41. First Missing Positive

Feb. 2, 2019 | 46.9K views

Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

```
Input: [1,2,0]
Output: 3
```

Example 2:

```
Input: [3,4,-1,1]
Output: 2
```

Example 3:

```
Input: [7,8,9,11,12]
Output: 1
```

Note:

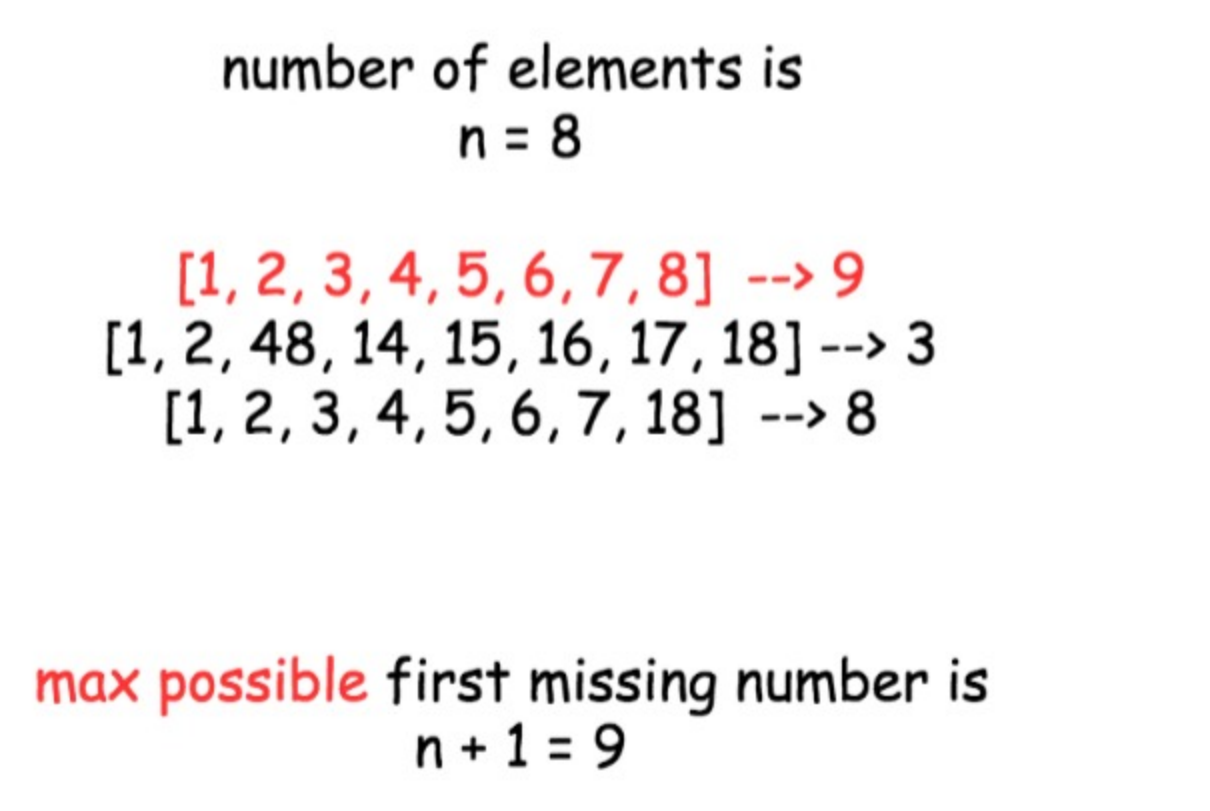
Your algorithm should run in $O(n)$ time and uses constant extra space.

