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**
- **-10^5 <= nums[i] <= 10^5**

Solution:

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> result = new LinkedList<>();
       for(int i=0;i<nums.length-2;i++){
          if(i==0 ||(i>0 && nums[i]!=nums[i-1])){
             int low=i+1;
             int high=nums.length-1;
             int sum=0-nums[i];
             while(low<high){
                if(nums[low]+nums[high]==sum){
                   result.add(Arrays.asList(nums[i],nums[low],nums[high]));
                   while(low<high && nums[low]==nums[low+1])    low++;
                   while(low<high && nums[high]==nums[high-1])  high--;
                   low++;
                   high--;
                }
                else if(nums[low]+nums[high]<sum){
                   low++;
                }
                else{
                   high--;
                }
             }
          }
       }
       return result;
    }
}
```