1224. Maximum Equal Frequency

Given an array `nums` of positive integers, return the longest possible length of an array prefix of `nums`, such that it is possible to remove **exactly one** element from this prefix so that every number that has appeared in it will have the same number of occurrences.

If after removing one element there are no remaining elements, it's still considered that every appeared number has the same number of ocurrences (0).

Example 1:

```
Input: nums = [2,2,1,1,5,3,3,5]
Output: 7
Explanation: For the subarray [2,2,1,1,5,3,3] of length 7, if we remove
nums[4] = 5, we will get [2,2,1,1,3,3], so that each number will appear
exactly twice.
```

Example 2:

```
Input: nums = [1,1,1,2,2,2,3,3,3,4,4,4,5]
Output: 13
```

Constraints:

- `2 <= nums.length <=` 10^5
- `1 <= nums[i] <=` 10^5

Intuition

Count frequency of the elements
We also need to count the number of elements with that frequency

Explanation

There are 2 cases where we need to update the result:

Case 1:
frequency * (number of elements with that frequency) == length AND `i` != nums.length - 1
E.g. `[1,1,2,2,3]`
When the iteration is at index 3, the count will be equal to the length. It should update the result with (length + 1) as it should take an extra element in order to fulfil the condition.

Case 2:
frequency * (number of elements with that frequency) == length - 1
E.g. `[1,1,1,2,2,3]`
When the iteration is at index 4, the count will be equal to (length - 1). It should update the result with length as it fulfil the condition.

Complexity

Time: `O(N)` where N is the number of elements
Space: `O(N)`

Java

```
public int maxEqualFreq(int[] nums) {
        Map<Integer, Integer> countMap = new HashMap<>();
        Map<Integer, Integer> freqMap = new HashMap<>();
        int res = 0;
        for (int i = 0; i < nums.length; i++) {
                // update counts
                countMap.put(nums[i], countMap.getOrDefault(nums[i], 0) + 1);
                int freq = countMap.get(nums[i]);
                // update counts with that frequency
                freqMap.put(freq, freqMap.getOrDefault(freq, 0) + 1);

                int count = freqMap.get(freq) * freq;
                if (count == i + 1 && i != nums.length - 1) { // case 1
                        res = Math.max(res, i + 2);
                } else if (count == i) { // case 2
                        res = Math.max(res, i + 1);
                }
        }
        return res;
}
```

C++

```
int maxEqualFreq(vector<int>& nums) {
        unordered_map<int, int> countMap;
        unordered_map<int, int> freqMap;
        int res = 0;
        for (int i = 0; i < nums.size(); i++) {
                // update counts
                countMap[nums[i]]++;
                int freq = countMap[nums[i]];
                // update counts with that frequency
                freqMap[freq]++;

                int count = freqMap[freq] * freq;
                if (count == i + 1 && i != nums.size() - 1) { // case 1
                        res = max(res, i + 2);
                } else if (count == i) { // case 2
                        res = max(res, i + 1);
                }
        }
        return res;
}
```

Python

```
def maxEqualFreq(self, nums):
        counts, freq = collections.Counter(), collections.Counter()
        res = 0
        for i, num in enumerate(nums):
                # update counts
                counts[num] += 1
                # update counts with that frequency
                freq[counts[num]] += 1

                count = freq[counts[num]] * counts[num]
                if count == i + 1 and i != len(nums) - 1: # case 1
                        res = max(res, i + 2)
                elif count == i: # case 2
                        res = max(res, i + 1)
        return res
```

Missing Test Cases

```
[1,1,1,2,2,2,3,3,3]
```