Solution

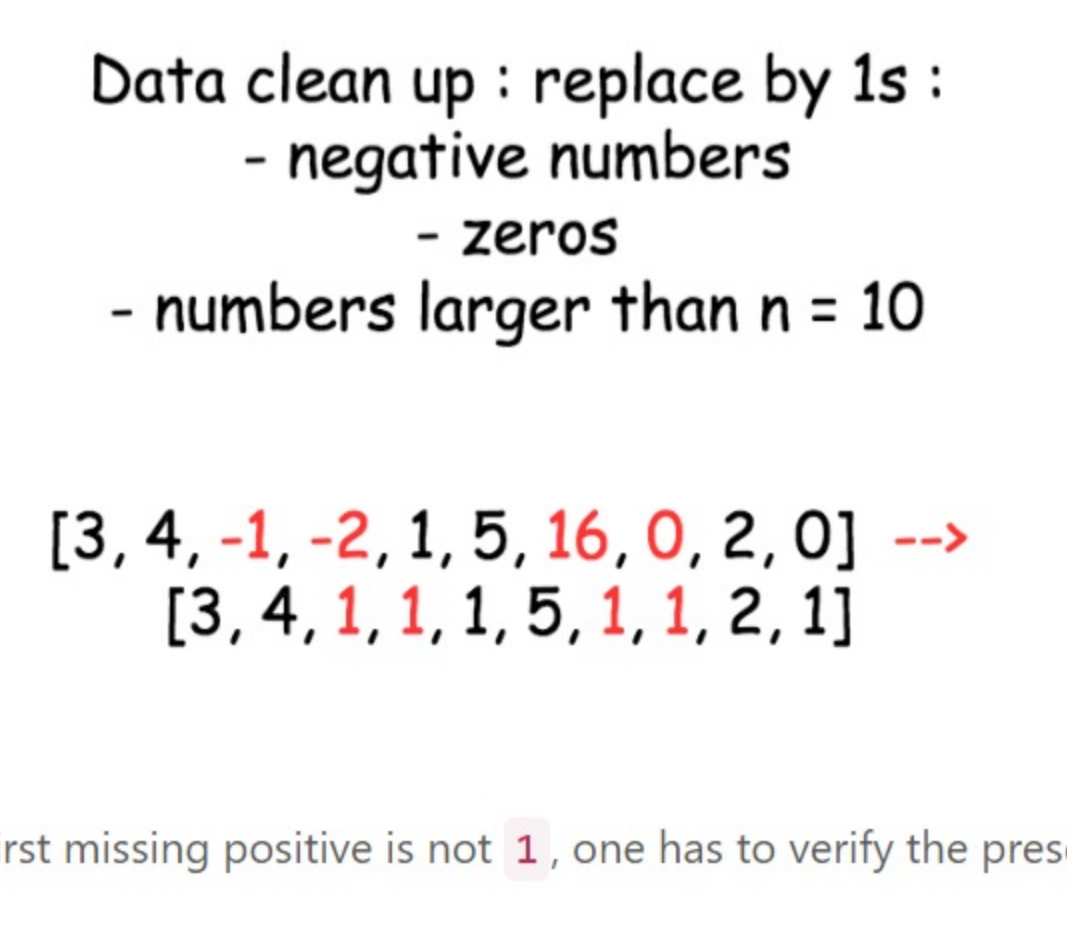
Approach 1: Index as a hash key.

Data clean up

First of all let's get rid of negative numbers and zeros since there is no need of them. One could get rid of all numbers larger than `n` as well, since the first missing positive is for sure smaller or equal to `n + 1`. The case when the first missing positive is equal to `n + 1` will be treated separately.



What does it mean - to get rid of, if one has to keep $\mathcal{O}(N)$ time complexity and hence could not pop unwanted elements out? Let's just replace all these by `1`s.

