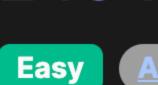
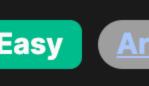
# 2404. Most Frequent Even Element





Hash Table



The problem requires us to find the most frequently occurring even element in an array of integers. The steps are as follows:

1. Traverse through the list of integers.

**Problem Description** 

- 2. Count the frequency of each even integer. 3. Determine the even integer that appears most frequently.

array does not contain any even numbers, we should return -1. These specifics need to be taken into account while implementing the solution. Intuition

If there is a tie (more than one even number with the highest frequency), we should return the smaller of those numbers. If the

### To solve this problem, we need to think about how to efficiently count the occurrences of each even number and then determine

1. Filter out all the even numbers. 2. Count the occurrences of each even number.

which one is the most frequent or the smallest among the most frequent. The steps include:

- 3. Track the most frequent even number.
- 4. Handle the tie-breaking condition by comparing the current number with the most frequent one found so far, updating with the smaller number
- if their frequencies are the same.
- 5. Return -1 if there are no even numbers.
- Solution Approach The provided solution leverages the Counter class from Python's collections module to count the frequency of the even

is the required output for this case.

or use additional data structures.

### numbers. Here is how the solution is implemented: Filtering Even Numbers: A generator expression is used to filter out even numbers from the nums list: (x for x in nums if x

% 2 == 0). This ensures that the Counter only processes even numbers. Counting Frequencies: The Counter is populated with the filtered even numbers, resulting in a dictionary-like object that

- maps each even number to its count of occurrences: Counter(x for x in nums if x % 2 == 0).
- **Iterating Through Counts**: The solution iterates over the items of the Counter with for x, v in cnt.items():, where x is the number and v is its count (frequency).
- number and its corresponding count. During iteration, if the current count v exceeds the maximum count mx, or if the count equals mx but the current number x is smaller than ans, then ans is updated to x and mx is updated to v.

Finding the Most Frequent and Smallest: The variables ans and mx are maintained to keep track of the most frequent even

Result: The value of ans at the end of the iteration is the desired result. If no even numbers were found, ans remains -1, which

the additional loop to find the most frequent even number runs in linear time, O(n), where n is the size of the filtered list of even numbers. Thus, the overall time complexity of the algorithm is O(n), where n is the length of the input list.

designing the conditionals within the iteration loop, the algorithm efficiently solves the problem without needing to sort the input

The pattern used here is quite common in problems related to frequency counts where tie-breaking rules apply. By carefully

The algorithm is efficient because Counters are designed to provide constant time complexity, 0(1), for element counting, and

**Example Walkthrough** Let's consider an array of integers nums with the following elements: [3, 5, 8, 3, 10, 8, 8, 10]. Following the solution approach:

**Filtering Even Numbers:** Firstly, we filter out the even numbers using the expression (x for x in nums if x % 2 == 0)

in the case of a tie.

class Solution:

Counting Frequencies: We use the Counter from the collections module on the filtered sequence to get a dictionary-like object representing the frequencies: {8: 3, 10: 2}. This indicates that the number 8 occurs three times and the number 10

• For the first item 8:3, v is 3, which is greater than mx, so mx is updated to 3, and ans is updated to 8.

which produces a generator for the sequence [8, 10, 8, 8, 10].

are set initially at -1 and 0, respectively. During the iteration:

occurs two times.

- **Iterating Through Counts**: We then iterate through this dictionary. The loop checks both the number x and its frequency v. Finding the Most Frequent and Smallest: To determine the most frequent and smallest even number, the variables ans and mx
- Next, for 10:2, v is 2, which is less than mx. So there is no update, and ans remains 8. **Result**: At the end of the iteration, the value of ans is 8, which is the most frequent even number in our array nums. If there were no even numbers, ans would have remained -1.

This illustrates how the provided solution approach efficiently finds the most frequent even number or the smallest such number

Solution Implementation

# Initialize variables for the most frequent even number and its count

# Set answer to -1 initially, which means there's no even number in the list

# Update the most frequent number and the highest frequency

**Python** 

def mostFrequentEven(self, nums: List[int]) -> int: # Create a frequency counter for even numbers only even\_count = Counter([x for x in nums if x % 2 == 0])

# If a number has a higher frequency than the current or the same frequency but smaller in value

if frequency > highest\_frequency or (frequency == highest\_frequency and number < most\_frequent\_even):</pre>

