**Student:** Yerbolat Pazyl

**Group:** IS-1808K

**ID:** 26244

**Laboratory work #3. Sets and Maps**

Deadline: 22nd September 2020

Solve these problems using Sets or Maps.

**Problem - 1**

<https://leetcode.com/problems/contains-duplicate/>

**217. Contains Duplicate**

**Description**

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

**Example 1:**

**Input:** [1,2,3,1]

**Output:** true

**Solution**

Сreate a HashSet and add all the elements from the nums array. This way the HashSet will only store unique elements. At the end we check the size of the arrays, if they are equal then there is no duplicate and if they are not equal then there is a duplicate.

**Java Code**

class Solution {

public boolean containsDuplicate(int[] nums) {

HashSet<Integer> sets = new HashSet<>();

for(int a : nums)

sets.add(a);

return (sets.size() != nums.length);

}

}

**Problem - 2**

<https://leetcode.com/problems/two-sum/>

**1. Two Sum**

**Description**

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to *target*.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Output:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Solution**

I create a HashMap for storing the array value as a key and the element index as a value. Opening the for loop will run through the array of nums and will check if the element is in map (map.containsKey(target - nums[i])) if there is then we found the answer. Write these two indexes to the result array i and map.get(target - nums[i]). If there is no such key then we add this element to the map.

**Java Code**

class Solution {

public int[] twoSum(int[] nums, int target) {

HashMap<Integer, Integer> map = new HashMap<>();

int [] result = new int [2];

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

if(map.containsKey(target - nums[i])) {

result[0] = i;

result[1] = map.get(target - nums[i]);

break;

} else {

map.put(nums[i], i);

}

}

return result;

}

}

**Problem – 3**

<https://leetcode.com/problems/intersection-of-two-arrays/>

**349. Intersection of Two Arrays**

**Description**

Given two arrays, write a function to compute their intersection.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2]

**Solution**

Сreating two Hashsets for two arrays nums1 and nums2. HashSet stores only unique elements. Removing duplicate elements from nums1 and nums2.

After that, create an array res with the size set 1. size(). using the shortened for loop, we find the intersection.

The res array may not be fully. Do fully array Arrays.copyOf(res, i).

**Java Code**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

HashSet<Integer> set1 = new HashSet<Integer>();

HashSet<Integer> set2 = new HashSet<Integer>();

for(int a : nums1)

set1.add(a);

for(int a : nums2)

set2.add(a);

int [] res = new int[set1.size()];

int i = 0;

for (Integer a : set1)

if (set2.contains(a))

res[i++] = a;

return Arrays.copyOf(res, i);

}

}

**Problem - 4**

<https://leetcode.com/problems/group-anagrams/>

**49. Group Anagrams**

**Description**

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Solution**

Creating the LinkedHashMap key type String and the type value List<String>. Also create a list<List<String>> result collection. Сreate a sort function that sorts the string. Open the loop and run through the strs array. The algorithm is this take a string sort and check whether there is such a key in map if there is then add str to the value. If not create a new key sortStr with the value str. After that we take the weight map value and put it in result. Return result.

**Java Code**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

LinkedHashMap<String, List<String>> map = new LinkedHashMap<>();

List<List<String>> result = new ArrayList<>();

for(String str : strs){

String sortStr = sort(str);

if(map.containsKey(sortStr)) {

map.get(sortStr).add(str);

} else {

map.put(sortStr, new ArrayList<>());

map.get(sortStr).add(str);

}

}

for(List<String> a : map.values())

result.add(a);

return result;

}

public String sort(String str) {

char tempArray[] = str.toCharArray();

Arrays.sort(tempArray);

return new String(tempArray);

}

}

**Problem - 5**

<https://leetcode.com/problems/word-pattern/>

**290. Word Pattern**

**Description**

Given a pattern and a string s, find if s follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** word in s.

**Example 1:**

**Input:** pattern = "abba", s = "dog cat cat dog"

**Output:** true

**Solution**

Creating a HashMap (map) with the character key type and a value with the String type. Dividing the string by spaces (Do split) and store it in the array strs. Creating a variable n to store the size of the strs array. The very first check is checking whether the length of pattern is equal to the length of the array strs. If not equal the answer is false.

