2475. Number of Unequal Triplets in Array

Hash Table Array Easy

# **Problem Description**

out how many groups of three indices (i, j, k) exist such that the following conditions are met: 1. The indices are in strictly increasing order, i.e., i < j < k.

You are provided with an array of positive integers called <a href="nums">nums</a>. Each element of the array has a 0-based index. The task is to find

- 2. The values of nums at these indices are pairwise distinct. This means that nums[i], nums[j], and nums[k] should all be different from each other.
- In other words, for a triplet (i, j, k) to be counted, it must consist of three unique numbers from the nums array, each coming

from a unique position in the array, where i, j, and k represent the positions (indices) of these numbers.

The goal is to return the count of all such triplets.

Intuition

### To solve this problem, we need to find a way to calculate the number of distinct triplets without having to manually check each

We can use the following approach: 1. Count the frequency of each number in the array using a data structure like a dictionary (Counter in Python).

The intuition stems from the fact that if we know how many times a particular number occurs, and how many numbers are before

and after it in the array (that haven't been used in a triplet yet), we can calculate all the valid triplets it can form.

possible combination, which would be inefficient especially for large arrays.

Here are the steps taken in the solution code: • Use Counter to get the frequency of each distinct number in nums.

2. Iterate through each unique number's frequency, calculating how many distinct triplets can be formed with it.

• Initialize two counters, ans and a. ans will hold the final count of triplets, and a will keep track of the count of numbers processed so far. Loop through the frequency count of each distinct number:

 Let b represent the frequency of the current number. ∘ Calculate c as the number of remaining elements that come after the current number, which is n - a - b (where n is the total length of

- nums). • The number of triplets we can form with the current number as the middle element is a \* b \* c. This product comes from selecting one of
- the numbers before the current one (a choices), the current number itself (b choices), and one of the numbers after (c choices). Add this triplet count to the ans.
- Update a to include the current number as well by adding b to it. • The final answer is contained in ans, so we return it.
- Solution Approach

• We use the Counter class from Python's collections module to efficiently count the frequency of each element in the nums array.

The solution approach is based on the concept of counting and combinatorics. To implement the solution, the following components are used:

#### A dictionary-like container maps unique elements to their respective frequencies. Variables:

**Data Structures:** 

 ans: An accumulator to sum the number of valid triplets found. a: Keeps track of the number of elements processed so far (left side of the current element).

- o c: Represents the number of elements that are available to be picked after the current element. Algorithm:
  - 2. Initialize a variable ans to accumulate the total number of valid triplets and a variable a to keep track of the count of numbers already processed.

Update a to include the element just processed by adding b to it.

1. First, we use Counter to get the frequency of each unique number in nums.

cnt = Counter(nums) # cnt will be Counter( $\{1: 2, 2: 2, 3: 1\}$ )

We set n to the total number of elements in nums, which is 5.

■ Update ans by adding the triplet count, ans = ans + 4 to 4.

■ Update a = a + b to 4 (as we've now processed two occurrences of 2).

triplets without having to check each possible combination of three numbers.

# Initialize the answer and the count for the first number in the triplet

count\_third\_number = total\_numbers - count\_first\_number - count\_second\_number

# Calculate the number of triplets where the first, second, and third numbers

answer += count\_first\_number \* count\_second\_number \* count\_third\_number

# Count of the third number in the triplet is total numbers

// Create a map to store the frequency of each number in the array

// Initialize the answer variable to store the count of unequal triplets

// Variable 'prefixCount' is used to keep track of the count of numbers processed so far

// Calculate the count of numbers remaining after excluding the current number

// Update the answer by adding the number of unequal triplets that can be formed

// Update the prefix count by adding the frequency of the current number

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

// 'n' is the total number of elements in the input array

int suffixCount = n - prefixCount - frequency;

answer += prefixCount \* frequency \* suffixCount;

# minus the count first number and count second number

■ Triplet count for 3 is a \* b \* c = 4 \* 1 \* 0 = 0. No additional triplets can be formed.

1. Count the frequency of each unique number using cnt = Counter(nums).

cnt: A Counter object that holds the frequency of each unique element in nums.

• b: The frequency of the current element being considered as the middle element of a triplet.

handle large arrays without incurring a performance penalty typical of brute-force solutions.

3. Iterate over each count b in the values of cnt (since keys represent the unique numbers and values their respective counts): Calculate the number of available elements after the current element (which could potentially be the third element of the triplet) as c =

n - a - b.

o n: The total number of elements in nums.

4. After the loop ends, ans holds the total number of valid triplets, which is returned as the final answer. Each step of the algorithm is designed to avoid checking each triplet individually by leveraging the frequency counts and

positions of numbers to calculate the number of potential triplets directly. This leads to a more efficient implementation that can

• Calculate the number of triplets possible with the current number as the middle element as a \* b \* c and add this to ans.

- **Example Walkthrough** Let's consider an example nums array to illustrate the solution approach:
- From the problem statement, we want to count distinct triplets (i, j, k) such that nums[i], nums[j], and nums[k] are all unique.

## Initialize the variables ans for the total number of valid triplets and a for counting processed numbers. ans = 0

2, 2: 2, 3: 1}.

a = 0

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

from collections import Counter

• First iteration (for number 1):

• c = n - a - b which is 5 - 2 - 2 = 1. There is one element that can come after 2 to form a triplet.

