# Leetcode-2094

## Problem Summary

An integer array `digits` is given, possibly containing duplicates.

The task is to find all unique 3-digit integers such that:

1. They are formed by concatenating exactly 3 elements from `digits`.
2. They have **no leading zeros**.
3. They are **even numbers**.

The output should be sorted in ascending order.

---

## Approach in the Given Code

1. **Store Digit Frequencies**
   - An `unordered_map<int,int>` named `mp` is created to store how many times each digit appears in the array `digits`.
   - Example: If `digits = [2, 2, 8]`, then `mp = {2: 2, 8: 1}`.
2. **Iterate Through All Possible Even 3-Digit Numbers**
   - The loop runs from `100` to `999` with a step of `2` (`i += 2`).
   - This ensures that only **even numbers** are considered (last digit even).
3. **Extract Digits of the Current Number**
   - `a` = units place (last digit)
   - `b` = tens place (middle digit)
   - `c` = hundreds place (first digit)
   - These are extracted using modulo `%` and integer division `/`.
4. **Check Digit Availability Using Frequency Map**
   - First, check if digit `a` is present in the map.

- If found, decrement its frequency (simulating that it has been used).
- If the frequency becomes zero, remove it from the map temporarily.

- Then, check if digit `b` is present in the updated map.
  - If found, decrement its frequency.
  - If the frequency becomes zero, remove it from the map temporarily.
- Finally, check if digit `c` exists in the further updated map.
  - If found, push the number into the result vector `ans`.

5. **Restore Frequencies After Each Check**
   - After checking `b` and `c`, the frequency of `b` is incremented back.
   - After finishing with `a`, its frequency is also incremented back.
   - This ensures that the frequency map is reset for the next iteration.
6. **Return the Result**
   - The vector `ans` is returned.
   - Since the loop runs from `100` upwards, the result is already sorted.

---

# Key Details in the Code

- `unordered_map<int,int> mp;` → Stores frequency of each digit.
- `for(int i=100;i<=999;i+=2)` → Iterates through all even 3-digit numbers.
- **Digit extraction:**

```
int a = x % 10; // units
x = x / 10;
int b = x % 10; // tens
x /= 10;
int c = x % 10; // hundreds
```

- **Frequency decrement and erase:**
  - `mp[a]--; if(mp[a]==0) mp.erase(a);`

Removes the digit from the map temporarily if its count becomes zero.

- **Frequency restoration:**
  - After processing, the counts of `a` and `b` are restored by incrementing them again.

---

## Why This Works Efficiently

- The loop only runs for numbers from `100` to `999` (900 numbers).
- Since the step is `2`, only 450 iterations happen.
- Each check is **O(1)** because of `unordered_map`.
- The approach directly eliminates invalid numbers (odd numbers, numbers with unavailable digits, numbers with leading zero).

---

## Time Complexity

- **O(450 × constant)** → effectively **O(1)** for all practical purposes.

---

## Example Walkthrough

**Input:** `digits = [2, 1, 3, 0]`

- `mp = {2:1, 1:1, 3:1, 0:1}`

Loop:

- For `i = 102`:
  - a = 2 ✅ available → decrement mp[2]
  - b = 0 ✅ available → decrement mp[0]
  - c = 1 ✅ available → push `102` into ans.
  - Restore mp[0] and mp[2].
- For `i = 104` → fails because `4` not in map.

Repeating this process produces:

`[102, 120, 130, 132, 210, 230, 302, 310, 312, 320]`.

## Original Code

```
class Solution {
public:
    vector<int> findEvenNumbers(vector<int>& digits) {
        vector<int>ans;
        unordered_map<int,int> mp;
        for(int ele : digits){
            mp[ele]++;
        }
        for(int i=100;i<=999;i+=2){
            int x = i;
            int a = x % 10;//last digit
            x = x / 10;
            int b = x % 10;//second last digit
            x /= 10;
            int c = x % 10;
            if(mp.find(a) != mp.end()){
                //a found
                mp[a]--;
                if(mp[a]==0)mp.erase(a);
                if(mp.find(b) != mp.end()){
                    //b found in the map
                    mp[b]--;
                    if(mp[b]==0)mp.erase(b);
                    if(mp.find(c) != mp.end()){
                        //c is also existing
                        ans.push_back(i);
                    }
                    mp[b]++;
                }
                mp[a]++;
            }
        }
```

```
        return ans;
    }
};
```

---