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.

number of elements is
n = 8

[1, 2, 3, 4, 5, 6, 7, 8] --> 9
[1, 2, 48, 14, 15, 16, 17, 18] --> 3
[1, 2, 3, 4, 5, 6, 7, 18] --> 8

max possible first missing number is
n + 1 = 9

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.

Data clean up : replace by 1s :
- negative numbers
- zeros
- numbers larger than n = 10

[3, 4, -1, -2, 1, 5, 16, 0, 2, 0] -->
[3, 4, 1, 1, 1, 5, 1, 1, 2, 1]

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.

O(N) space complexity solution
with hash-map

[3, 4, 1, 1, 1, 5, 1, 1, 2, 1] -->

{1: 6, 2: 1, 3: 1, 4: 1, 5: 1, 6: missing}

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.

"O(1) space complexity" solution
with string

[3, 4, 1, 1, 1, 5, 1, 1, 2, 1] -->

number 6 is
missing

"1 1 1 1 1 0 0 0 0 0"

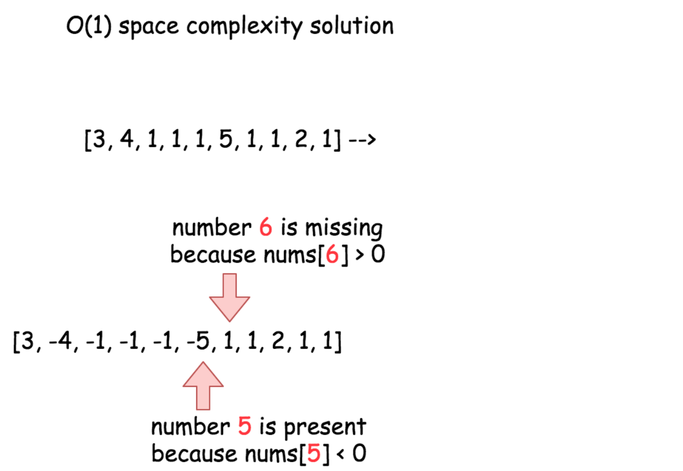
number 5 is
present

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.

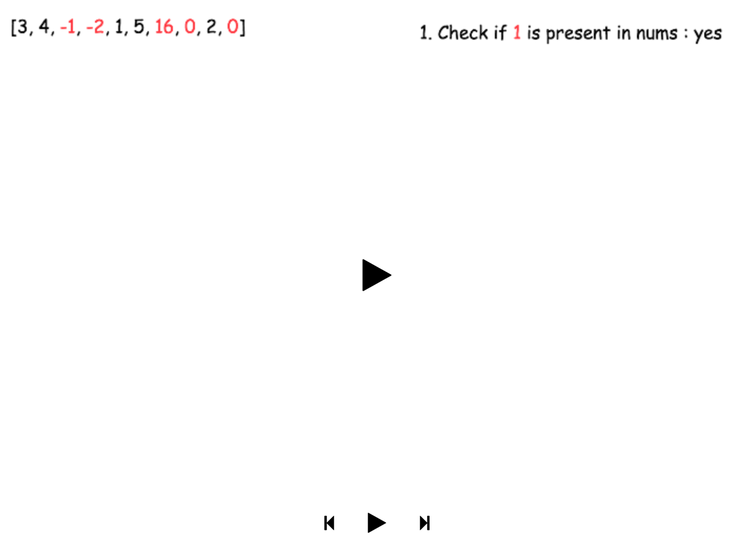


Algorithm

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`
```

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.

Comments: 49

Sort By

Type comment here... (Markdown is supported)

Preview | Post

(/coder_xyzzy)

coder_xyzzy (/coder_xyzzy) ★167 March 27, 2019 8:19 PM

Nice solution, but it's possible to do this in one pass, considering each value at most twice, and not destroying the contents of the array (though changing its order).

```
class Solution {
public:
```

Read More

40 Share Reply

SHOW 7 REPLIES

(/karbayev)

karbayev (/karbayev) ★45 May 5, 2020 10:20 AM

Solution with a string of length of n is not O(1) space complexity, it's O(n)

34 Share Reply

SHOW 1 REPLY

(/haomin0817)

Haomin0817 (/haomin0817) ★31 April 26, 2020 7:38 PM

Another O(n) solution use cycle sort:

```
def firstMissingPositive(self, nums: List[int]) -> int:
    i = 0
    n = len(nums)
```

Read More

18 Share Reply

SHOW 3 REPLIES

(/parag_meshram)

parag_meshram (/parag_meshram) ★24 April 24, 2020 2:02 PM

The problem should specify that it is ok if you alter the existing array because to my surprise the LeetCode solution doesn't care about modifying the existing array.

13 Share Reply

SHOW 1 REPLY

(/nickyflash)

nickyflash (/nickyflash) ★34 August 3, 2020 5:01 PM

This doesn't really count as O(1) because it relies on destructively modifying the input, and in almost any real life use case, you never want your functions to modify the input data.

10 Share Reply

(/totsubo)

totsubo (/totsubo) ★944 June 24, 2019 8:32 PM

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

This isn't obvious and could use some more explanation.

Read More

11 Share Reply

SHOW 2 REPLIES

(/manchesterunited)

manchesterunited (/manchesterunited) ★5 February 2, 2019 10:53 PM

Should the following statement:
If array contains only one element and it's **not** 1, the answer is 2.
be:
If array contains only one element and it's 1, the answer is 2.
?

Read More

3 Share Reply

SHOW 2 REPLIES

(/nqm149)

nqm149 (/nqm149) ★16 May 7, 2020 2:13 AM

we can apply cyclic replacement which cost O(n) to swap and put all numbers in their correct positions in array

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

Read More

4 Share Reply

SHOW 2 REPLIES

(/manchesterunited)

manchesterunited (/manchesterunited) ★5 February 2, 2019 11:27 PM

And ArrayIndexOutOfBoundsException while processing {1,3,3};

Should:

```
else if (nums[a] > 0)
nums[a] *= -1;
```

Read More

2 Share Reply

SHOW 4 REPLIES

(/harvey2015)

harvey2015 (/harvey2015) ★4 April 2, 2019 2:25 PM

two pointers

for each position i, find if i+1 exists in the array
j increases from i to len(nums), for each step it tries to find i+1 by swaping
every element is at most swapped 1 time, so o(n) time complexity

Read More

1 Share Reply

SHOW 1 REPLY

Notes

Notes