LeetCode 15. 3Sum

1. Problem Title & Link

- **15. 3Sum**
- https://leetcode.com/problems/3sum/

2. Problem Statement (Short Summary)

Given an integer array nums, return all **unique triplets** [nums[i], nums[j], nums[k]] such that:
i != j != k and nums[i] + nums[j] + nums[k] == 0
The solution set must **not contain duplicate triplets**.

3. Examples (Input → Output)

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

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

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

4. Constraints

- 3 <= nums.length <= 3000
- -10^5 <= nums[i] <= 10^5

5. Thought Process (Step by Step)

We need **three numbers** that sum to zero.

Brute force would be $O(n^3)$ — but we can do it in **$O(n^2)$** using sorting + two pointers 💡

Step 1: Sort the Array

Sorting allows us to use two-pointer technique efficiently and handle duplicates easily.

Step 2: Fix One Element, Find the Other Two

For each index i (0 to n−3):

- Let the target = -nums[i]
- Use two pointers:
  - left = i + 1
  - right = n − 1
- Move pointers inward based on sum comparison:
  - If nums[i] + nums[left] + nums[right] == 0 → valid triplet
  - If sum < 0 → move left++
  - If sum > 0 → move right--

Step 3: Skip Duplicates

After finding one valid triplet, skip over repeated numbers for both left and right to avoid duplicates.

6. Pseudocode

```
sort(nums)
result = []

for i in 0..n-3:
  if i > 0 and nums[i] == nums[i-1]:
    continue  # skip duplicate

  left = i + 1
  right = n - 1

  while left < right:
    s = nums[i] + nums[left] + nums[right]
    if s == 0:
      result.append([nums[i], nums[left], nums[right]])
      while left < right and nums[left] == nums[left+1]:
        left++
      while left < right and nums[right] == nums[right-1]:
        right--
      left++
      right--
    elif s < 0:
      left++
    else:
      right--
return result
```

7. Code Implementation

✅ **Python**

```
class Solution:
  def threeSum(self, nums: List[int]) -> List[List[int]]:
    nums.sort()
    res = []
    n = len(nums)

    for i in range(n - 2):
      if i > 0 and nums[i] == nums[i - 1]:
        continue

      left, right = i + 1, n - 1

      while left < right:
        total = nums[i] + nums[left] + nums[right]
        if total == 0:
          res.append([nums[i], nums[left], nums[right]])
          while left < right and nums[left] == nums[left + 1]:
            left += 1
```

```
            while left < right and nums[right] == nums[right - 1]:
                right -= 1
            left += 1
            right -= 1
        elif total < 0:
            left += 1
        else:
            right -= 1
    return res
```

✅ **Java**

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();

        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            int left = i + 1, right = nums.length - 1;

            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    res.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right--;
                    left++;
                    right--;
                } else if (sum < 0) left++;
                else right--;
            }
        }
        return res;
    }
}
```

8. Time & Space Complexity

- **Time:** $O(n^2)$
- **Space:** O(1) (excluding output list)

9. Dry Run (Step-by-Step Execution)

👉 Input: nums = [-1,0,1,2,-1,-4]

After sorting: [-4, -1, -1, 0, 1, 2]

i	nums[i]	left	right	sum	Action	Result
0	-4	1	5	-3	sum < 0 → left++	-
0	-4	2	5	-3	left++	-
0	-4	3	5	-2	left++	-
0	-4	4	5	-1	left++	-
1	-1	2	5	0	✅ add [-1,-1,2]	[[-1,-1,2]]
1	-1	3	4	0	✅ add [-1,0,1]	[[-1,-1,2],[-1,0,1]]
2	-1	skip duplicate				
3	0	4	5	3	right--	

✅ Output: [[-1,-1,2], [-1,0,1]]

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Two-Pointer after Sorting (3-way combination) — fix one, use two pointers for the other two.	- K-Sum family (2Sum, 3Sum, 4Sum) - Pair & triplet search - Avoiding duplicates	- Forgetting to skip duplicates - Wrong pointer movement - Not sorting first	🔷 Builds to **LeetCode 18 (4Sum)** 🔷 Connects with **Two Sum II (LC 167)** 🔷 Reinforces “fix-one + shrink window” pattern

11. Common Mistakes / Edge Cases

- Not skipping duplicates for both i, left, and right.
- Forgetting to sort first (breaks logic).
- Using hash sets unnecessarily ($O(n^3)$ logic).

12. Variations / Follow-Ups

- **4Sum (LC 18)** → extend to 4 pointers.
- **3Sum Closest (LC 16)** → find closest sum instead of exact 0.
- **K-Sum (generalized)** → recursive version using the same principle.