Previous Next

137. Single Number II

Aug. 11, 2019 | 26.4K views

Average Rating: 4.30 (56 votes)

Given a **non-empty** array of integers, every element appears *three* times except for one, which appears exactly once. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

```
Input: [2,2,3,2]
Output: 3
```

Example 2:

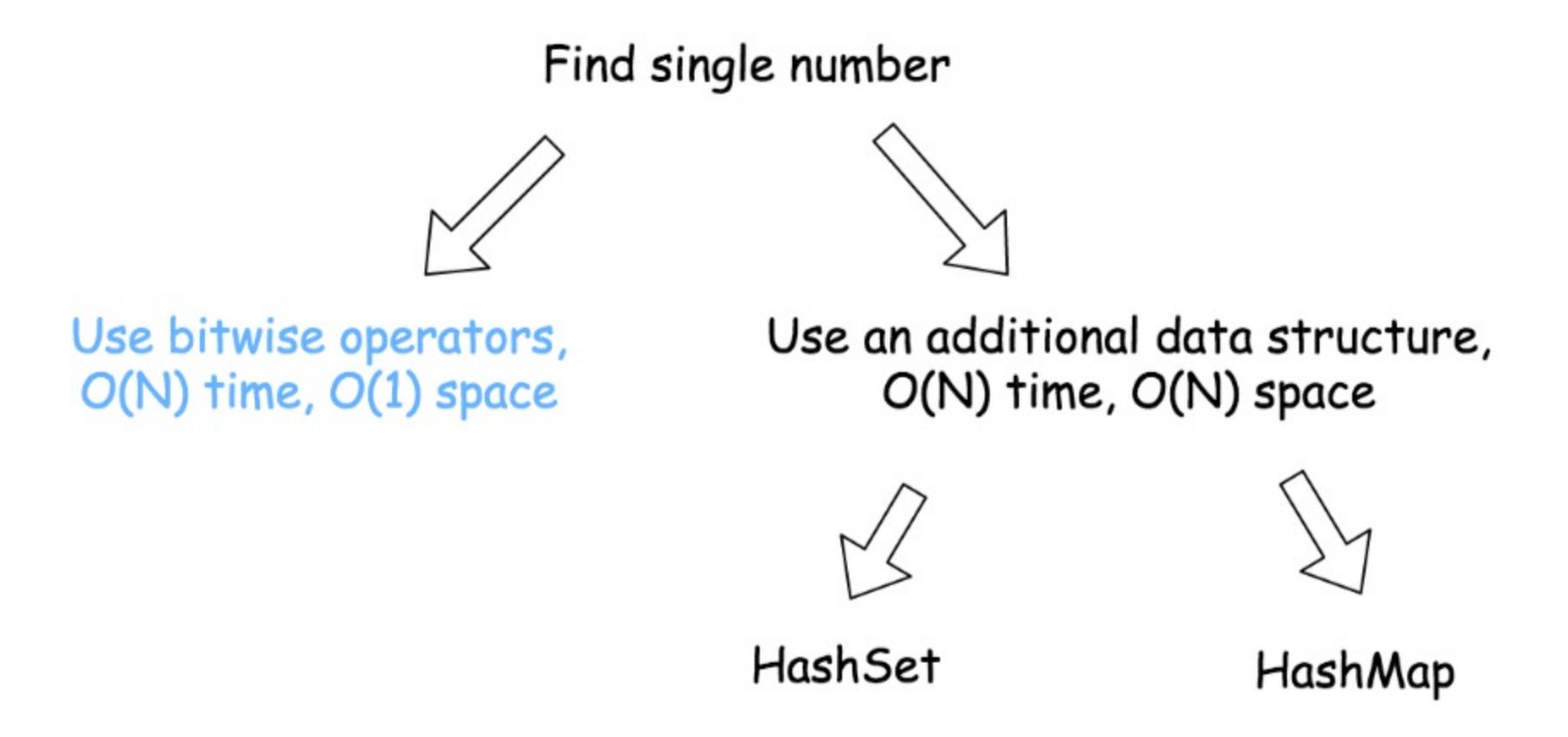
```
Input: [0,1,0,1,0,1,99]
Output: 99
```

Solution

Overview

The problem seems to be quite simple and one could solve it in $\mathcal{O}(N)$ time and $\mathcal{O}(N)$ space by using an additional data structure like set or hashmap.

The real game starts at the moment when Google interviewer (the problem is quite popular at Google the last six months) asks you to solve the problem in a constant space, testing if you are OK with bitwise operators.