### # Iterate over the frequency counter's items for number, frequency in even\_count.items():

from collections import Counter

 $most_frequent_even = -1$ 

return most\_frequent\_even

most\_frequent\_even = number

# Return the most frequent even number

highest\_frequency = frequency

highest\_frequency = 0

```
# Note: The List type hint should be imported from the typing module for complete correctness,
# which depends on the Python version. If you are using Python 3.9 or newer, List type hints
# are built into the Python standard library. For Python 3.8 or earlier, use `from typing import List`.
Java
class Solution {
    public int mostFrequentEven(int[] nums) {
       // Create a HashMap to keep track of the count of even numbers
       Map<Integer, Integer> countMap = new HashMap<>();
       // Iterate over all the elements in the array
        for (int num : nums) {
           // Check if the current element is even
           if (num % 2 == 0) {
                // If it is even, increment its count in the HashMap
                // Using merge function to handle both existing and new numbers
                countMap.merge(num, 1, Integer::sum);
       // Variable to keep track of the most frequent even number
       int mostFrequentNum = -1;
       // Variable to keep track of the maximum frequency
        int maxFrequency = 0;
       // Iterate over the entries in the HashMap
        for (var entry : countMap.entrySet()) {
            int number = entry.getKey();
            int frequency = entry.getValue();
           // Check if the current frequency is greater than the maxFrequency found so far
           // or if the frequency is same as maxFrequency and number is smaller than current mostFrequentNum
            if (maxFrequency < frequency || (maxFrequency == frequency && mostFrequentNum > number)) {
                // Update the most frequent number and the maximum frequency
                mostFrequentNum = number;
                maxFrequency = frequency;
       // Return either the most frequent even number or -1 if no such number was found
       return mostFrequentNum;
```

unordered\_map<int, int> frequencyMap; // Map to store the frequency of each even number

++frequencyMap[num]; // Increment the frequency count for this number

int maxFrequency = 0; // Variable to store the maximum frequency of an even number

// Update mostFrequent if the current frequency is greater than maxFrequency

return mostFrequent; // Return the most frequent even number, or -1 if none exists

// Iterate through the frequency map to find the most frequent even number

int frequency = keyValue.second; // Frequency of the current number

// or if the frequency is the same but the number is smaller

maxFrequency = frequency; // Update the max frequency

int mostFrequent = -1; // Variable to store the most frequent even number. Initialized to -1.

if (maxFrequency < frequency || (maxFrequency == frequency && mostFrequent > number)) {

## **TypeScript** // Finds the most frequent even element in an array of numbers.

C++

public:

#include <vector>

class Solution {

using namespace std;

#include <unordered\_map>

int mostFrequentEven(vector<int>& nums) {

for (auto& keyValue : frequencyMap) {

mostFrequent = number;

// If there are no even numbers, returns -1.

for (const num of nums) {

let answer: number = -1;

let maxCount: number = 0;

if (num % 2 === 0) {

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

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

// Iterate over the array and count the even numbers.

// Map to store the count of even numbers.

for (int num : nums) { // Iterate through the array

if (num % 2 == 0) { // If the element is even

int number = keyValue.first; // Current number

// If multiple elements are equally frequent, returns the smallest one.

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

// Variable to keep track of the highest occurrence count found so far.

// Variable to keep track of the most frequent even number.

```
// Iterate over the countMap to find the most frequent even number.
      for (const [number, count] of countMap) {
          // If the count for the current number is greater than maxCount,
          // or if the count is equal but the number is smaller than the current answer,
          // update answer and maxCount.
          if (maxCount < count || (maxCount === count && answer > number)) {
              answer = number;
              maxCount = count;
      // Return the most frequent even number or -1 if there are no even numbers.
      return answer;
from collections import Counter
class Solution:
   def mostFrequentEven(self, nums: List[int]) -> int:
       # Create a frequency counter for even numbers only
        even_count = Counter([x for x in nums if x % 2 == 0])
       # Initialize variables for the most frequent even number and its count
       # Set answer to -1 initially, which means there's no even number in the list
       most_frequent_even = -1
        highest_frequency = 0
       # Iterate over the frequency counter's items
        for number, frequency in even_count.items():
            # If a number has a higher frequency than the current or the same frequency but smaller in value
            if frequency > highest_frequency or (frequency == highest_frequency and number < most_frequent_even):</pre>
                # Update the most frequent number and the highest frequency
                most_frequent_even = number
                highest frequency = frequency
       # Return the most frequent even number
       return most_frequent_even
# Note: The List type hint should be imported from the typing module for complete correctness,
# which depends on the Python version. If you are using Python 3.9 or newer, List type hints
# are built into the Python standard library. For Python 3.8 or earlier, use `from typing import List`.
```

The given code snippet is a Python function that finds the most frequent even element in a list. To compute the time complexity

Therefore, the total time complexity is the sum of all these steps, which would be 0(n) + 0(k). Since k is bounded by n, the worst-

### A Counter object is constructed with a generator expression: Counter(x for x in nums if x % 2 == 0). This step goes through all n elements of nums, checking for x % 2 == 0 to consider only even numbers. Therefore, this step has a time

complexity of O(n).

3.

Time and Space Complexity

and space complexity, let's analyse the given operations:

A for loop iterates over each item in the Counter object: for x, v in cnt.items(). This could be up to n iterations in the worst case (if all numbers in nums are even and unique). However, since the total number of unique elements in the Counter object is unlikely to be n, let's assume there are k unique even numbers after filtering. The time complexity for this loop is

The ans and mx variables are initialized. This is a constant time operation, so its time complexity is 0(1).

- 0(k). Inside the loop, there are a few constant time comparisons, and possibly an assignment operation. These constant time operations inside the loop do not affect the overall time complexity of O(k) for the loop.
- case time complexity simplifies to O(n). As for space complexity:

The Counter object will hold at most k unique even numbers, which requires O(k) space.

The total space complexity is O(k). Since k is bounded by n, the worst-case space complexity is O(n).

The ans and mx variables are constant space, O(1).

- In conclusion: • Time Complexity: 0(n)
- Space Complexity: 0(n)