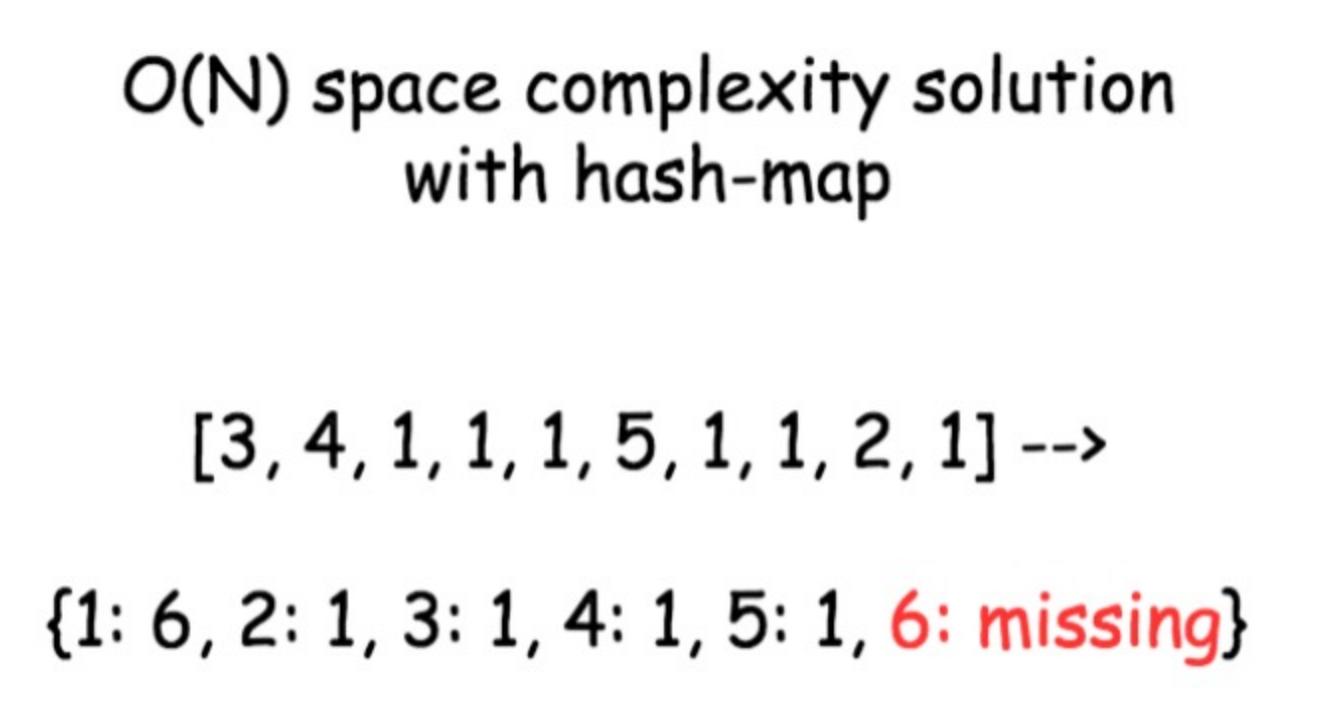


To ensure that the first missing positive is not `1`, one has to verify the presence of `1` before proceeding to this operation.