Approach 1: HashSet

The idea is to convert an input array into hashset and then to compare the tripled sum of the set with the array sum

$$3 \times (a + b + c) - (a + a + a + b + b + b + c) = 2c$$

Implementation

Java Python Copy

```
class Solution:
    def singleNumber(self, nums):
        return (3 * sum(set(nums)) - sum(nums)) // 2
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to iterate over the input array.
- Space complexity : $\mathcal{O}(N)$ to keep the set of $N/3$ elements.

Approach 2: HashMap

Let's iterate over the input array to count the frequency of each number, and then return an element with a frequency 1.

Implementation

Java Python Copy

```
from collections import Counter
class Solution:
    def singleNumber(self, nums):
        hashmap = Counter(nums)

        for k in hashmap.keys():
            if hashmap[k] == 1:
                return k
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to iterate over the input array.
- Space complexity : $\mathcal{O}(N)$ to keep the hashmap of $N/3$ elements.

Approach 3: Bitwise Operators : NOT, AND and XOR

Intuition

Now let's discuss $\mathcal{O}(1)$ space solution by using three bitwise operators

$$\begin{aligned} \sim x & \quad \textrm{that means} \quad \textrm{bitwise NOT} \\ x \& y & \quad \textrm{that means} \quad \textrm{bitwise AND} \\ x \oplus y & \quad \textrm{that means} \quad \textrm{bitwise XOR} \end{aligned}$$

XOR

Let's start from XOR operator which could be used to detect the bit which appears odd number of times: 1, 3, 5, etc.

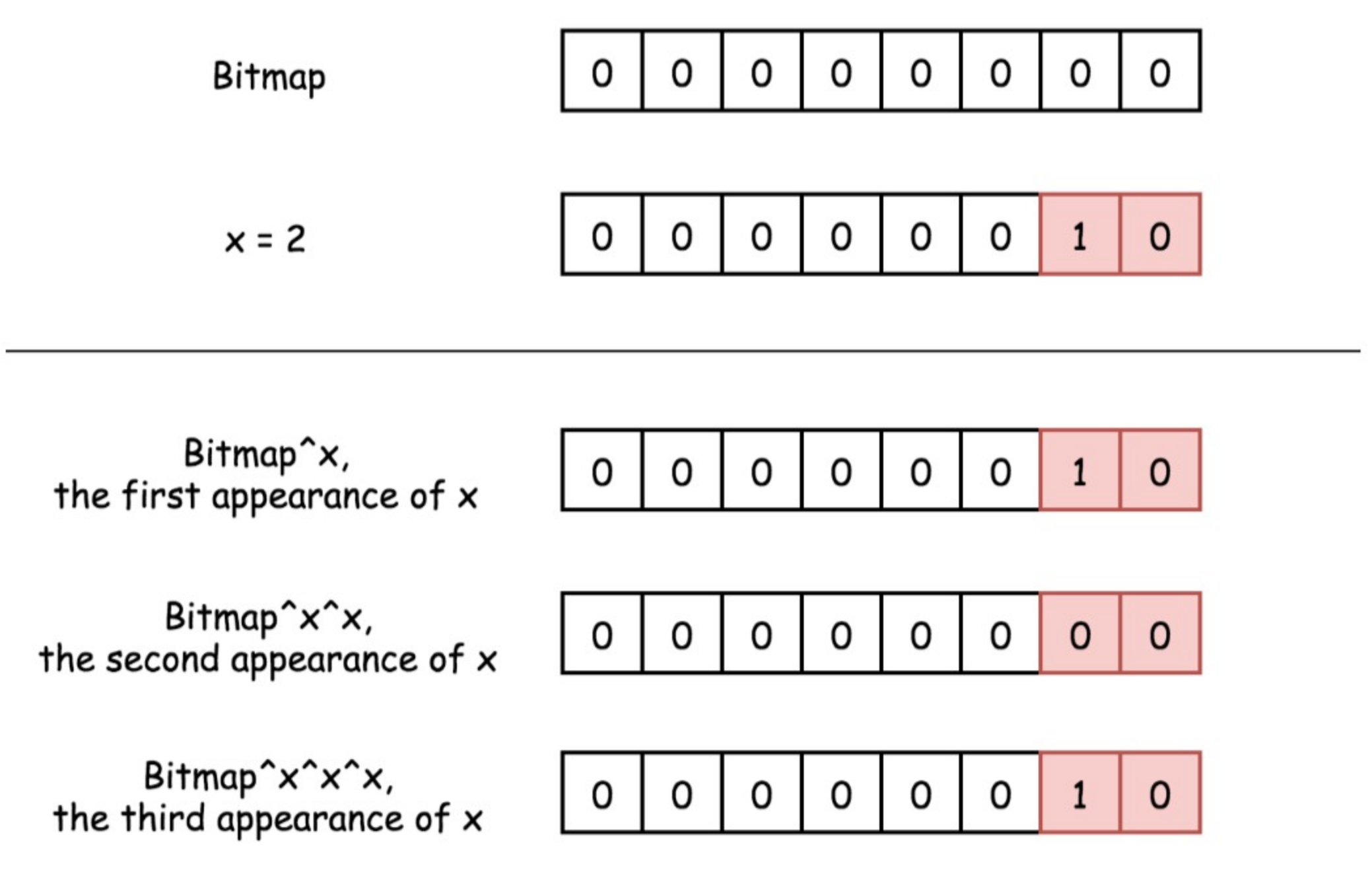
XOR of zero and a bit results in that bit

