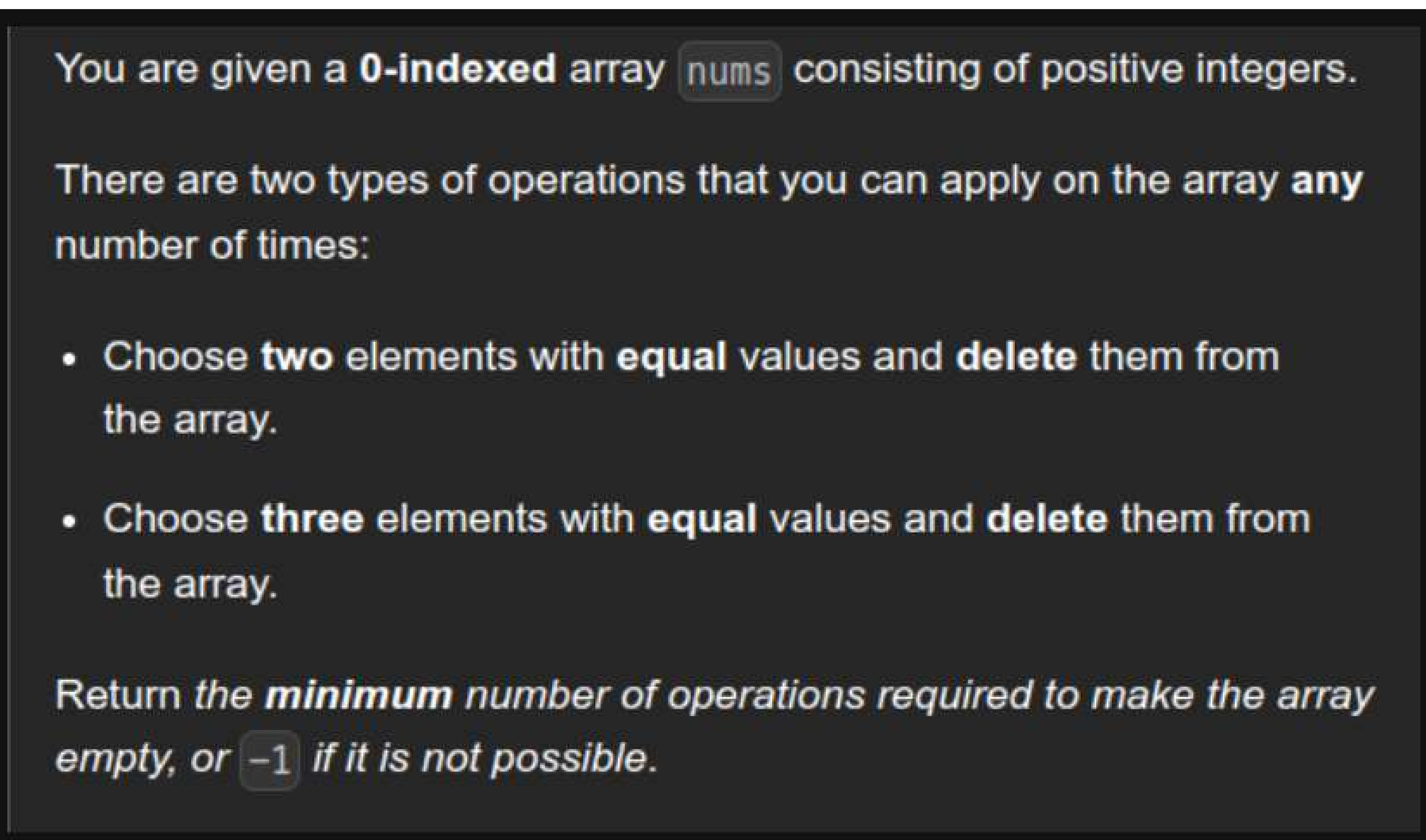


You are given a **0-indexed** array `nums` consisting of positive integers.

There are two types of operations that you can apply on the array **any** number of times:

- Choose **two** elements with **equal** values and **delete** them from the array.
- Choose **three** elements with **equal** values and **delete** them from the array.