Open the loop and take the letters pattern in turn:

**1.** Check whether there is such a key in map. If there is it, take the value of this key and check with the value of strs[i]. If is equal then all is well, continue. If is not equal answer will be false;

**2.** If there is no such key we check whether there is such a value in other keys. If such a value exists in a different key the answer is false. If it doesn't exist then add a new key with the value strs[i].

**Java Code**

class Solution {

public boolean wordPattern(String pattern, String s) {

HashMap<Character, String> map = new HashMap<>();

String [] strs = s.split(" ");

int n = strs.length;

if(strs.length != pattern.length())

return false;

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

if(map.containsKey(pattern.charAt(i))) {

if(!map.get(pattern.charAt(i)).equals(strs[i])) {

return false;

}

} else {

if(!map.containsValue(strs[i])) {

map.put(pattern.charAt(i), strs[i]);

} else {

return false;

}

}

}

return true;

}

}

**Problem – 6**

<https://leetcode.com/problems/3sum/>

**15. 3Sum**

**Description**

Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

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]]

**Solution**

First, sort the names array and create a set (Set<List<Integer>>). Opening the for loop starts from 0 to nums.length - 2. Inside the loop, create two variables j and k.

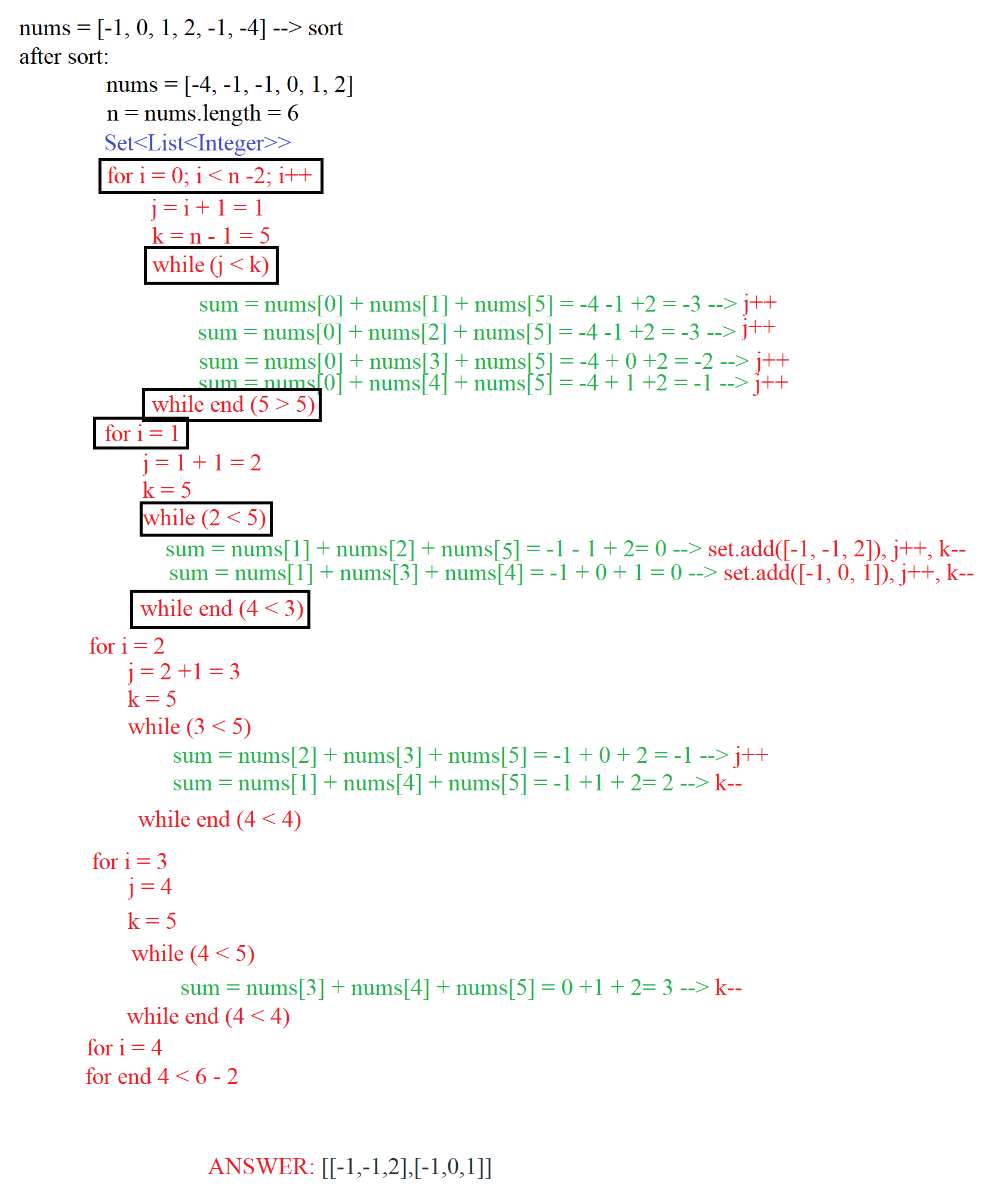
j = i + 1, k = nums.length -1. And open the while loop with the condition j < k. and inside we create the variable sum (sum = nums[i] + nums[j] + nums[k]). And check on the conditions:

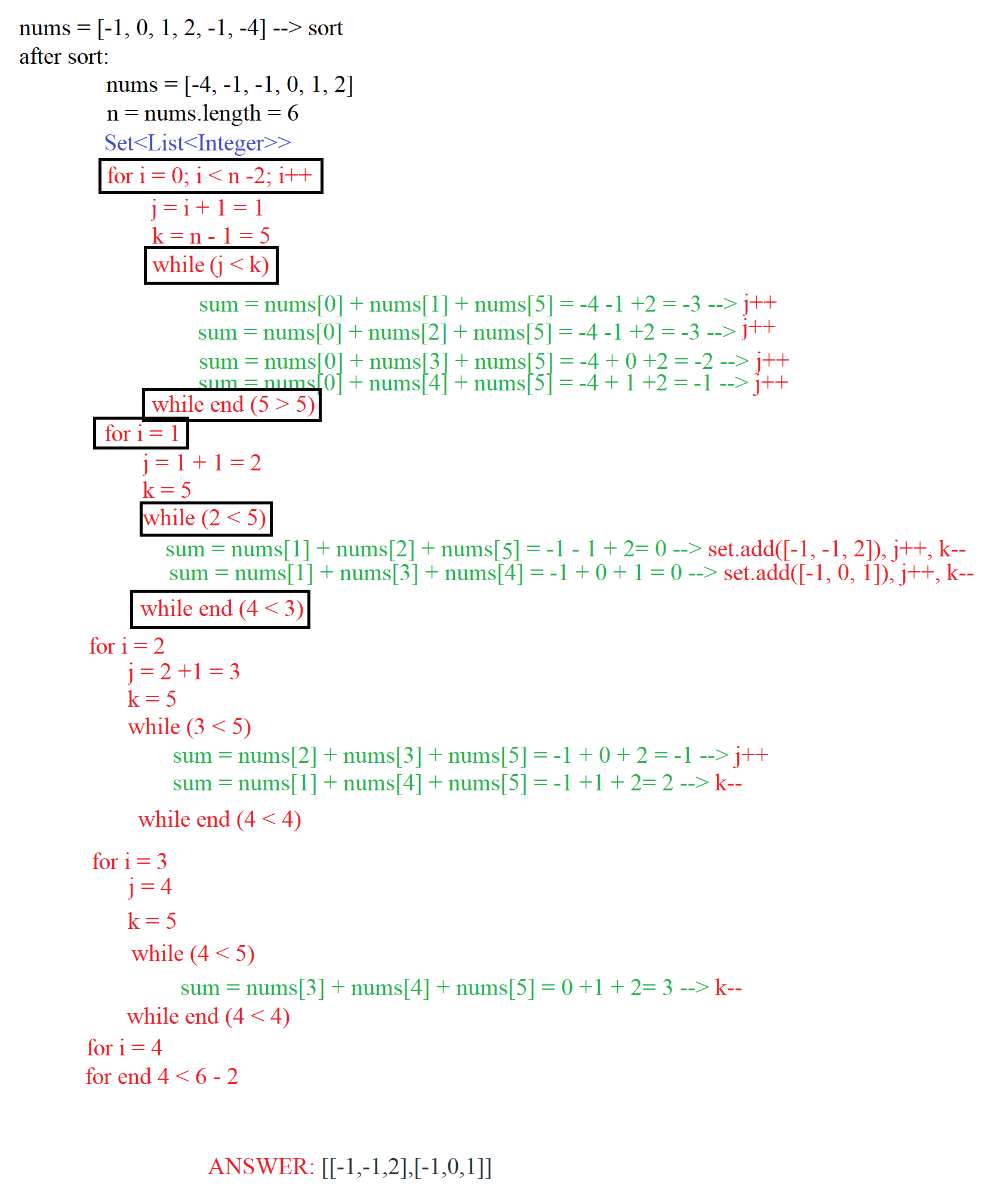
1. if sum == 0 🡪 set.add(Arrays.asList(nums[i], nums[j++], nums[k--])).