$$0 \oplus x = x$$

XOR of two equal bits (even if they are zeros) results in a zero

$$x \oplus x = 0$$

and so on and so forth, i.e. one could see the bit in a bitmask only if it appears odd number of times.



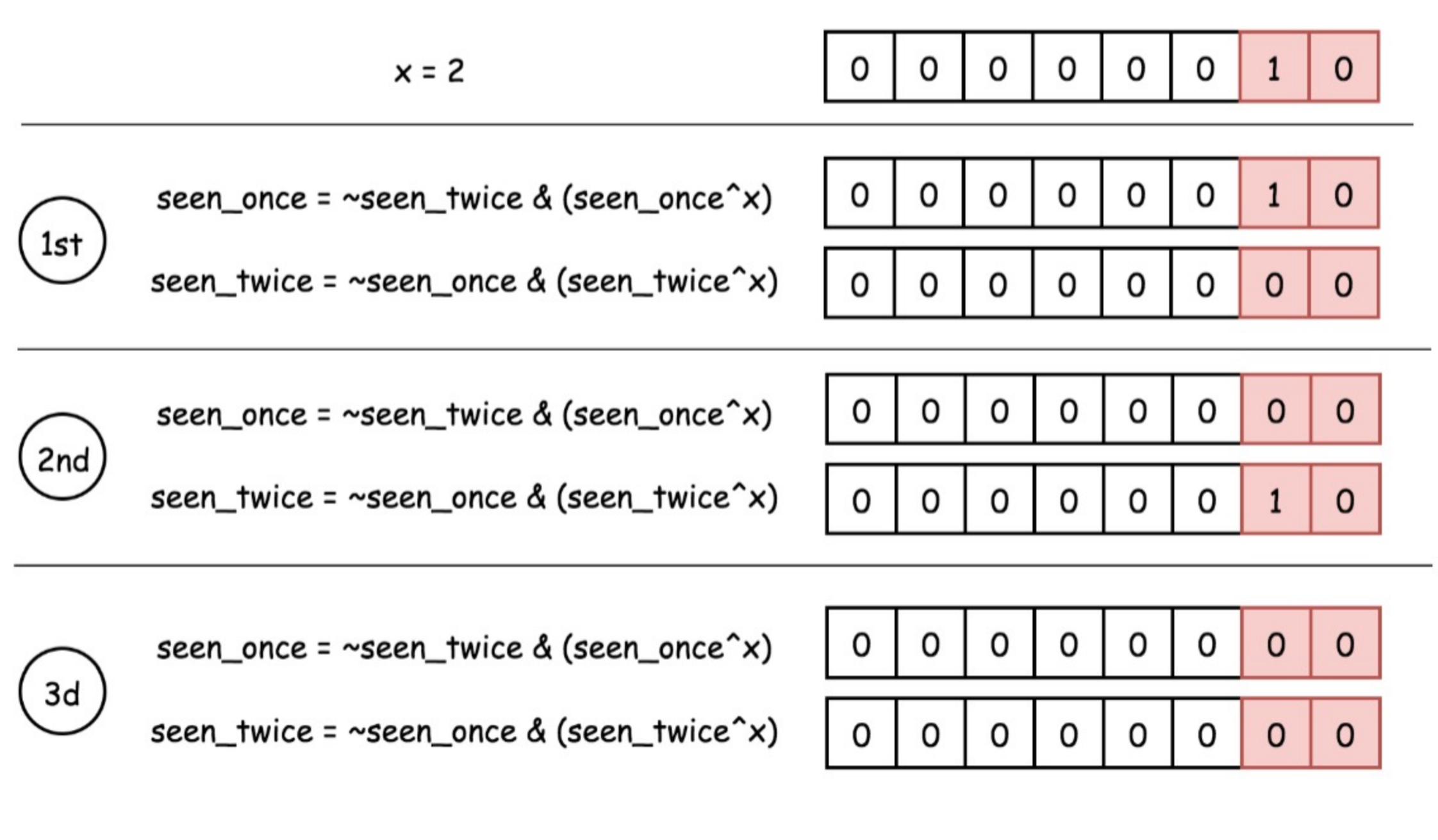
That's already great, so one could detect the bit which appears once, and the bit which appears three times. The problem is to distinguish between these two situations.