Return *the **minimum** number of operations required to make the array empty, or* `-1` *if it is not possible.*

```
input = [2,3,3,2,2,4,2,3,4]

min number of operations = 4
```

Approach:

1→ Store frequency of each element in map

2→ Iterate through map and find min operations

Observation:

1→ We will prioritize the pairs of three to get min number of operations to make array empty

2→ What if we use ceil function??

```
2  → 1
3  → 1
4  → 2
5  → 2
6  → 2
7  → 3
8  → 3
9  → 3
10 → 4
11 → 4
12 → 4
```

As you can see from the table, dividing the frequency by 3 and using ceil always gives the correct minimum number of operations needed to delete all elements.

DRY RUN:

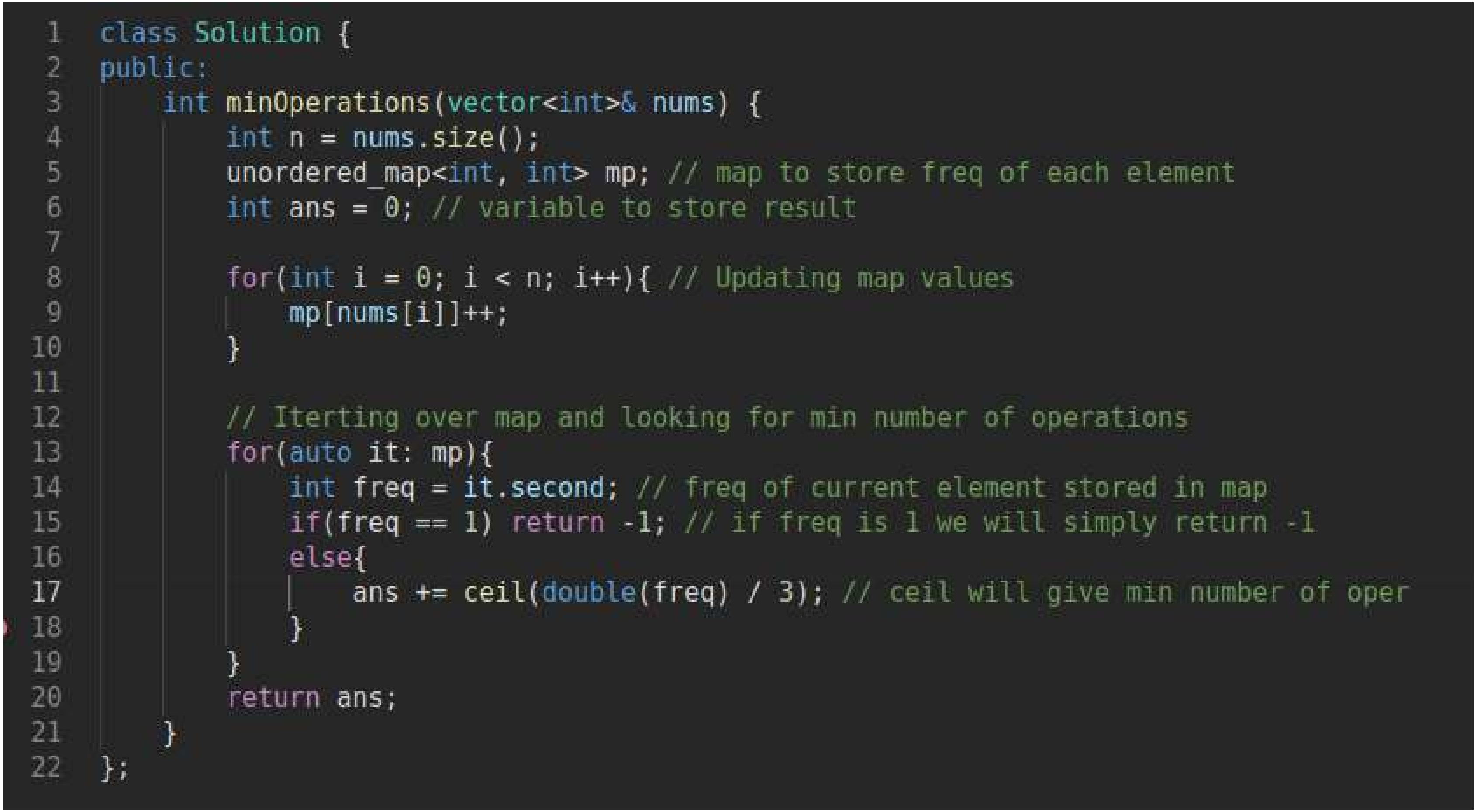
```
input = [2,3,3,2,2,4,2,3,4]
```

ans = ceil(double(4)/3) = 2
ans = ceil(double(3)/3) = 1
ans = ceil(double(2)/3) = 1

