287. Find the Duplicate Numbe

Problem Statement:

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and uses only constant extra space.

Example 1:

> **Input:** nums = [1,3,4,2,2]
> **Output:** 2

Example 2:

> **Input:** nums = [3,1,3,4,2]
> **Output:** 3

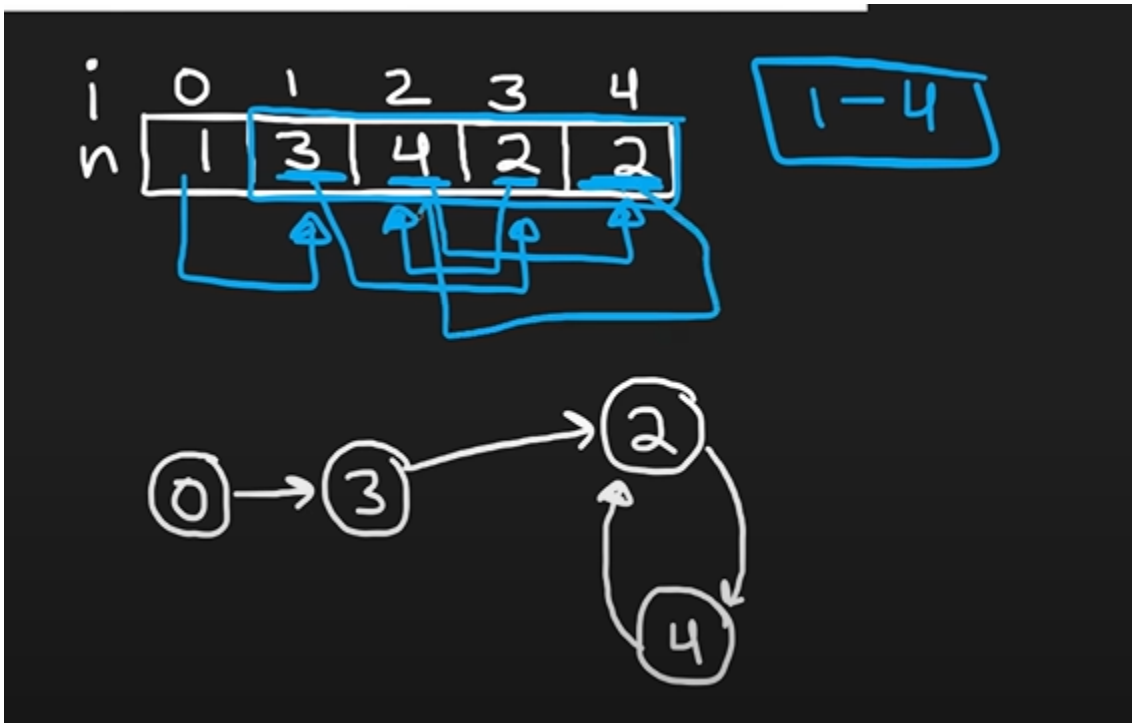
Example 3:

> **Input:** nums = [3,3,3,3,3]
> **Output:** 3

Solution:

1. Here this is a linked list problem

2. Given that our nums[i] will be in range of [1,n]



3. So now think each array value as pointer…map them like ..num[0] points to index 1 ,nums[1] points to index 3, nums[2] points to index 4…similarly nums[i] will points to the index i
4. Now we can see the cycle in the picture

5.

4. Fast-Slow Pointers Approach (Floyd's Cycle Detection)

Detailed Logic Behind the Fast-Slow Pointers Approach

Step 1: Initialize Pointers
First, we initialize two pointers, `slow` and `fast`, both set to `nums[0]`. This is the starting point for both pointers, and it's done to set up the conditions for Floyd's cycle detection algorithm.

Step 2: Detect a Cycle
In this step, we enter a `while` loop where the `slow` pointer moves one step at a time (`slow = nums[slow]`), while the `fast` pointer moves two steps (`fast = nums[nums[fast]]`). The loop continues until both pointers meet at some point within the cycle. Note that this meeting point is not necessarily the duplicate number; it's just a point inside the cycle.

Why does this happen? Because there is a duplicate number, there must be a cycle in the 'linked list' created by following `nums[i]` as next elements. Once a cycle is detected, the algorithm breaks out of this loop.

Step 3: Find the Start of the Cycle (Duplicate Number)
After identifying a meeting point inside the cycle, we reinitialize the `slow` pointer back to `nums[0]`. The `fast` pointer stays at the last meeting point. Now, we enter another `while` loop where both pointers move one step at a time. The reason behind this is mathematical: according to Floyd's cycle detection algorithm, when both pointers move at the same speed, they will eventually meet at the starting point of the cycle. This is the duplicate number we are looking for.

Step 4: Return the Duplicate Number
Finally, when the `slow` and `fast` pointers meet again, the meeting point will be the duplicate number, and we return it as the output.

6. Code:

Code Fast Slow Pointers

Python | Go | Rust | C++ | Java | JavaScript | PHP | C#

```python
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        slow = nums[0]
        fast = nums[0]

        while True:
            slow = nums[slow]
            fast = nums[nums[fast]]
            if slow == fast:
                break

        slow = nums[0]
        while slow != fast:
            slow = nums[slow]
            fast = nums[fast]

        return slow
```

Clear explanation :
https://leetcode.com/problems/find-the-duplicate-number/solutions/4062141/97-77-6-approaches-set-count-binary-search-fast-slow-mark-sort/