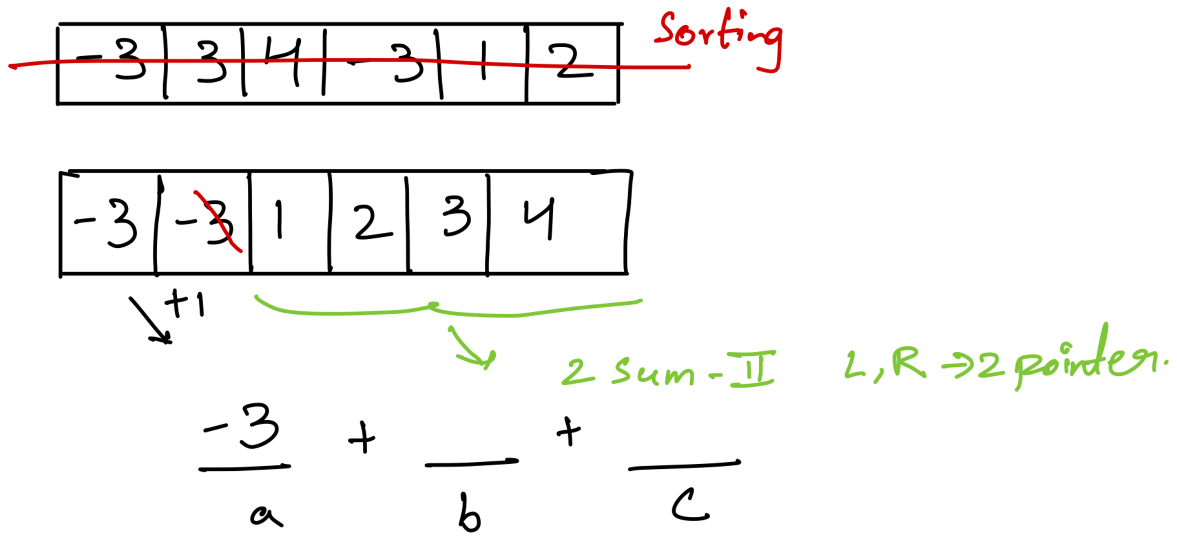
```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # Initialize an empty list to store the resulting triplets
        res = []

        # Sort the input list in ascending order
        nums.sort()

        # Loop through the input list
        for i, a in enumerate(nums):
            # Skip duplicate values to avoid duplicate triplets
            if i > 0 and a == nums[i - 1]:
                continue

            # Initialize two pointers, l and r, to find the other two elements
            l, r = i + 1, len(nums) - 1

            while l < r:
                # Calculate the sum of three elements
                threeSum = a + nums[l] + nums[r]

                # Check if the sum is greater than 0
                if threeSum > 0:
                    r -= 1
                # Check if the sum is less than 0
                elif threeSum < 0:
                    l += 1
                # If the sum is 0, we found a triplet
                else:
                    res.append([a, nums[l], nums[r]])
                    l += 1
                    # Skip duplicate values to avoid duplicate triplets
                    while nums[l] == nums[l - 1] and l < r:
                        l += 1

        return res
```

Explanation

1. When the sum of `a`, `nums[l]`, and `nums[r]` is 0, it means we have found a valid triplet, so we add it to the `res` list.
2. After adding the triplet to the result list, we increment the left pointer `l` by 1. This is done because we want to continue searching for other valid triplets, and moving `l` to the right ensures that we are considering different combinations.
3. To avoid adding duplicate triplets to the result, we enter a while loop. This loop checks if the current value at `nums[l]` is the same as the previous value at `nums[l-1]`, and if `l` is less than `r` (to ensure we don't go out of bounds).
4. Inside the loop, we keep incrementing `l` until we find a different value. This effectively skips over duplicate values, ensuring that we don't create duplicate triplets in the result list.

Time and Space Complexity

(*) 3Sum :-



~~-3 | 3 | 4 | -3 | 1 | 2~~
Sorting
-3 | ~~-3~~ | 1 | 2 | 3 | 4
+1
2 sum - II   L, R → 2 pointer.
-3 + ___ + ___
a b c