Total ans = 2 + 1 + 1 = 4

MAP

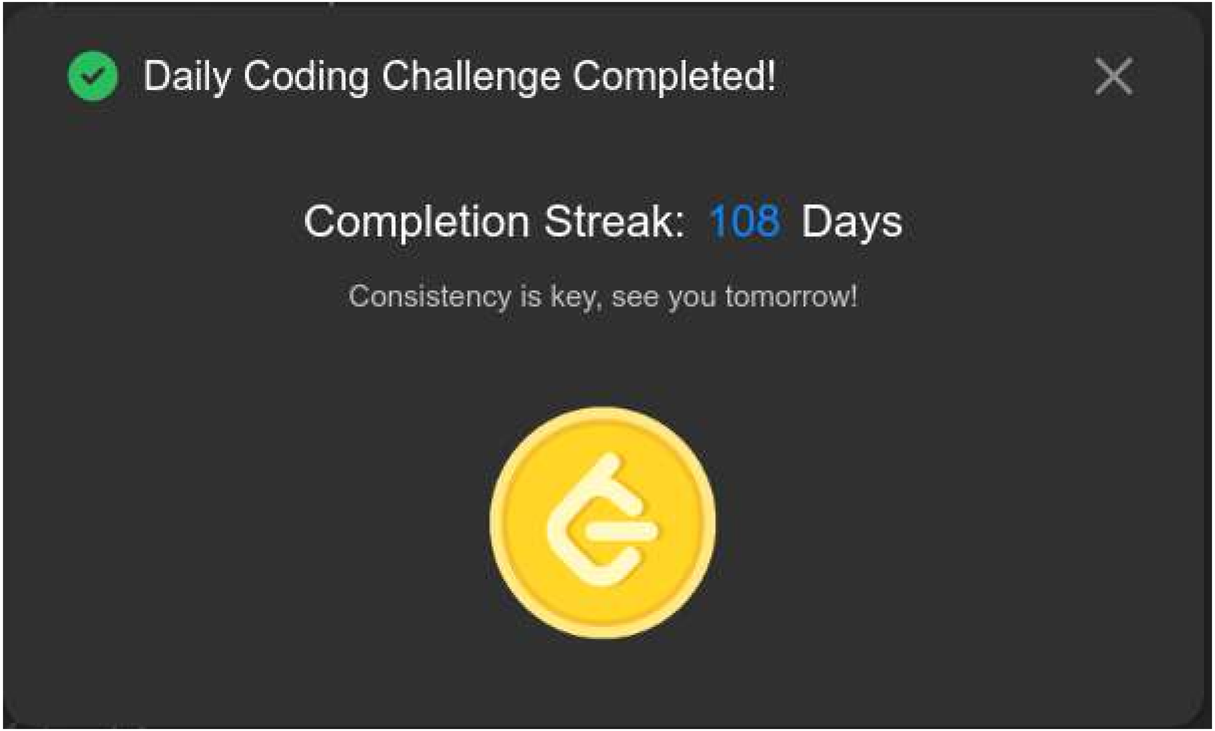
KEY	Freq
4	2
3	3
2	4



```
class Solution {
public:
    int minOperations(vector<int>& nums) {
        int n = nums.size();
        unordered_map<int, int> mp; // map to store freq of each element
        int ans = 0; // variable to store result

        for(int i = 0; i < n; i++){ // Updating map values
            mp[nums[i]]++;
        }

        // Iterting over map and looking for min number of operations
        for(auto it: mp){
            int freq = it.second; // freq of current element stored in map
            if(freq == 1) return -1; // if freq is 1 we will simply return -1
            else{
                ans += ceil(double(freq) / 3); // ceil will give min number of oper
            }
        }
        return ans;
    }
};
```



Daily Coding Challenge Completed!

Completion Streak: 108 Days

Consistency is key, see you tomorrow!