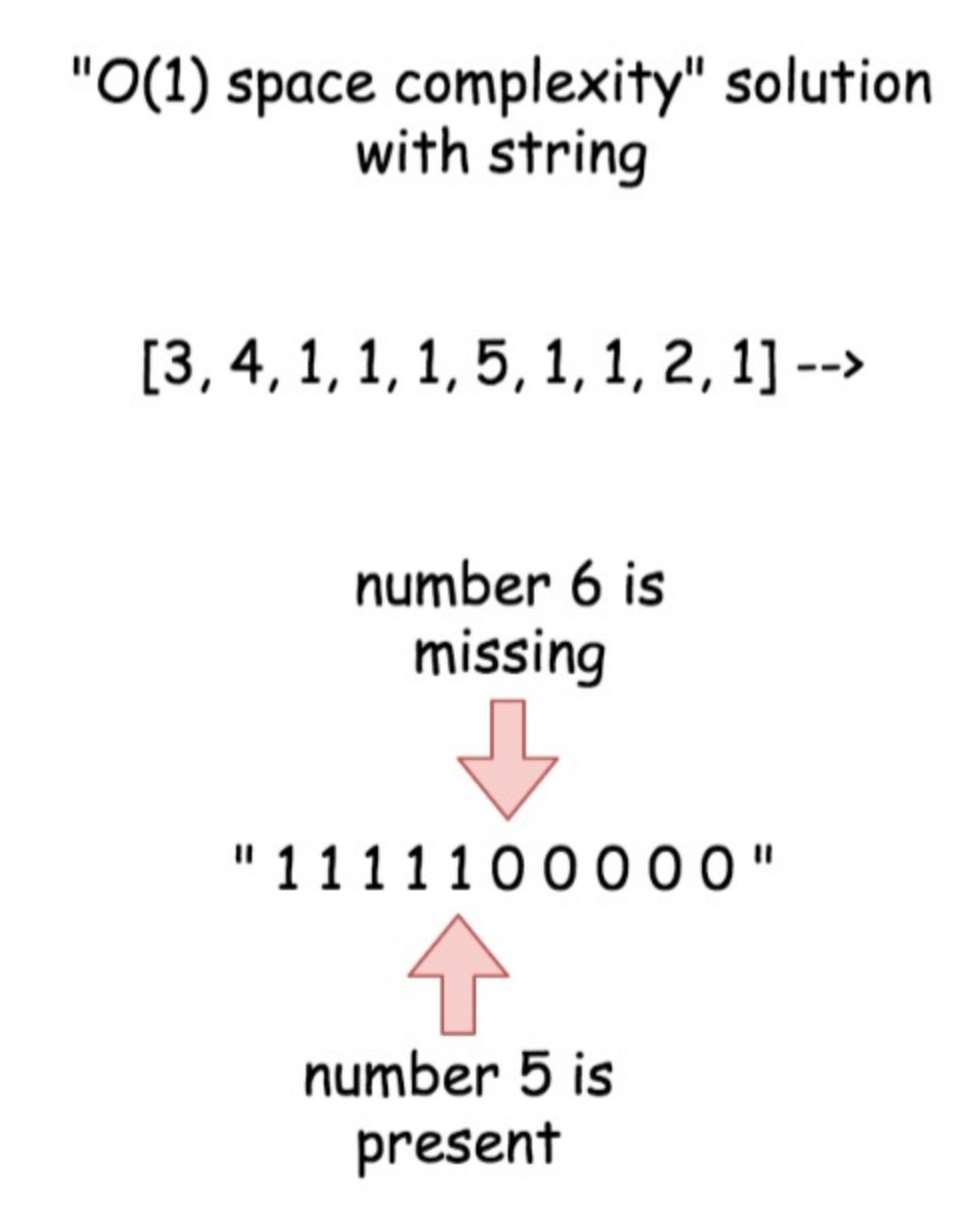
How to solve in-place

Now there we have an array which contains only positive numbers in a range from `1` to `n`, and the problem is to find a first missing positive in $\mathcal{O}(N)$ time and constant space.

That would be simple, if one would be allowed to have a hash-map `positive number -> its presence` for the array.



Sort of "dirty workaround" solution would be to allocate a string `hash_str` with `n` zeros, and use it as a sort of hash map by changing `hash_str[i]` to `1` each time one meets number `i` in the array.