■ Triplet count for 2 is a \* b \* c = 2 \* 2 \* 1 = 4. We can form 4 triplets with 2 as the middle element.

b for number 1 is 2. • c = n - a - b which is 5 - 0 - 2 = 3. There are three elements that can come after the first 1 to form a triplet. ■ Triplet count for 1 is a \* b \* c = 0 \* 2 \* 3 = 0 (since a is 0, no triplets can be formed with 1 as the middle element yet). ■ Update a = a + b to 2 (since we've now processed two occurrences of 1).

Now, we iterate over each count b in the values of cnt. The dictionary cnt has elements along with their frequencies: {1:

 Third iteration (for number 3): b for number 3 is 1. • c = n - a - b which is 5 - 4 - 1 = 0. There are no elements left to form triplets with 3 as the middle element.

To conclude, using this example with nums = [1, 1, 2, 2, 3], the algorithm efficiently found the total number of 4 distinct

After the iteration ends, and holds the total number of valid triplets. For this example, there are 4 valid triplets, and thus we return 4.

ans remains 4.

from collections import Counter

def unequalTriplets(self, nums: List[int]) -> int:

number counts = Counter(nums)

answer = count\_first\_number = 0

# are all different

public int unequalTriplets(int[] nums) {

total\_numbers = len(nums)

# Count the frequency of each number in nums

# Iterating through the counts of each unique number

for count second number in number counts.values():

**Python** 

Java

**}**;

**TypeScript** 

class Solution {

class Solution:

Second iteration (for number 2):

b for number 2 is 2.

Solution Implementation

# Increment count first number for the next iteration of the loop count\_first\_number += count\_second\_number # Return the total number of unequal triplets return answer

// If the number is already in the map, increment its frequency, // otherwise insert it with frequency 1 frequencyMap.merge(num, 1, Integer::sum);

int answer = 0;

int prefixCount = 0;

int n = nums.length;

// Iterate through the frequency map

prefixCount += frequency;

function unequalTriplets(nums: number[]): number {

const countMap = new Map<number, number>();

// Initialize variable to hold the result

// Update the accumulated count

accumulated += currentCount;

total\_numbers = len(nums)

answer = count first number = 0

# are all different

return answer

for (const currentCount of countMap.values()) {

// Initialize a map to keep count of each number's occurrences

// Iterate over the map to calculate the answer using the formula

// Calculate the count for the third element in the triplet

const remaining = lengthOfNums - accumulated - currentCount;

// Calculate the number of unequal triplets for the current iteration

// Count the occurrences of each number in the nums array

countMap.set(num, (countMap.get(num) ?? 0) + 1);

result += accumulated \* currentCount \* remaining;

# Iterating through the counts of each unique number

for count second number in number counts.values():

count\_first\_number += count\_second\_number

# Return the total number of unequal triplets

// Find the length of the nums array

const lengthOfNums = nums.length;

for (const num of nums) {

let result = 0;

let accumulated = 0;

for (int frequency : frequencyMap.values()) {

for (int num : nums) {

```
// Return the final count of unequal triplets
        return answer;
C++
#include <vector>
#include <unordered map>
using namespace std;
class Solution {
public:
    // This function counts the number of unequal triplets within the input vector.
    int unequalTriplets(vector<int>& nums) {
        // Create a hash table to keep track of the count of each number in the vector.
        unordered map<int, int> numCounts;
        // Increment the count for each value in the nums vector.
        for (int value : nums) {
            ++numCounts[value];
        int result = 0;  // Variable to store the result.
        int accumulated = 0; // Accumulated counts of previous numbers.
        // Calculate the number of unequal triplets.
        for (auto& [value, count] : numCounts) {
            int remaining = nums.size() - accumulated - count; // Count of numbers that are not equal to current number.
            result += accumulated * count * remaining: // Multiply by count to form unequal triplets.
            accumulated += count; // Update accumulated with count of the current number.
        // Return the total number of unequal triplets.
        return result;
```

#### // Return the total number of unequal triplets return result; from collections import Counter class Solution: def unequalTriplets(self, nums: List[int]) -> int: # Count the frequency of each number in nums number counts = Counter(nums)

# Initialize the answer and the count for the first number in the triplet

count\_third\_number = total\_numbers - count\_first\_number - count\_second\_number

# Calculate the number of triplets where the first, second, and third numbers

answer += count\_first\_number \* count\_second\_number \* count\_third\_number

# Increment count first number for the next iteration of the loop

# Count of the third number in the triplet is total numbers

# minus the count first number and count second number

// 'accumulated' will keep track of the accumulated counts for each processed number

Time and Space Complexity **Time Complexity** The time complexity of the code is primarily determined by the number of operations within the for loop which iterates over the values of the Counter object cnt. The Counter object itself is created by iterating over the nums list once, which takes O(n) time where n is the number of elements in nums.

of unique elements in nums. Since k can vary from 1 to n, in the worst case where all elements are unique, k is equal to n. Therefore, the total time complexity is O(n) + O(k) = O(n) as the first iteration to create Counter and the second iteration over unique values can be bounded by n.

Inside the for loop, each operation is constant time, so the overall time complexity of the for loop is 0(k), where k is the number

**Space Complexity** 

The space complexity is affected by the storage used for the Counter object cnt which stores the frequency of each unique element in nums. In the worst case, if all elements of nums are unique, the Counter would take O(n) space where n is the number of elements in **nums**.

Thus, the overall space complexity of the code is O(n).