Array Mastery: Logic, Math, and Constraints

A code review session on LeetCode 485 & 1295

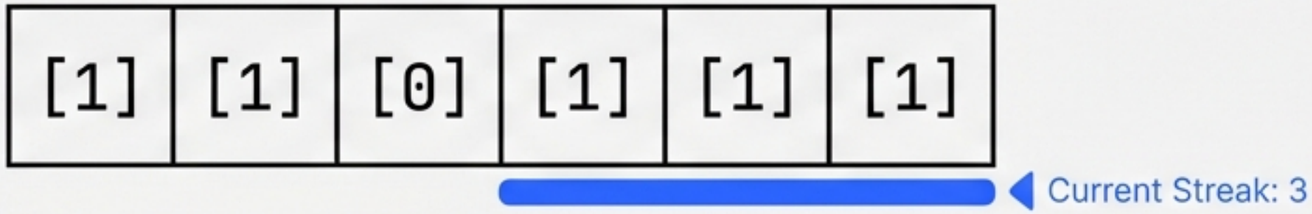
```
1 1 0
1 1 0 1       1
1 1 0 1     1 1
1 1 0 1 0   1 1
1 1 0 1 0 1 1 1
1 1 0 1 0 1 1 1
1 1 0 1 0 1 1 1
1 1 1 1 0 1 1 1
1 1   1 0 1 1 1
1       1 0 1 1
          1 1 1
```

This deck explores two fundamental array problems. We begin with basic iteration patterns and evolve into advanced optimization techniques using logarithms and constraint analysis.

The Goal: Move from “It works” to “It scales.”

The Warm-Up: Max Consecutive Ones (LC 485)

THE INPUT



THE CHALLENGE

Challenge: Given a binary array, find the maximum number of consecutive 1s.

```
Input:  [1, 1, 0, 1, 1, 1]
Output: 3
```

THE CONSTRAINTS

Constraints
• Array length up to 10^5 • Single pass required (O(N) time) • O(1) Space complexity