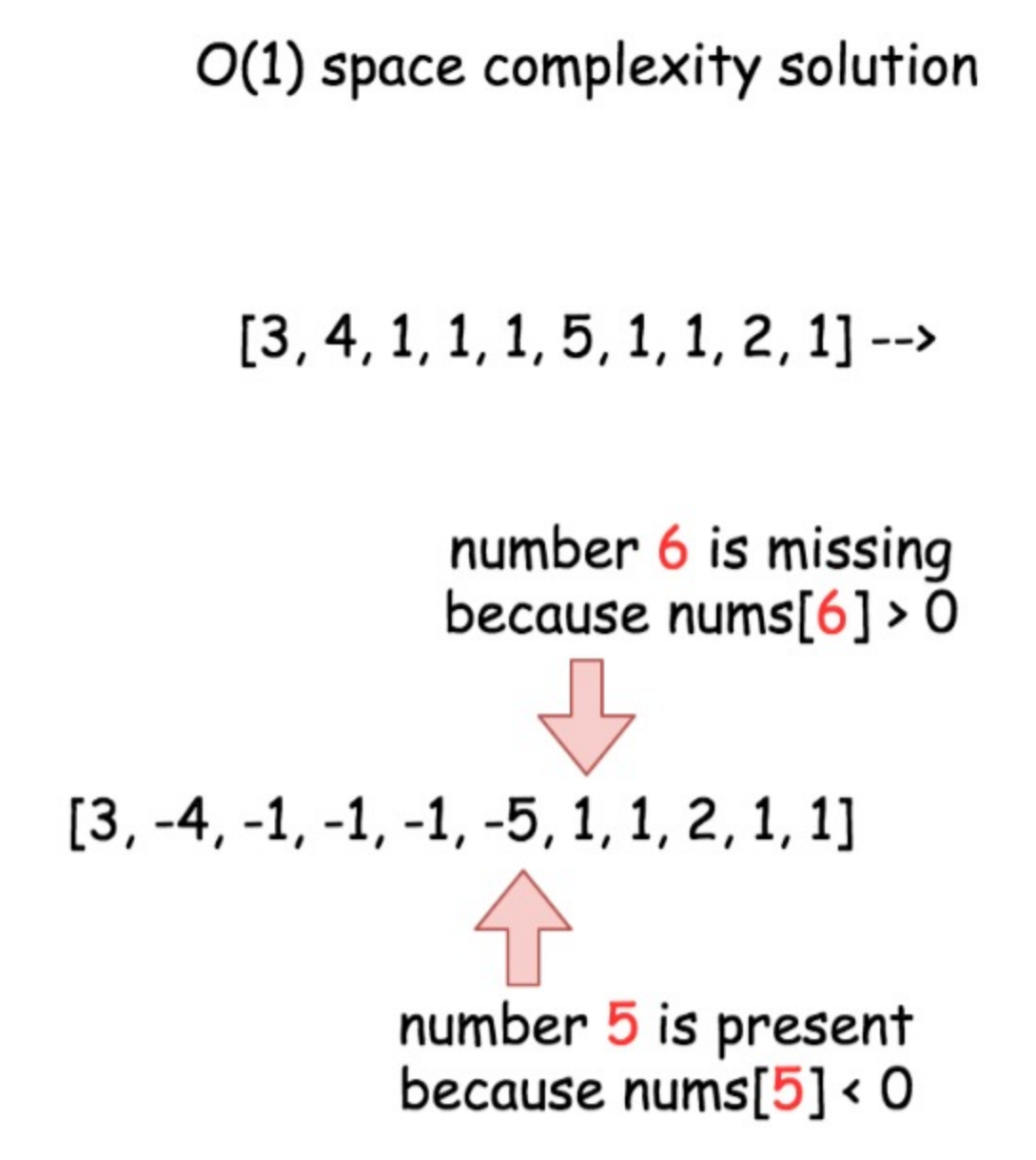


Let's not use this solution, but just take away a pretty nice idea *to use index as a hash-key* for a positive number.

The final idea is to *use index in nums as a hash key* and *sign of the element as a hash value* which is presence detector.

> For example, negative sign of `nums[2]` element means that number `2` is present in `nums`. The positive sign of `nums[3]` element means that number `3` is not present (missing) in `nums`.

To achieve that let's walk along the array (which after clean up contains only positive numbers), check each element value `elem` and change the sign of element `nums[elem]` to negative to mark that number `elem` is present in `nums`. Be careful with duplicates and ensure that the sign was changed only once.

