<https://leetcode.com/problems/unique-number-of-occurrences/description/>

Given an array of integers arr, return true if the number of occurrences of each value in the array is unique or false otherwise.

**Example 1:**

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

**Output:** true

**Explanation:** The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

**Example 2:**

**Input:** arr = [1,2]

**Output:** false

**Example 3:**

**Input:** arr = [-3,0,1,-3,1,1,1,-3,10,0]

**Output:** true

**Constraints:**

1 <= arr.length <= 1000

-1000 <= arr[i] <= 1000

**Attempt 1: 2024-11-29**

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

class Solution {

    public boolean uniqueOccurrences(int[] arr) {

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

        for(int a : arr) {

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

        }

        Set<Integer> set = new HashSet<>();

        for(int fq : freq.values()) {

            if(set.contains(fq)) {

                // terminate as soon as a duplicate is detected

                return false;

            } else {

                set.add(fq);

            }

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/unique-number-of-occurrences/solutions/392858/java-python-3-4-liner-and-2-liner-using-map-and-set-w-brief-explanation-and-analysis/>

**Q & A**

Q: Your method 1 is a succinct solution, but what if the input is very large and all occurrences are duplicates? Then the code will always have to finish filling out all occurrences before comparing.

A: You have a solid point, and we can terminate the iteration once finding a duplicate occurrence. The early exit codes are in method 2 Actually, it is a trade off between performance (method 2) and readability (method 1).

**End of Q & A**

**Method 1:**

Count the occurrences of each char;

Compare if the numbers of distinct chars and distinct counts are equal.

public boolean uniqueOccurrences(int[] arr) {

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

for (int a : arr)

count.put(a, 1 + count.getOrDefault(a, 0));

return count.size() == new HashSet<>(count.values()).size();

}

**Method 2: Early Exit**

public boolean uniqueOccurrences(int[] arr) {

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

for (int a : arr) {

count.merge(a, 1, Integer::sum);

}

Set<Integer> seen = new HashSet<>();

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

if (!seen.add(freq)) {

return false;

}

}

return true;

}

**Analysis:**

Time & space: O(n), where n = arr.length