2. else if sum < 0 🡪 j++

3. else (sum > 0) k- -

So we find all sub arrays.



**Java Code**

class Solution {

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

if (nums.length < 3)

return new ArrayList<>();

Arrays.sort(nums);

int n = nums.length;

Set<List<Integer>> set = new HashSet<>();

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

int j = i + 1;

int k = n - 1;

while (j < k) {

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

if (sum == 0)

set.add(Arrays.asList(nums[i], nums[j++], nums[k--]));

else if (sum < 0)

j++;

else

k--;

}

}

return new ArrayList<>(set);

}

}

**Problem - 7**

<https://leetcode.com/problems/contains-duplicate-ii/>

**219. Contains Duplicate II**

**Description**

Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that **nums[i] = nums[j]** and the **absolute** difference between *i* and *j* is at most *k*.

**Example 1:**

**Input:** nums = [1,2,3,1], k = 3

**Output:** true

**Solution**

Сreating a HashMap where the key is the value from the nums array and the value is the index of this element. Open the for loop take an element from the array nums and check whether there is such a key in map if there is then do the operation Math. abs(map. get(nums[i]) - i) < = k if it is true then we found the answer if the operation is not true then we do overwrite the map key map.put(nums[i], i). If this element is not in map keys then add (map.put(nums[i], i)).

**Java Code**

class Solution {

public boolean containsNearbyDuplicate(int[] nums, int k) {

HashMap<Integer, Integer> map = new HashMap<>();

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

if (map.containsKey(nums[i])) {

if ( Math.abs(map.get(nums[i]) - i) <= k ) {

return true;

} else {

map.put(nums[i], i);

}

} else {

map.put(nums[i], i);

}

}

return false;

}

}

**Problem – 8**

<https://leetcode.com/problems/random-pick-index/>

**398. Random Pick Index**

**Description**

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

**Note:**  
The array size can be very large. Solution that uses too much extra space will not pass the judge.

**Example:**

int[] nums = new int[] {1,2,3,3,3};

Solution solution = new Solution(nums);

// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.

solution.pick(3);

// pick(1) should return 0. Since in the array only nums[0] is equal to 1.

solution.pick(1);

**Solution**

Creating a new variable HashMap <Integer, List<Integer> > map = new HashMap<>(). In the constructor, first find out the size of the nums array. Open the loop and insert data from the nums array, the key will be the value of the array element and the key value will be the index of the element from the nums array.

1. if such a key already exists then we do map.get(nums[i]).add(i).

2. if there is no such key we do:

map.put(nums[i], new ArrayList<>());

map.get(nums[i]).add(i);

In the pick method we find out the size of the collection with the target key and create a Random object to randomly extract values from the collection. And return map.get(target).get(random.nextInt(n)).