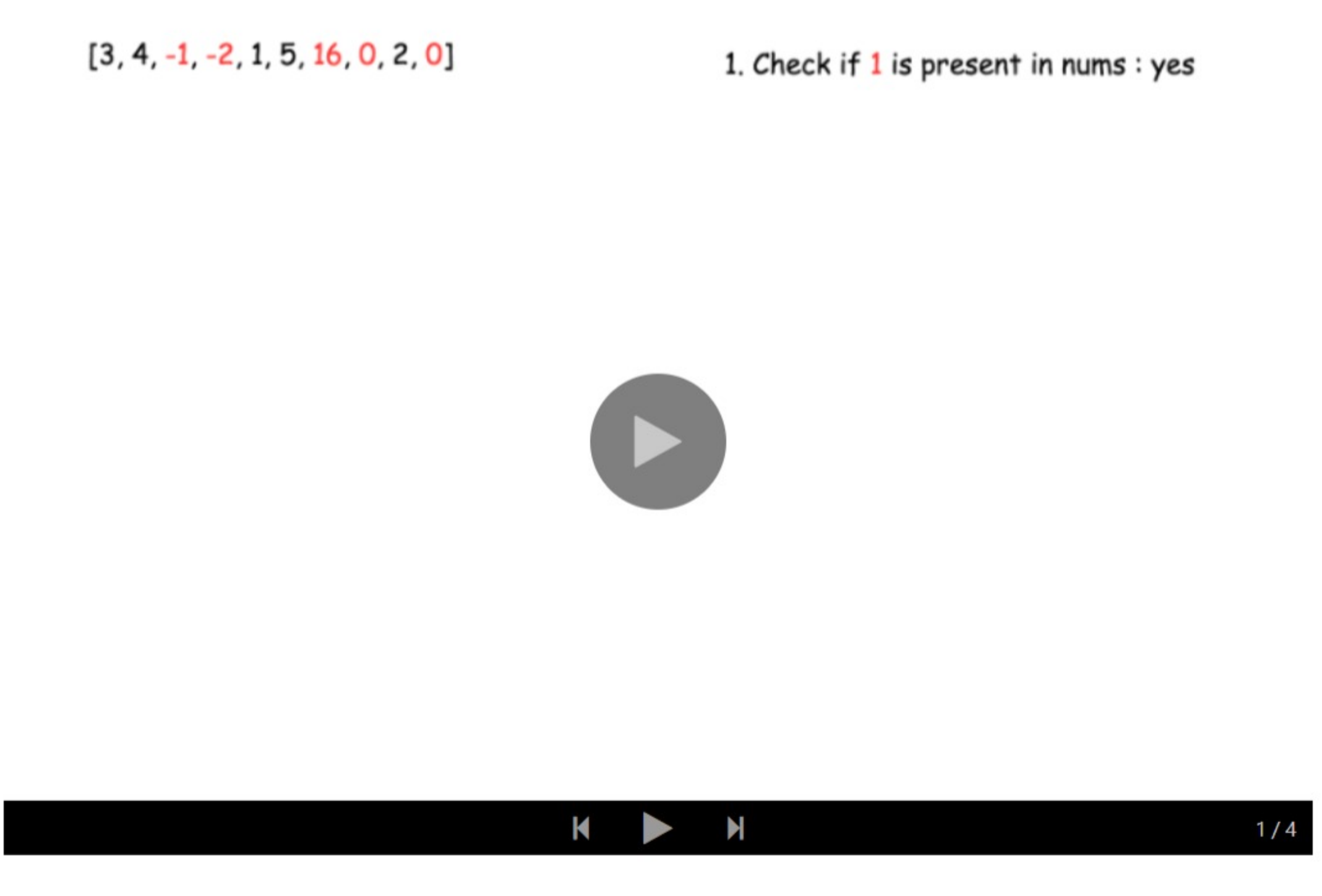


Algorithm

Now everything is ready to write down the algorithm.

- Check if `1` is present in the array. If not, you're done and `1` is the answer.
- If `nums = [1]`, the answer is `2`.
- Replace negative numbers, zeros, and numbers larger than `n` by `1`s.
- Walk along the array. Change the sign of a-th element if you meet number `a`. Be careful with duplicates : do sign change only once. Use index `0` to save an information about presence of number `n` since index `n` is not available.
- Walk again along the array. Return the index of the first positive element.
- If `nums[0] > 0` return `n`.
- If on the previous step you didn't find the positive element in nums, that means that the answer is `n + 1`.

Implementation



Java | Python | Copy

```python
class Solution:
    def firstMissingPositive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n = len(nums)

        # Base case.
        if 1 not in nums:
            return 1

        # nums = [1]
        if n == 1:
            return 2

        # Replace negative numbers, zeros,
        # and numbers larger than n by 1s.
        # After this convertion nums will contain
        # only positive numbers.
        for i in range(n):
            if nums[i] <= 0 or nums[i] > n:
                nums[i] = 1

        # Use index as a hash key and number sign as a presence detector.
        # For example, if nums[1] is negative that means that number `1`
        # is present in the array.
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since all we do here is four walks along the array of length `N`.
- Space complexity : $\mathcal{O}(1)$ since this is a constant space solution.