Blind 75: Array Essentials

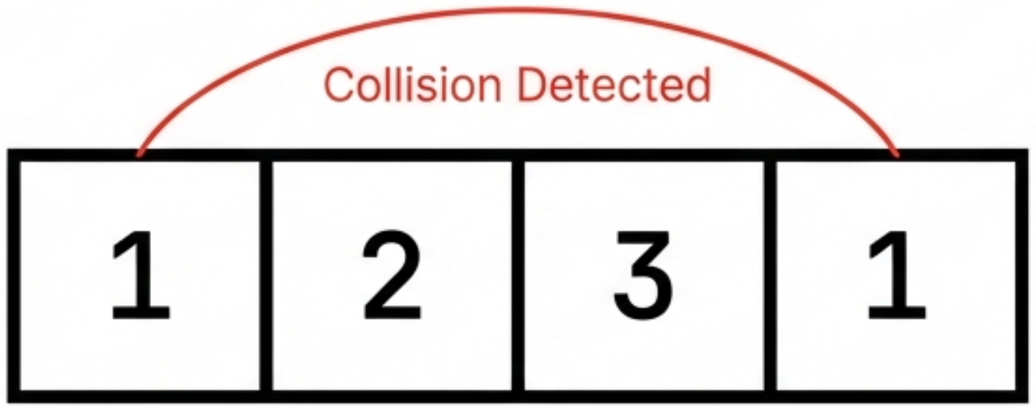
Mastering the “Contains Duplicate” Pattern

Problem 01 / Array Module

Two Pointers
Sliding Window
Stack
Binary Search
Linked List
Array > Contains Duplicate
Trees
Graphs
Dynamic Programming
Greedy
Backtracking

The Binary Goal: Distinct or Repeated?

Case A: Repetition



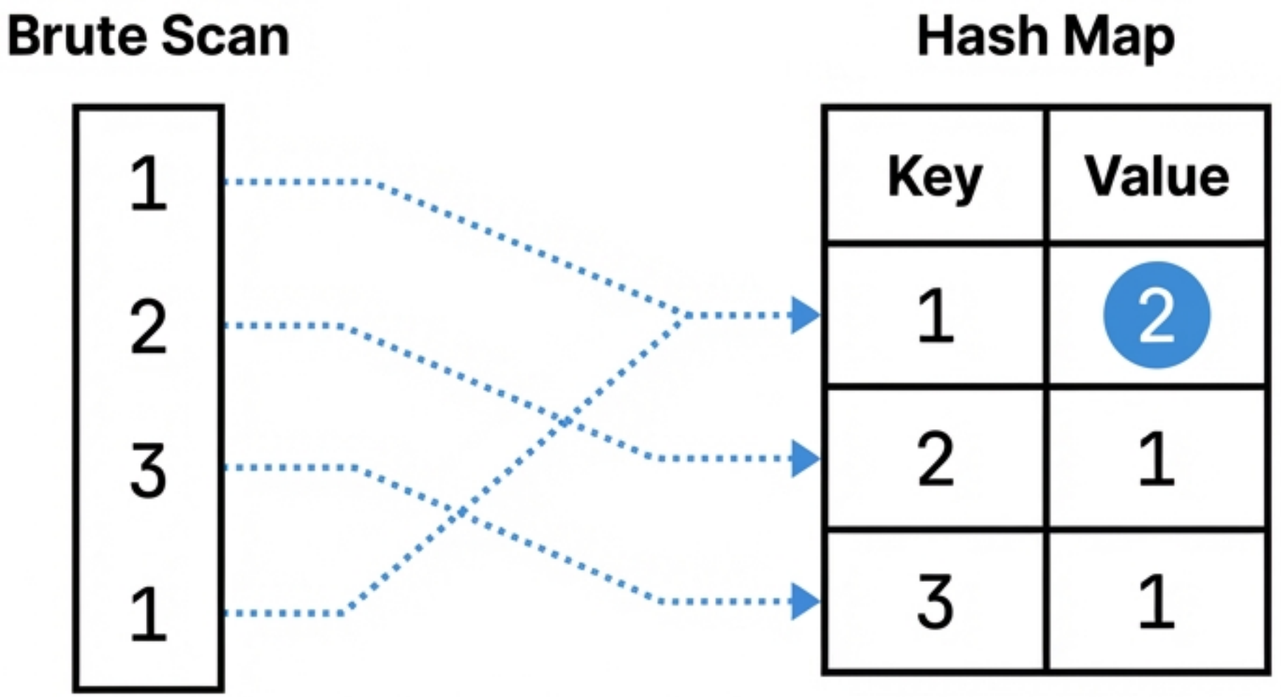
Return True

Case B: Distinct

Return False

Return true if any value appears at least twice. Return false if every element is distinct.

The Passive Approach: Frequency Mapping



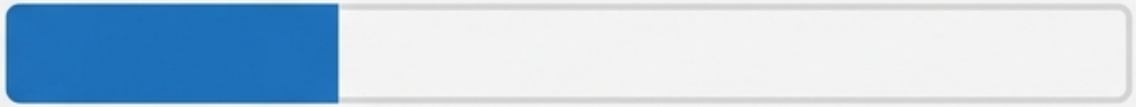
Critique: The algorithm passively counts every element before returning a result. It does not stop early.

