<https://leetcode.com/problems/minimum-rounds-to-complete-all-tasks/description/>

You are given a **0-indexed** integer array tasks, where tasks[i] represents the difficulty level of a task. In each round, you can complete either 2 or 3 tasks of the **same difficulty level**.

Return *the****minimum****rounds required to complete all the tasks, or*-1*if it is not possible to complete all the tasks.*

**Example 1:**

**Input:** tasks = [2,2,3,3,2,4,4,4,4,4]

**Output:** 4

**Explanation:** To complete all the tasks, a possible plan is:

- In the first round, you complete 3 tasks of difficulty level 2.

- In the second round, you complete 2 tasks of difficulty level 3.

- In the third round, you complete 3 tasks of difficulty level 4.

- In the fourth round, you complete 2 tasks of difficulty level 4.

It can be shown that all the tasks cannot be completed in fewer than 4 rounds, so the answer is 4.

**Example 2:**

**Input:** tasks = [2,3,3]

**Output:** -1

**Explanation:** There is only 1 task of difficulty level 2, but in each round, you can only complete either 2 or 3 tasks of the same difficulty level. Hence, you cannot complete all the tasks, and the answer is -1.

**Constraints:**

1 <= tasks.length <= 10^5

1 <= tasks[i] <= 10^9

**Note:** This question is the same as [2870: Minimum Number of Operations to Make Array Empty](https://leetcode.com/problems/minimum-number-of-operations-to-make-array-empty/description/)

**Attempt 1: 2025-06-01**

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

class Solution {

    public int minimumRounds(int[] tasks) {

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

        for(int task : tasks) {

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

        }

        int result = 0;

        for(int count: freq.values()) {

            if(count == 1) {

                return -1;

            }

            // The minimum rounds is ceil(count / 3)

            result += (count + 2) / 3;

        }

        return result;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

This problem requires us to find the minimum number of rounds needed to complete all tasks, where in each round we can complete 2 or 3 tasks of the same difficulty level.

**Approach**

**Count Frequencies**: First, we count how many tasks there are of each difficulty level using a hash map.

**Calculate Rounds**: For each difficulty level, we determine the minimum number of rounds needed to complete all tasks of that level:

If the count is 1, it's impossible (return -1)

Otherwise, the minimum rounds is count/3 rounded up (since we can use combinations of 2s and 3s)

**Sum Rounds**: Sum up the rounds for all difficulty levels.

**Solution Code**

import java.util.HashMap;

import java.util.Map;

class Solution {

public int minimumRounds(int[] tasks) {

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

// Count the frequency of each task difficulty

for (int task : tasks) {

frequencyMap.put(task, frequencyMap.getOrDefault(task, 0) + 1);

}

int rounds = 0;

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

if (count == 1) {

return -1;

}

// The minimum rounds is ceil(count / 3)

rounds += (count + 2) / 3;

}

return rounds;

}

}

**Explanation**

**Frequency Counting**: We use a hash map to count how many times each task difficulty appears in the input array.

**Impossible Case Handling**: If any task difficulty appears exactly once, we immediately return -1 since we can't complete it (you can't do 1 task in rounds of 2 or 3).

**Round Calculation**: For each task count, we calculate the minimum rounds needed by using the formula (count + 2) / 3. This effectively gives us the ceiling of count / 3 without using floating point arithmetic.

For example:

count=3: (3+2)/3 = 1 round (exactly 3)

count=4: (4+2)/3 = 2 rounds (3+1 isn't possible, so 2+2)

count=5: (5+2)/3 = 2 rounds (3+2)

count=6: (6+2)/3 = 2 rounds (3+3)

**Summing Rounds**: We sum up the rounds needed for all task difficulties and return the total.

This approach efficiently calculates the minimum rounds by leveraging integer arithmetic and handles edge cases properly.

**Refer to**

[L2451.Odd String Difference (Ref.L2244)](note://WEB0daa0a5ac102958be928a0f47ec17a4b)