Find the Duplicate Number

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
```

Approach 1: Sort

Time Complexity: $O(n\log n)$

Space Complexity: $O(n)$

Intuition

In an unsorted array, duplicate elements may be scattered across the array. However, in a sorted array, duplicate numbers will be next to each other.

```
class Solution {
    public int findDuplicate(int[] nums) {
        Arrays.sort(nums);
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] == nums[i-1])
                return nums[i];
        }

        return -1;
    }
```

```
}
```

Approach 2: Set

Intuition

As we traverse the array, we need a way to "remember" values that we've seen. If we come across a number that we've seen before, we've found the duplicate. An efficient way to record the seen values is by adding each number to a set as we iterate over the $nums$ array.

Time Complexity: $O(n)$

Space Complexity: $O(n)$

```java
class Solution {
    public int findDuplicate(int[] nums) {
        Set<Integer> seen = new HashSet<Integer>();
        for (int num : nums) {
            if (seen.contains(num))
                return num;
            seen.add(num);
        }
        return -1;
    }
}
```

Approach 3: Negative Marking

Time Complexity: $O(n)$

Space Complexity: $O(1)$

Intuition

There are $n+1$ positive numbers in the array ($nums$) (all in the range $[1,n]$). Since the array only contains positive integers, we can track each number (num) that has been seen before by flipping the sign of the number located at index $|num|$, where $||$ denotes absolute value.

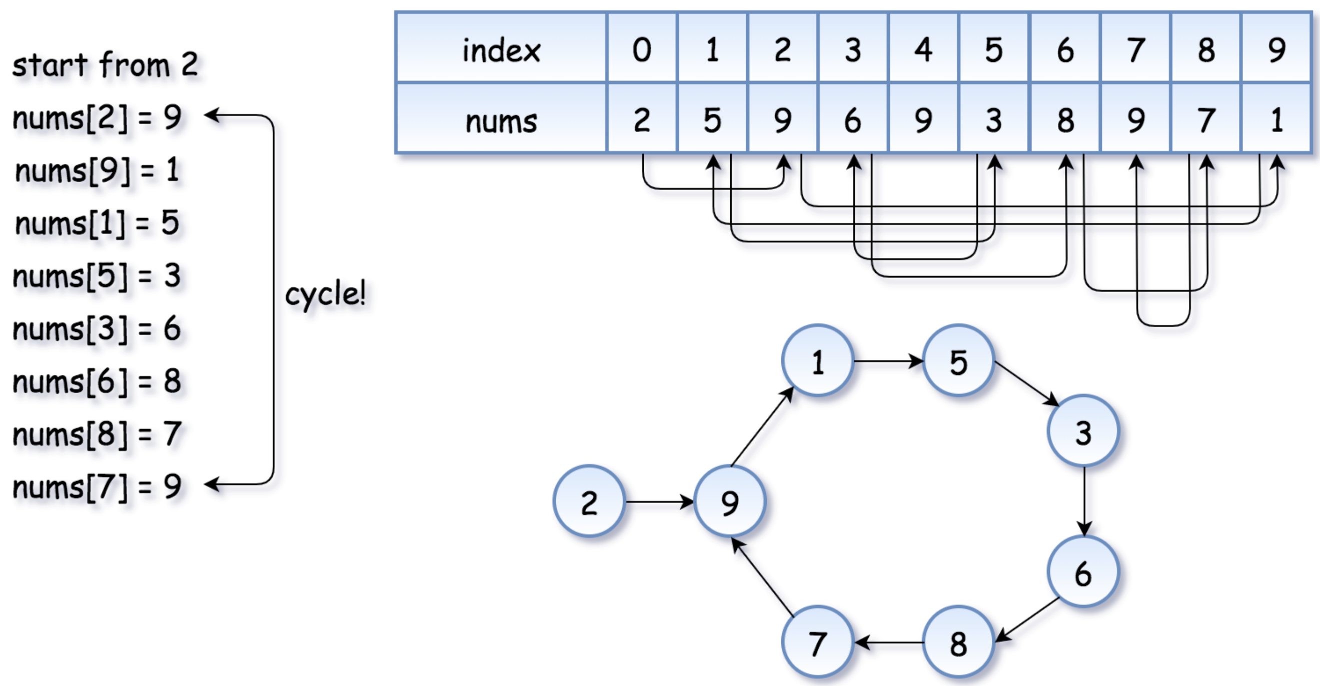
For example, if the input array is $[1,3,3,2]$, then for 1, flip the number at index 1, making the array $[1,-3,3,2]$. Next, for -3 flip the number at index 3, making the array $[1,-3,3,-2]$. Finally, when we reach the second 3, we'll notice that $nums[3]$ is already negative, indicating that 3 has been seen before and hence is the duplicate number.

```
class Solution {

    public int findDuplicate(int[] nums) {

        int duplicate = -1;

        for (int i = 0; i < nums.length; i++) {

            int cur = Math.abs(nums[i]);

            if (nums[cur] < 0) {

                duplicate = cur;

                break;

            }

            nums[cur] *= -1;

        }



        // Restore numbers

        for (int i = 0; i < nums.length; i++)

            nums[i] = Math.abs(nums[i]);



        return duplicate;

    }

}
```

Approach 4: Floyd's Tortoise and Hare (Cycle Detection)

Time Complexity: $O(n)$

Space Complexity: $O(1)$




```
class Solution {
    public int findDuplicate(int[] nums) {

        // Find the intersection point of the two runners.
        int tortoise = nums[0];
        int hare = nums[0];

        do {
            tortoise = nums[tortoise];
            hare = nums[nums[hare]];
        } while (tortoise != hare);
```

```
        // Find the "entrance" to the cycle.
        tortoise = nums[0];

        while (tortoise != hare) {
            tortoise = nums[tortoise];
            hare = nums[hare];
        }

        return hare;
    }
}
```