<https://leetcode.com/problems/maximum-number-of-pairs-in-array/description/>

You are given a **0-indexed** integer array nums. In one operation, you may do the following:

Choose **two** integers in nums that are **equal**.

Remove both integers from nums, forming a **pair**.

The operation is done on nums as many times as possible.

Return *a****0-indexed****integer array*answer*of size*2*where*answer[0]*is the number of pairs that are formed and*answer[1]*is the number of leftover integers in*nums*after doing the operation as many times as possible*.

**Example 1:**

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

**Output:** [3,1]

**Explanation:**

Form a pair with nums[0] and nums[3] and remove them from nums. Now, nums = [3,2,3,2,2].

Form a pair with nums[0] and nums[2] and remove them from nums. Now, nums = [2,2,2].

Form a pair with nums[0] and nums[1] and remove them from nums. Now, nums = [2].

No more pairs can be formed. A total of 3 pairs have been formed, and there is 1 number leftover in nums.

**Example 2:**

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

**Output:** [1,0]

**Explanation:**

Form a pair with nums[0] and nums[1] and remove them from nums. Now, nums = [].

No more pairs can be formed. A total of 1 pair has been formed, and there are 0 numbers leftover in nums.

**Example 3:**

**Input:** nums = [0]

**Output:** [0,1]

**Explanation:**

No pairs can be formed, and there is 1 number leftover in nums.

**Constraints:**

1 <= nums.length <= 100

0 <= nums[i] <= 100

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

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

class Solution {

    public int[] numberOfPairs(int[] nums) {

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

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

            freq.put(nums[i], freq.getOrDefault(nums[i], 0) + 1);

        }

        int pairs = 0;

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

            pairs += count / 2;

        }

        int remain = nums.length - pairs \* 2;

        return new int[]{pairs, remain};

    }

}

Time Complexity: O(n), where n is the length of the array. We perform two passes: one to count frequencies and another to calculate pairs.

Space Complexity: O(n), in the worst case where all elements are unique, the HashMap will store all elements.

**Refer to Deepseek**

This problem requires finding the maximum number of pairs that can be formed from an array of numbers, where each pair consists of two equal numbers, and also returning the count of leftover numbers after pairing.

**Solution Approach**

Count Frequencies: Count how many times each number appears in the array.

Calculate Pairs: For each number, the number of pairs is its frequency divided by 2.

Sum Pairs: Sum all the pairs to get the total number of pairs.

Calculate Leftovers: The leftover numbers are the total elements minus twice the number of pairs.

**Java Solution**

class Solution {

public int[] numberOfPairs(int[] nums) {

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

int pairs = 0;

// Count the frequency of each number

for (int num : nums) {

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

}

// Calculate the number of pairs

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

pairs += count / 2;

}

// Calculate the leftover numbers

int leftovers = nums.length - pairs \* 2;

return new int[]{pairs, leftovers};

}

}

**Explanation**

Frequency Counting: We use a HashMap to count how many times each number appears in the array.

Pair Calculation: For each number, the number of possible pairs is the integer division of its frequency by 2 (e.g., frequency 5 → 2 pairs).

Total Pairs: Sum all individual pairs to get the total number of pairs.

Leftover Calculation: The leftover numbers are calculated by subtracting twice the number of pairs from the total elements (since each pair uses 2 numbers).

Return Result: Return an array where the first element is the total number of pairs and the second is the count of leftover numbers.

**Complexity Analysis**

Time Complexity: O(n), where n is the length of the array. We perform two passes: one to count frequencies and another to calculate pairs.

Space Complexity: O(n), in the worst case where all elements are unique, the HashMap will store all elements.

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

If the problem constraints specify that the numbers are within a small range (e.g., 0 to 100), we can optimize space by using an array instead of a HashMap:

class Solution {

public int[] numberOfPairs(int[] nums) {

int[] frequency = new int[101]; // Assuming numbers are between 0 and 100

int pairs = 0;

for (int num : nums) {

frequency[num]++;

}

for (int count : frequency) {

pairs += count / 2;

}

int leftovers = nums.length - pairs \* 2;

return new int[]{pairs, leftovers};

}

}

This solution reduces space complexity to O(1) (fixed-size array) while maintaining O(n) time complexity. It's more efficient when the number range is limited and known.

**Refer to**

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