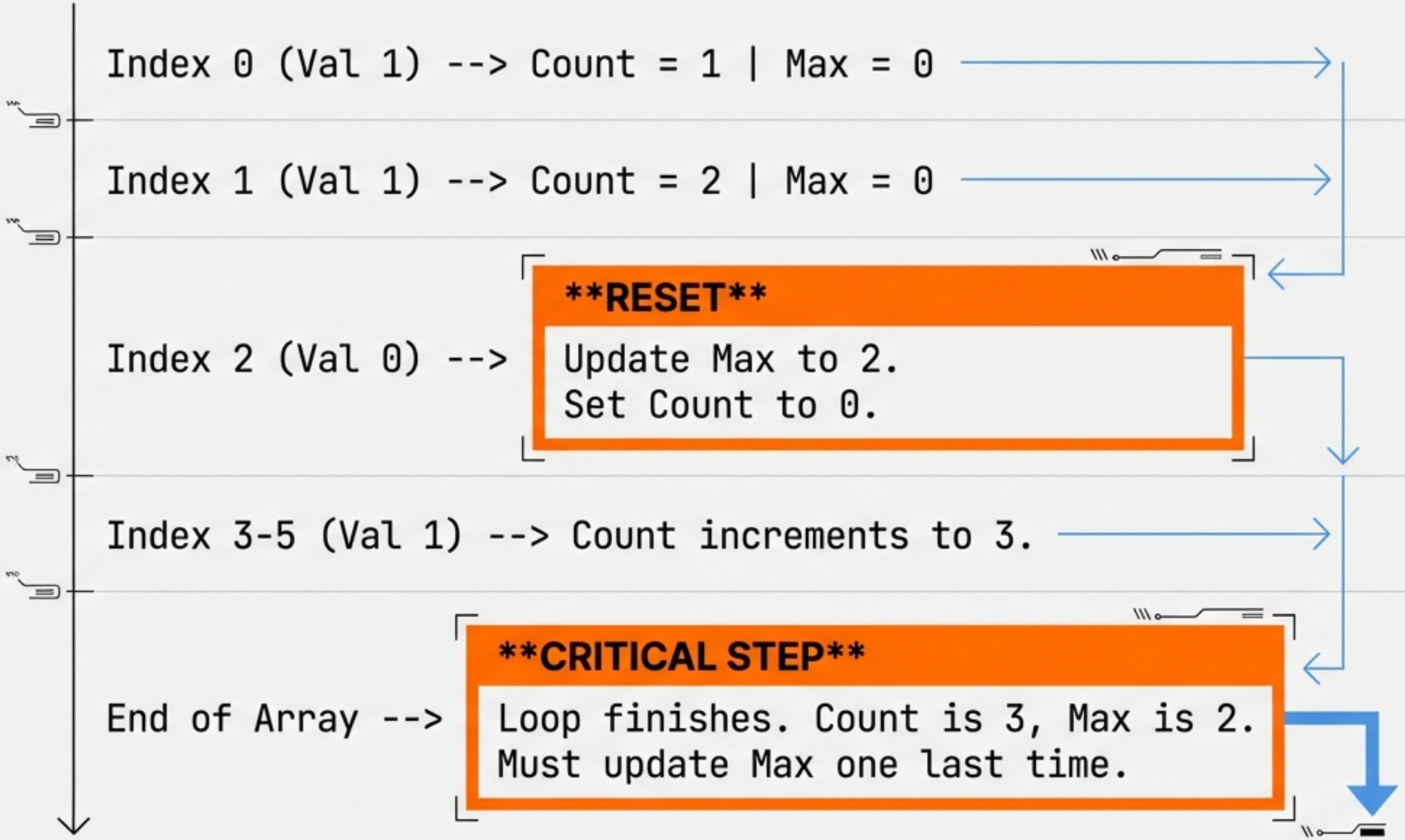
⚠ **Warning**

The Trap: It looks simple, but missing the final streak at the end of the array is a common failure mode.

The Logic: Reset & Record

Key Insight:

If the array ends with a 1, the logic inside the loop (which triggers on 0) never executes for that final streak.



Implementation: The Single-Pass Solution

Solution.java

```java
int count = 0;
int max = 0;
for (int i = 0; i < nums.length; i++) {
    if (nums[i] == 1) {
        count++;
    } else {
        max = Math.max(max, count); // Capture streak
        count = 0; // Reset
    }
}
return Math.max(max, count); // Handle edge case
```

Constraint Orange

Reset Logic

Edge Case Handling

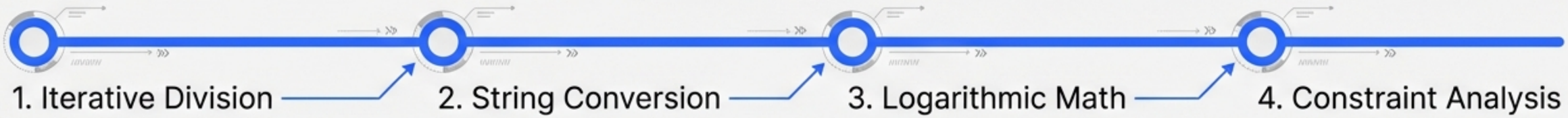
Performance
Time Complexity: O(N) (One pass)
Space Complexity: O(1) (No extra structures)

The Deep Dive: Numbers with Even Digits (LC 1295)

Transitioning from simple iteration to mathematical optimization.

Total Count: 2

The Optimization Roadmap



Approach 1: The Scrappy Solution (Iterative Division)

Visual Logic

Inter

Code & Critique

Inter

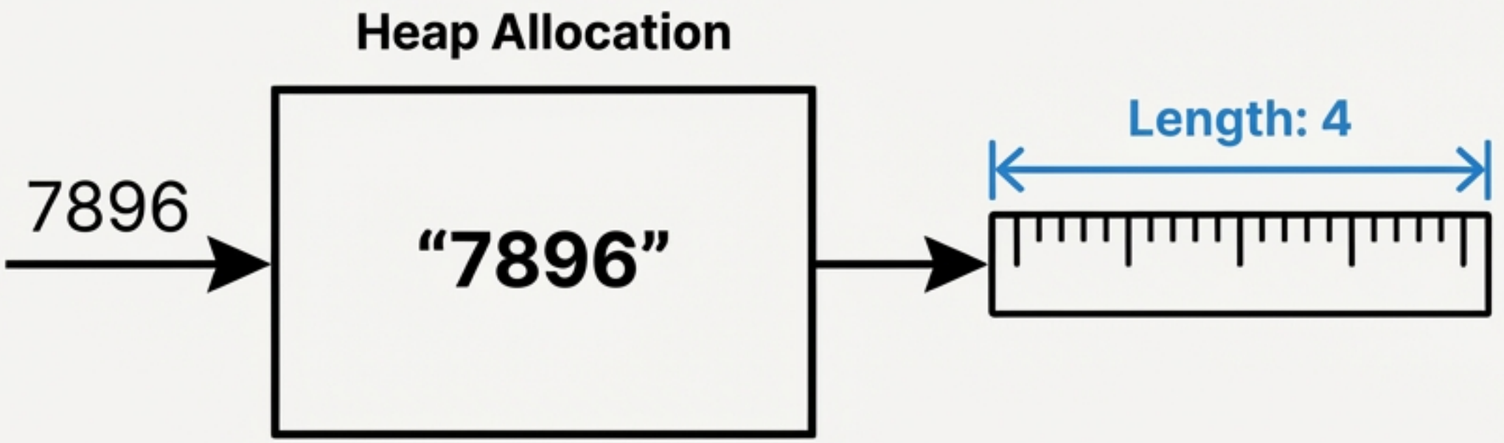
Solution.java

```java
while (num != 0) {
    num /= 10;
    digitCount++;
}
```

Critique: Intuitive, but requires nested operations.