Optimization: From Counting to Looking Up

Counting Strategy

Scan Entire List

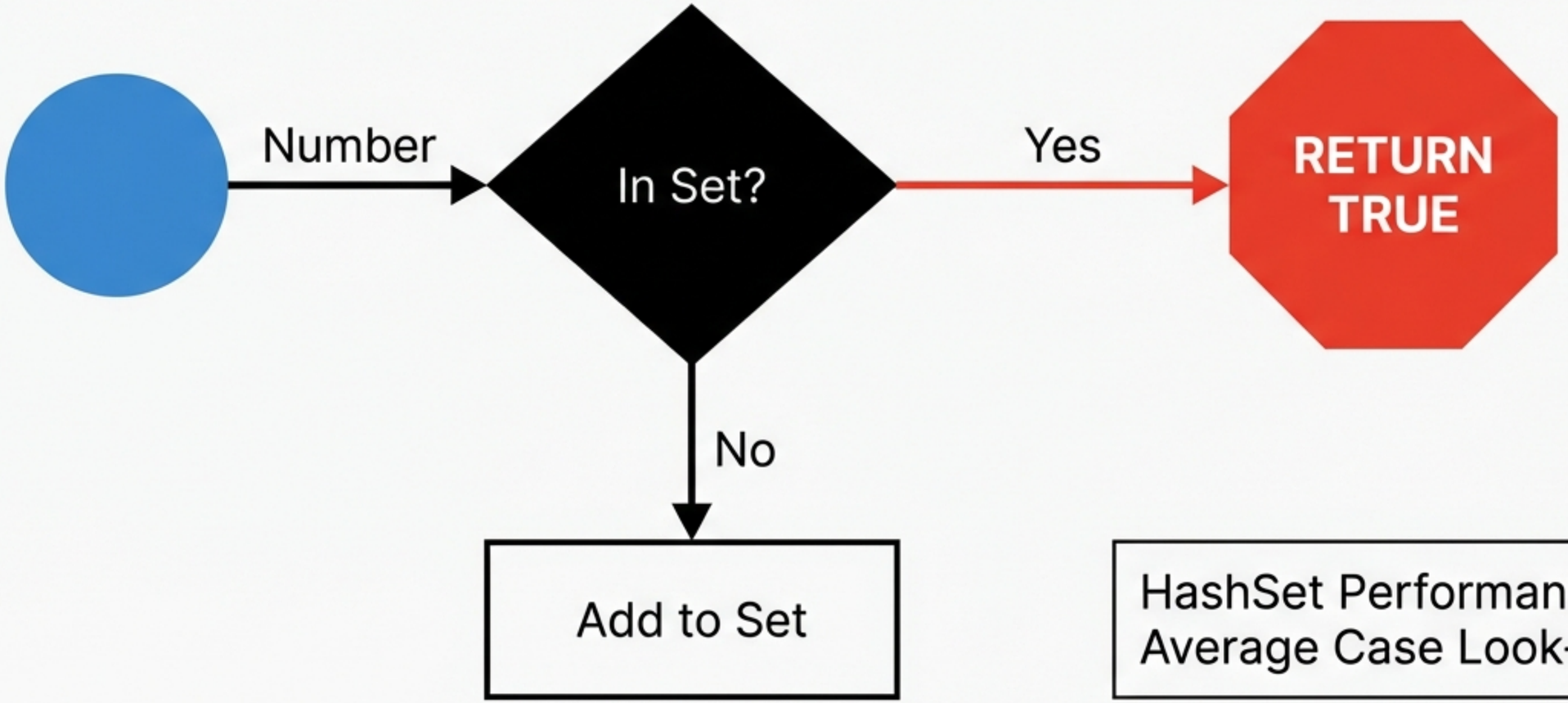
Look-Up Strategy



Stop at First Duplicate

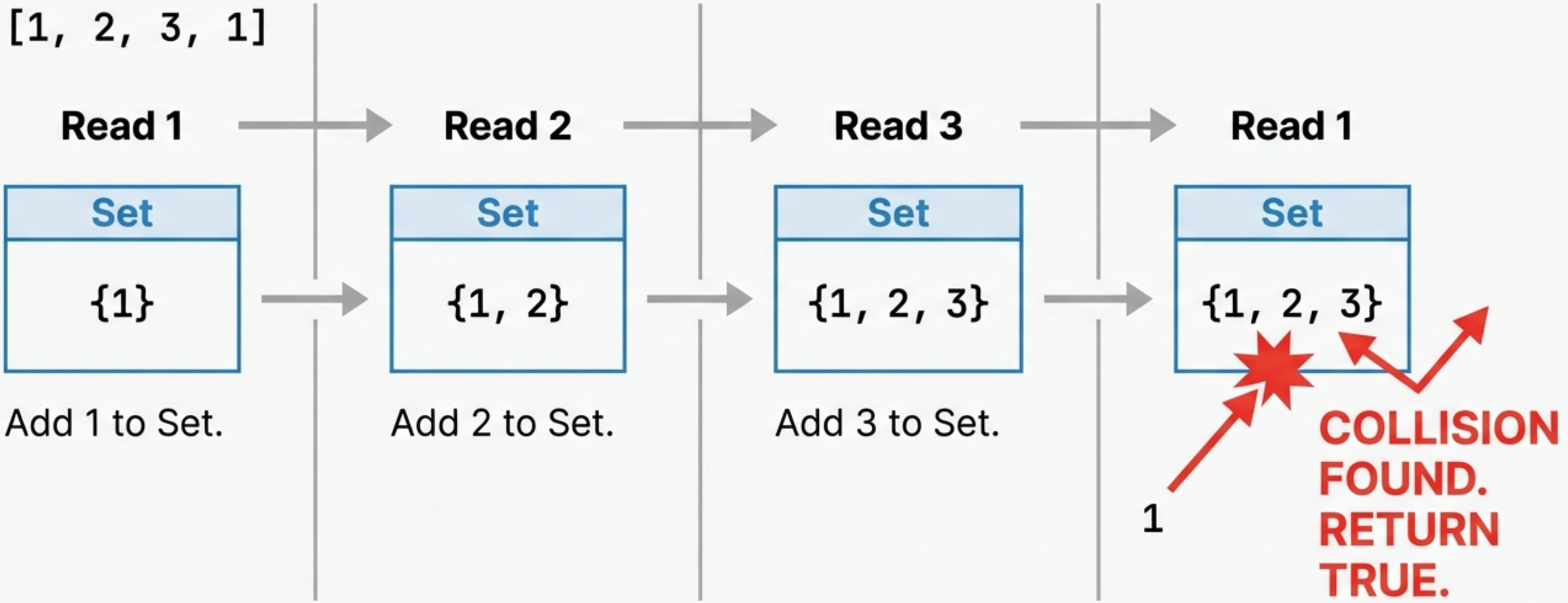
We do not need to count. We only need to detect.

The Gatekeeper: Using a HashSet



HashSet Performance:
Average Case Look-up: O(1)

The Algorithm in Action



Implementation: The Iterative Check

```python
def containsDuplicate(nums):
    seen = set()
    for num in nums:
        if num in seen:
            return True
        seen.add(num)
    return False
```

The O(1) Look-up

The Early Exit

Bonus:
The Pythonic One-Liner

```python
def containsDuplicate(nums):
    return len(nums) != len(set(nums))
