<https://leetcode.com/problems/most-frequent-even-element/description/>

Given an integer array nums, return *the most frequent even element*.

If there is a tie, return the **smallest** one. If there is no such element, return -1.

**Example 1:**

**Input:** nums = [0,1,2,2,4,4,1]

**Output:** 2

**Explanation:**

The even elements are 0, 2, and 4. Of these, 2 and 4 appear the most.We return the smallest one, which is 2.

**Example 2:**

**Input:** nums = [4,4,4,9,2,4]

**Output:** 4

**Explanation:**

4 is the even element appears the most.

**Example 3:**

**Input:** nums = [29,47,21,41,13,37,25,7]

**Output:** -1

**Explanation:**

There is no even element.

**Constraints:**

1 <= nums.length <= 2000

0 <= nums[i] <= 10^5

**Attempt 1: 2025-03-30**

**Solution 1: Hash Table (10 min)**

**Style 1: Two Passes**

class Solution {

    public int mostFrequentEven(int[] nums) {

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

        int maxFreq = 0;

        for(int num : nums) {

            if(num % 2 == 0) {

                freq.put(num, freq.getOrDefault(num, 0) + 1);

                maxFreq = Math.max(maxFreq, freq.get(num));

            }

        }

        int result = 100001;

        for(int num : freq.keySet()) {

            if(freq.get(num) == maxFreq) {

                result = Math.min(result, num);

            }

        }

        return result = (result == 100001 ? -1 : result);

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 2: One Pass**

class Solution {

    public int mostFrequentEven(int[] nums) {

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

        int maxFreq = 0;

        int result = 100001;

        for(int num : nums) {

            if(num % 2 == 0) {

                int currFreq = freq.getOrDefault(num, 0) + 1;

                freq.put(num, currFreq);

                if(currFreq > maxFreq) {

                    maxFreq = currFreq;

                    result = num;

                } else if(currFreq == maxFreq && num < result) {

                    result = num;

                }

            }

        }

        return result = (result == 100001 ? -1 : result);

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

This problem requires finding the most frequent even element in an array. If there's a tie, return the smallest one. If no even elements exist, return -1.

**Approach**

Count Frequencies: Use a frequency map to count occurrences of each even number.

Track Maximum: While counting, keep track of:

The highest frequency found

The smallest number with that highest frequency

Handle Edge Cases: Return -1 if no even numbers exist.

**Solution Code**

class Solution {

public int mostFrequentEven(int[] nums) {

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

int maxFreq = 0;

int result = Integer.MAX\_VALUE;

boolean hasEven = false;

for (int num : nums) {

if (num % 2 == 0) {

hasEven = true;

int freq = freqMap.getOrDefault(num, 0) + 1;

freqMap.put(num, freq);

// Update max frequency and result

if (freq > maxFreq) {

maxFreq = freq;

result = num;

} else if (freq == maxFreq && num < result) {

result = num;

}

}

}

return hasEven ? result : -1;

}

}

**Explanation**

Frequency Counting: We iterate through the array and for each even number, we update its count in a HashMap.

Tracking Maximum: While processing each even number:

If its frequency becomes higher than the current maximum, we update both the maximum frequency and result

If it ties the current maximum frequency but is a smaller number, we update the result

Edge Handling: The hasEven flag ensures we return -1 when no even numbers exist.

**Complexity Analysis**

Time Complexity: O(n) - Single pass through the array

Space Complexity: O(n) - For storing frequency counts in the worst case

**Alternative Solution (Using Array for Frequency Count)**

If the input range is limited (e.g., 0 <= nums[i] <= 10^5), we can optimize space:

class Solution {

public int mostFrequentEven(int[] nums) {

int[] freq = new int[100001]; // Adjust size based on constraints

int maxFreq = 0;

int result = Integer.MAX\_VALUE;

boolean hasEven = false;

for (int num : nums) {

if (num % 2 == 0) {

hasEven = true;

freq[num]++;

if (freq[num] > maxFreq) {

maxFreq = freq[num];

result = num;

} else if (freq[num] == maxFreq && num < result) {

result = num;

}

}

}

return hasEven ? result : -1;

}

}

**Refer to**

[L451.P14.5.Sort Characters By Frequency (Ref.L347)](note://512A6121D907491D8CF8FDD9119E5CC3)