

Single Number II

Bit Manipulation • Medium (Generalized Frequency Cancellation)

The Hook

This is Single Number's harder sibling. Now every number appears three times except one. XOR won't help — $x \oplus x \oplus x = x$, not 0. You need a way to count bits modulo 3 and keep only the bits that appeared exactly once. This problem forces you to think about bit counters.

Core Intuition — Bit Frequency Counting

For each of the 32 bit positions, count how many numbers have a 1 in that position. If a number appears three times, it contributes 0 or 3 to each bit count. Taking that count mod 3 gives 0 for those numbers. The single number contributes either 0 or 1 — and mod 3 preserves that. Build the answer bit by bit.

Two-Variable State Machine (ones & twos)

A cleaner $O(1)$ space approach uses two variables — ones and twos — as 2-bit counters per bit position. ones holds bits seen 1 time (mod 3). twos holds bits seen 2 times (mod 3). When a bit reaches 3 counts, it's cleared from both.

- $\text{ones} = \text{ones} \oplus n \ \& \ \sim \text{twos}$ (add to 'once' if not in 'twice')
- $\text{twos} = \text{twos} \oplus n \ \& \ \sim \text{ones}$ (add to 'twice' if not now in 'once')
- After all elements: ones holds the single number

Java Implementation

```
public class SingleNumberII {  
    // Approach 1: Bit counting (intuitive)  
    public static int singleNumber(int[] nums) {  
        int result = 0;  
        for (int i = 0; i < 32; i++) {  
            int bitSum = 0;  
            for (int num : nums)  
                bitSum += (num >> i) & 1;  
            // if bitSum % 3 == 1, single number has this bit set  
            result |= (bitSum % 3 & 1) < 1 ? (1 << i) : 0;  
        }  
        return result;  
    }  
}
```

```

result |= (bitSum % 3) << i;
}
return result;
}

// Approach 2: State machine (elegant)
public static int singleNumberFast(int[] nums) {
    int ones = 0, twos = 0;
    for (int n : nums) {
        ones = (ones ^ n) & ~twos;
        twos = (twos ^ n) & ~ones;
    }
    return ones;
}
}

```

■ Complexity

Case	Time	Space	Why
Bit count	$O(32n) = O(n)$	$O(1)$	32 passes over array, one per bit position
State machine	$O(n)$	$O(1)$	Single pass, two integer variables

■ Practice Problems

Problem	Difficulty
Single Number II (LeetCode 137)	Medium
Single Number III (LeetCode 260)	Medium
Find the element appearing once (k times)	Medium
Maximum XOR of Two Numbers (LeetCode 421)	Hard

■ Quick Revision

- Count bits at each position; take mod 3 → remaining bits form the answer
- State machine: ones, twos track 1-count and 2-count bits separately

■ Generalizes: for 'appears k times except one', use mod k

■ Mental Model

Think of each bit as a bucket. You're pouring votes in. Three votes cancel each other. At the end, buckets with exactly one vote left reveal the lone number. The state machine automates this bucket logic in $O(1)$ space.