**Java Code**

class Solution {

HashMap<Integer, List<Integer>> map = new HashMap<>();

public Solution(int[] nums) {

int n = nums.length;

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

if(!map.containsKey(nums[i])) {

map.put(nums[i], new ArrayList<>());

map.get(nums[i]).add(i);

} else {

map.get(nums[i]).add(i);

}

}

}

public int pick(int target) {

int n = map.get(target).size();

Random random = new Random();

return map.get(target).get(random.nextInt(n));

}

}

**Problem – 9**

<https://leetcode.com/problems/subarray-sum-equals-k/>

**560. Subarray Sum Equals K**

**Description**

Given an array of integers and an integer **k**, you need to find the total number of continuous subarrays whose sum equals to **k**.

**Example 1:**

**Input:**nums = [1,1,1], k = 2

**Output:** 2

**Solution**

We open a HashMap where we will store the segment with the sum of the elements of the nums array (prefix). Adding to map (0, 1) and create variable sum.

Every time:

1. We get current sum and check map.containsKey(sum - k). If it is true, do result += map.get(sum - k).

2.We encounter a new sum, we make a new entry in the hashmap corresponding to that sum. If the same sum occurs again, we increment the count corresponding to that sum in the hashmap.

After the complete array has been traversed, the result gives the required result.

**Java Code**

class Solution {

public int subarraySum(int[] nums, int k) {

int n = nums.length;

int result = 0, sum = 0;

HashMap<Integer, Integer> map = new HashMap<>();

map.put(sum, 1);

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

sum += nums[i];

if (map.containsKey(sum - k)) {

result += map.get(sum - k);

}

map.put(sum, map.getOrDefault(sum, 0) + 1);

}

return result;

}

}

**Problem – 10**

<https://leetcode.com/problems/4sum-ii/>

**454. 4Sum II**

**Description**

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

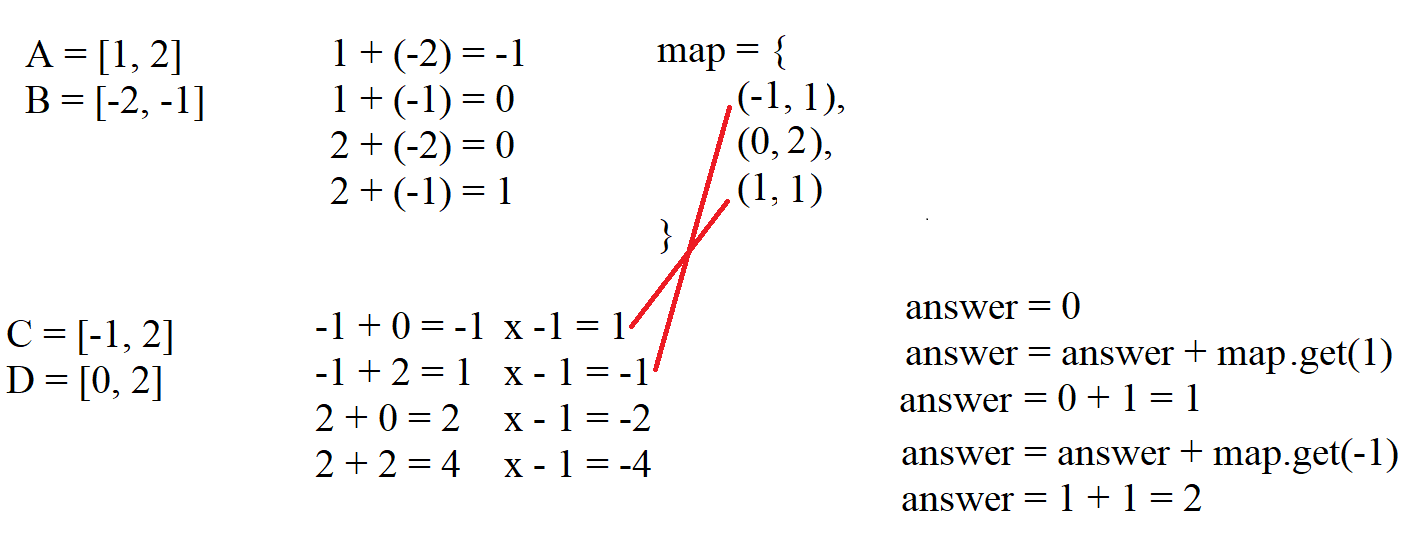
**Solution**

Сreate a HashMap and take the array a1 and b1 iterate through all the options for their sum of elements and put them in the map. If such a sum occurs then we make the increment value (default 1).

