## Problem Understanding

You're given an array nums of size n + 1, with all numbers from 1 to n. Exactly one number is **repeated at least once**.  
The goal is to return **that duplicate number**.

## Java Solution – Using Boolean Array

class Solution {

public int findDuplicate(int[] nums) {

int n = nums.length;

boolean[] dup = new boolean[n]; // index 0 to n-1 (nums[i] ∈ 1 to n-1)

for (int num : nums) {

if (dup[num]) {

return num; // already seen → duplicate

}

dup[num] = true;

}

return -1; // won't be hit as per problem guarantees

}

}

## Dry Run Using Table

### Input: nums = [1, 3, 4, 2, 2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | num | dup[] Before | dup[num] | Action |
| 1 | 1 | [F, F, F, F, F] | false | mark dup[1]=T |
| 2 | 3 | [F, T, F, F, F] | false | mark dup[3]=T |
| 3 | 4 | [F, T, F, T, F] | false | mark dup[4]=T |
| 4 | 2 | [F, T, F, T, T] | false | mark dup[2]=T |
| 5 | 2 | [F, T, T, T, T] | true | return 2 ✅ |

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n) |
| Space | **O(n)** ← uses boolean[] |

This solution is correct but **doesn't meet the problem's strict space constraint** (O(1) space required).

## Alternate Approaches

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Time | Space | Notes |
| ✅ **Floyd’s Cycle Detection** | O(n) | O(1) | ✅ optimal and allowed |
| 🔍 Binary Search on Values | O(nlogn) | O(1) | Clever use of Pigeonhole Principle |
| ❌ HashSet / Boolean Array | O(n) | O(n) | Easy but violates space constraint |

## Floyd's Tortoise and Hare – Cycle Detection

class Solution {

public int findDuplicate(int[] nums) {

int slow = nums[0];

int fast = nums[0];

// Step 1: Detect cycle (like Linked List cycle detection)

do {

slow = nums[slow];

fast = nums[nums[fast]];

} while (slow != fast);

// Step 2: Find the entrance to the cycle

slow = nums[0];

while (slow != fast) {

slow = nums[slow];

fast = nums[fast];

}

return slow;

}

}

## Dry Run Using Table

### Input: nums = [1, 3, 4, 2, 2]

1. **Index-based mapping like a Linked List:**

|  |  |
| --- | --- |
| Index | Value |
| 0 | 1 |
| 1 | 3 |
| 2 | 4 |
| 3 | 2 |
| 4 | 2 |

This makes a "linked list" path:  
0 → 1 → 3 → 2 → 4 → 2 → 4 → 2 → … (cycle between index 2 and 4)

1. **Floyd's Cycle Detection:**

|  |  |  |
| --- | --- | --- |
| Step | slow | fast |
| 0 | 1 | 1 |
| 1 | 3 | 2 |
| 2 | 2 | 2 ← meet (inside cycle) |

Now reset slow to 0 and move both one step:

|  |  |  |
| --- | --- | --- |
| Step | slow | fast |
| 0 | 0 | 2 |
| 1 | 1 | 4 |
| 2 | 3 | 2 |
| 3 | 2 | 2 ← meet at duplicate |

✅ **Duplicate = 2**

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n) |
| Space | O(1) |