AND and NOT

To separate number that appears once from a number that appears three times let's use two bitmasks instead of one: `seen_once` and `seen_twice`.

The idea is to

- change `seen_once` only if `seen_twice` is unchanged
- change `seen_twice` only if `seen_once` is unchanged



This way bitmask `seen_once` will keep only the number which appears once and not the numbers which appear three times.

Implementation

Java Python Copy

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        seen_once = seen_twice = 0

        for num in nums:
            # first appearance:
            # add num to seen_once
            # don't add to seen_twice because of presence in seen_once

            # second appearance:
            # remove num from seen_once
            # add num to seen_twice

            # third appearance:
            # don't add to seen_once because of presence in seen_twice
            # remove num from seen_twice
            seen_once = ~seen_twice & (seen_once ^ num)
            seen_twice = ~seen_once & (seen_twice ^ num)

        return seen_once
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to iterate over the input array.
- Space complexity : $\mathcal{O}(1)$ since no additional data structures are allocated.

Rate this article:

Previous Next

Comments: 24 Sort By

Type comment here... (Markdown is supported)

Preview Post

kenanlv ★ 107 August 14, 2019 2:35 AM
How could you come up with solution 3 during an interview???
101 Share Reply
SHOW 7 REPLIES

casd82 ★ 132 August 17, 2019 6:27 AM
This is art.
24 Share Reply

Zizhen_Huang ★ 86 December 28, 2019 12:10 PM
I think 90% people could not come up with such a bitwise approach in an interview
16 Share Reply
SHOW 5 REPLIES

lenchen1112 ★ 976 December 5, 2019 3:53 PM
For bitwise operations, I've wrote a medium article to explain how can we get the formula by truth table:
https://medium.com/@lenchen/leetcode-137-single-number-ii-31af98b0f462
21 Share Reply
SHOW 6 REPLIES

petulla ★ 29 February 8, 2020 4:22 AM
The question is poorly worded. It says no additional space, which implies modifying the array in place. The last answer uses constant additional space.
15 Share Reply
SHOW 2 REPLIES

hello_world_cn ★ 279 June 2, 2020 8:50 AM

My explanation for Approach 3: Bitwise Operators : NOT, AND and XOR:

Read More
10 Share Reply
SHOW 3 REPLIES

akhashr ★ 9 April 15, 2020 9:09 AM
Badly written article (esp 3rd method). May be helpful to folks who are already proficient but doesn't help the majority of the folks who are more of beginner to intermediate state.
5 Share Reply

szipan ★ 8 August 15, 2019 1:26 PM
For the bitwise algorithm, I think the commutative property of the two operators (seenOnce and seenTwice) should be proved first, i.e, why applying them to a number sequence in two different orders generates the same result. Or else, I think an interviewee only solves the problem by chance.
8 Share Reply
SHOW 4 REPLIES

dushuangli0835 ★ 8 September 14, 2019 11:03 AM
First two didn't match the requirement. they use O(n/3) extra space for hashmap.
Sorting also doesn't match, because O(nlogn) larger than O(n).

this question have to use part of the quicksort. every time find the position of pivot.
use the pivot final position to check whether the single number on its left or right.
Read More
4 Share Reply

kkzeng ★ 223 August 19, 2019 10:33 PM
Really elegant solution in the last approach. It's easy to see why this would work in a case like [1, 1, 1, 2, 2, 2, 3] and I know that order doesn't affect this solution i.e. [1, 2, 2, 1, 3, 1, 2] but I'm not sure I understand why it doesn't affect the solution?
4 Share Reply
SHOW 2 REPLIES

‹ 1 2 3 ›