/\*

\* Time complexity : O(n^2)

\* Space complexity : O(n)

\* use hashSet to store every set of result

\* \*/

public List<List<Integer>> threeSum1(int[] nums) {

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

if(nums.length < 3)

return res;

HashSet<ArrayList<Integer>> hashSet = new HashSet<>();

Arrays.sort(nums); // first, sort the array

for(int i = 0; i <= nums.length - 3; i++){

int low = i + 1; // set a pointer 'low', pointing to the next value of nums[i]

int high = nums.length - 1; // set 'high' pointer, pointing to the last nums[i]

while (low < high){

int sum = nums[i] + nums[low] + nums[high];

if(sum == 0){

ArrayList<Integer> curr = new ArrayList<>(); // use the new ArrayList to store the result

curr.add(nums[i]);

curr.add(nums[low]);

curr.add(nums[high]);

// check hashSet every turn, if hashSet doesn't contain this result array

// then add this result array to final res array

if(!hashSet.contains(curr)){

hashSet.add(curr);

res.add(curr);

}

// move two pointers

low++;

high--;

}else if(sum > 0) // if sum of three numbers is bigger than target, then move high pointer

high--;

else // if sum of three numbers is smaller than target, then move low pointer

low++;

}

}

return res;

}

/\*

\* Time complexity : O(n^2)

\* Space complexity : O(1)

\* \*/

public List<List<Integer>> threeSum2(int[] nums) {

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

if(nums.length < 3)

return res;

Arrays.sort(nums); // first, sort the array

for(int i = 0; i <= nums.length - 3; i++){

if(i == 0 || nums[i] != nums[i-1]){ // check if current value is equal to the previous one

int low = i + 1;

int high = nums.length - 1;

while(low < high){

int sum = nums[i] + nums[low] + nums[high];

if(sum == 0){

ArrayList<Integer> curr = new ArrayList<>(); // use the new ArrayList to store the result

curr.add(nums[i]);

curr.add(nums[low]);

curr.add(nums[high]);

res.add(curr);

low++;

high--;

// check if current value is equal to the previous one

while (low < high && nums[low] == nums[low-1])

low++;

// check if current value is equal to the previous one

while (low < high && nums[high] == nums[high+1])

high--;

}

else if(sum > 0) // if sum of three numbers is bigger than target, then move high pointer

high--;

else // if sum of three numbers is smaller than target, then move low pointer

low++;

}

}

}

return res;

}