Complexity: O(N * log10(max_val))

Approach 2: The Developer’s Shortcut (Strings)




```
if (String.valueOf(num).length() % 2 == 0) {
    evenCount++;
}
```

Critique

Pros:
One line of logic. Readable.

Cons:
Expensive. Converting int to String allocates new memory on the Heap.

Verdict:
Good for scripting, bad for high-performance systems.

Approach 3: The Mathematician's Way (Logarithms)

The Math

$10^3 = 1000$ (4 digits)

$10^4 = 10000$ (5 digits)

Digits = floor(**log10**(num)) + 1

log10(100) = 2.0 → floor(2.0) + 1 = 3

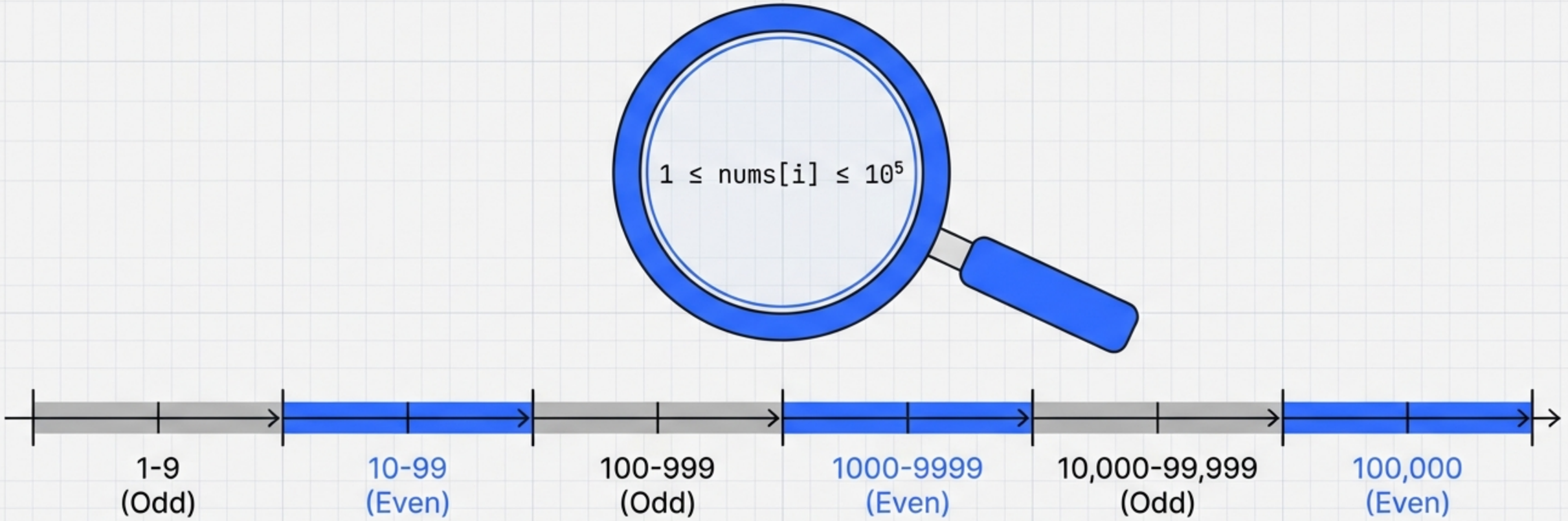
log10(999) ≈ 2.99 → floor(2.99) + 1 = 3

Implementation

```
1    int digits = (int) Math.floor(Math.log10(num)) + 1;
2
```

Critique: Mathematically elegant. No loops. Bound by math library performance.

The “Constraint Spotlight”



The Insight: We don’t need **a generic algorithm.** We only need to check if the number falls into an **“Even Range”.**

Approach 4: Constraint-Based Optimization

No Division Loops

No Heap Allocation

```
for (int num : nums) {
    if ((num >= 10 && num <= 99) ||
        (num >= 1000 && num <= 9999) ||
        (num == 100000)) {
        evenCount++;
    }
}
```

No Floating Point Math

Time Complexity: O(N) | **Space Complexity:** O(1) | **Status:** Interview Winner

Summary & Key Takeaways

The Loop Illusion (LC 485)

Loops terminate when conditions fail, but data processing might be incomplete. Always check the state of your variables after the loop terminates.

The Constraint Advantage (LC 1295)

A generic solution is good; a constraint-aware solution is optimal. If the input is bounded (10^5), hard-coding ranges is faster than calculating them dynamically.

Mastery is not just knowing the algorithms; it's knowing your data.