```

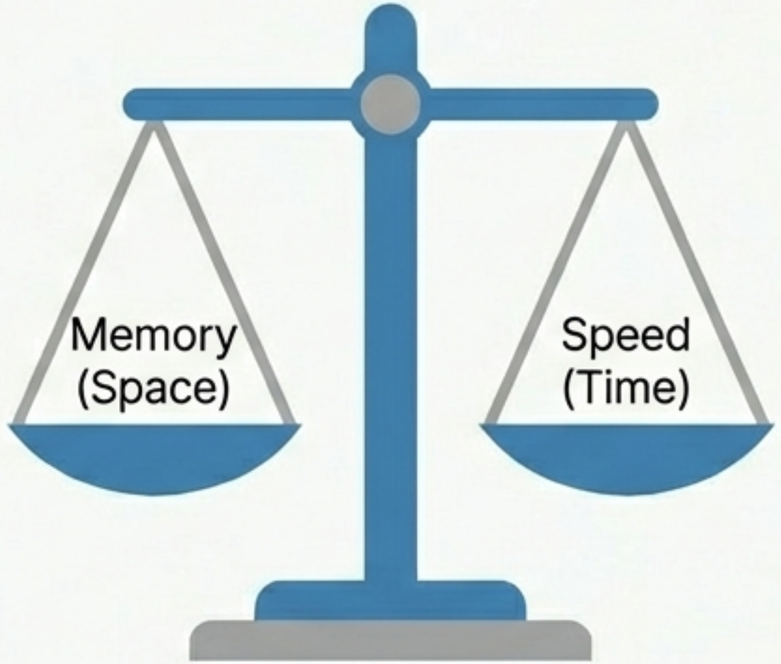
Length of List (4) != Length of Set (3)

Result: True

Elegant, but requires processing the entire list to build the set first.

Complexity Analysis: The Space-Time Trade-off

Scorecard	Value	Description
Time Complexity	`O(n)`	Linear Time: We traverse the list at most once.
Space Complexity	`O(n)`	Linear Space: In the worst case (distinct elements), we store every item.



We spend **memory** to buy time.

Key Takeaways & Pattern Mastery

The Pattern

Use HashSets for O(1) existence checks, not just storage.

The Strategy

Prioritize Early Exits. Don't scan if you can stop.

The Progress

Blind 75 Array Module: **1/10 Solved**.

Next Up: Valid Anagram.