Then we find the sum of the array elements C and D. Each time we do:

int sum = (C[i] + D[j]) \* -1;

answer += map.get Or Default(sum \* -1, 0). So we find the answer



**Java Code**

class Solution {

public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

Map<Integer, Integer> map = new HashMap<>();

int answer = 0;

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

for (int j = 0; j < B.length; j++) {

int sum = A[i] + B[j];

map.put(sum, map.getOrDefault(sum, 0) + 1);

}

}

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

for (int j = 0; j < D.length; j++) {

int sum = (C[i] + D[j]) \* -1;

answer += map.getOrDefault(sum, 0);

}

}

return answer;

}

}

**Problem - 11**

<https://leetcode.com/problems/happy-number/>

**202. Happy Number**

**Description**

Write an algorithm to determine if a number n is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1. Those numbers for which this process **ends in 1** are happy numbers.

Return True if n is a happy number, and False if not.

**Example:**

**Input:** 19

**Output:** true

**Solution**

Creating a HashSet to store all values of the variable n. In short, the algorithm works like this: each time it checks whether there are new values of the variable n in the HashSet. If there is an answer it will be a false. If the new value is equal to one then the answer is true

**Java Code**

class Solution {

public boolean isHappy(int n) {

HashSet <Integer> used = new HashSet<>();

used.add(n);

while (n != 1) {

int res = 0;

while (n > 0) {

res += (n % 10) \* (n % 10);

n /= 10;

}

n = res;

if(used.contains(n))

return false;

else

used.add(n);

}

return true;

}

}

**Problem – 12**

<https://leetcode.com/problems/top-k-frequent-elements/>

**347. Top K Frequent Elements**

**Description**

Given a non-empty array of integers, return the ***k*** most frequent elements.

**Example 1:**

**Input:** nums = [1,1,1,2,2,3], k = 2

**Output:** [1,2]

**Solution**

Firstly, to find the most frequent elements, we need to know the number and it’s count. For this we can use HashMap to store the number and it’s count.

Example:

nums = [1, 1, 1, 2, 2, 3]

map = {(1, 3), (2, 2), (3, 1)}

max maxFrequent = 3

Secondly, we need to keep track of maximum frequency, so it’s value in this case would be 3. Then in next step, create a ArrayList. At each index we can store multiple elements. It’s size would be maximum frequency + 1.

Thirdly, based on the frequency of a number, put the number in the appropriate ArrayList(list). There might be more than one numbers with the same frequency, so we have to put them in a same index.

iv) Now, iterate over the bucket elements and print the k most frequent elements.

**Java Code**

class Solution {

public int[] topKFrequent(int[] nums, int k) {

ArrayList<Integer> [] list;

ArrayList<Integer> result = new ArrayList<>();

HashMap<Integer, Integer> map = new HashMap<>();

int [] answer = new int [k];

int maxFrequent = 0;

for (int a : nums) {

int frequent = map.getOrDefault(a, 0) + 1;

map.put(a, frequent);

maxFrequent = Math.max(maxFrequent, frequent);

}

list = new ArrayList[maxFrequent + 1];

for (int key : map.keySet()) {

int frequent = map.get(key);

if (list[frequent] == null)

list[frequent] = new ArrayList<>();

list[frequent].add(key);

}

for(int i = list.length - 1; i >= 0 && k > result.size(); --i) {

if (list[i] != null)

result.addAll(list[i]);

}

for(int i = 0; i < result.size() && k > 0; i++, --k) {

answer[i] = result.get(i);

}

return answer;

}

}