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 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.

**Constraints:**

* 3 <= nums.length <= 3000
* -105 <= nums[i] <= 105

Problem is about finding sum of triples whos sum is zero

So my intuition behind is to choose a element in the array and find two sum of two other element and then if triples equal to zero then we got the ans

Key observation and problem requirement are

1. No duplicate elements and same goes to triplets solution
2. No indexes so we can sort the nums array
3. Handling Duplication in both n1 and n2 , n3 computation
4. First loop goes to n-3 iterations

To solve this problem we use two pointers approach

Here is the code

import java.util.\*;

class Solution {

    List<List<Integer>> ans = new ArrayList<>();

    public List<List<Integer>> threeSum(int[] nums) {

        int n = nums.length;

        if (n < 3) {

            return new ArrayList<>();

        }

        ans.clear();

        Arrays.sort(nums);

        for (int i = 0; i < n - 2; i++) {

            if (i > 0 && nums[i] == nums[i - 1]) {

                continue;

            }

            int target = -nums[i];

            twosum(nums, i + 1, n - 1, target, nums[i]);

        }

        return ans;

    }

    public void twosum(int[] nums, int i, int j, int target, int fn) {

        while (i < j) {

            if (nums[i] + nums[j] > target) {

                j--;

            } else if (nums[i] + nums[j] < target) {

                i++;

            } else {

                ans.add(Arrays.asList(fn, nums[i], nums[j]));

                while (i < j && nums[i] == nums[i + 1])

                    i++;

                while (i < j && nums[j] == nums[j - 1])

                    j--;

                i++;

                j--;

            }

        }

    }

}

Same but modified

class Solution {

public List<List<Integer>> threeSum(int[] nums) {

List<List<Integer>> res = new ArrayList<>();

Arrays.sort(nums);

for (int i = 0; i < nums.length; i++) {

if (i > 0 && nums[i] == nums[i-1]) {

continue;

}

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

int total = nums[i] + nums[j] + nums[k];

if (total > 0) {

k--;

} else if (total < 0) {

j++;

} else {

res.add(Arrays.asList(nums[i], nums[j], nums[k]));

j++;

while (nums[j] == nums[j-1] && j < k) {

j++;

}

}

}

}

return res;

}

}