Previous (/articles/binary-search-tree-iterator/) Next (/articles/ipo/)

1060. Missing Element in Sorted Array (/problems/missing-element-in-sorted-array/)

June 12, 2019 | 15.3K views

Average Rating: 4.10 (20 votes)

Given a sorted array `A` of **unique** numbers, find the `K-th` missing number starting from the leftmost number of the array.

Example 1:

```
Input: A = [4,7,9,10], K = 1
Output: 5
Explanation:
The first missing number is 5.
```

Example 2:

```
Input: A = [4,7,9,10], K = 3
Output: 8
Explanation:
The missing numbers are [5,6,8,...], hence the third missing number is 8.
```

Example 3:

```
Input: A = [1,2,4], K = 3
Output: 6
Explanation:
The missing numbers are [3,5,6,7,...], hence the third missing number is 6.
```

Note:

1. `1 <= A.length <= 50000`
2. `1 <= A[i] <= 1e7`
3. `1 <= K <= 1e8`

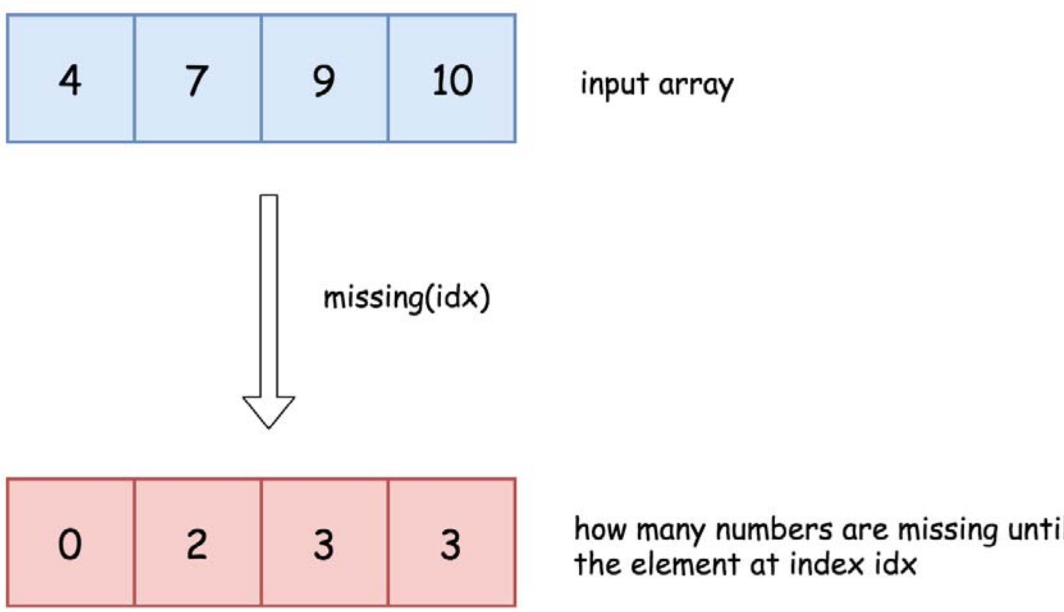
Solution

Approach 1: One Pass

Intuition

The problem is similar to First Missing Positive (https://leetcode.com/articles/first-missing-positive/) and the naive idea would be to solve it in a similar way by one pass approach.

Let's first assume that one has a function `missing(idx)` that returns how many numbers are missing until the element at index `idx` .

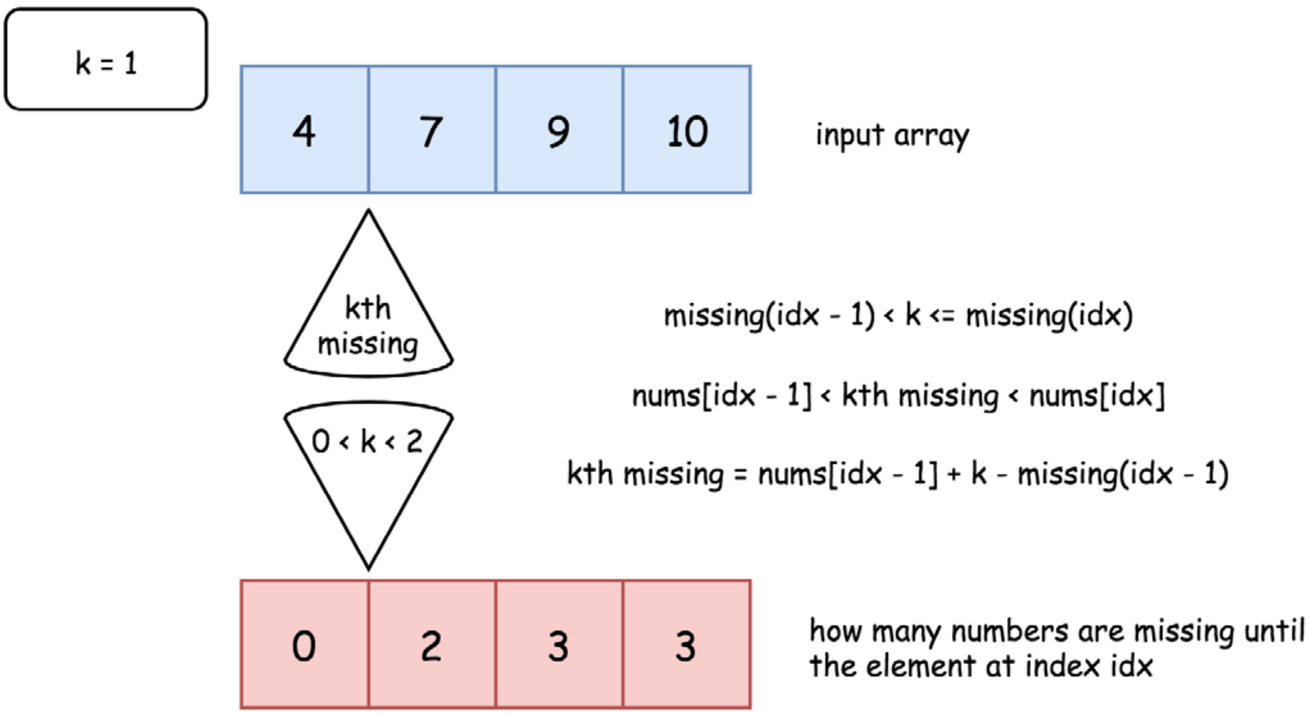


With the help of such a function the solution is straightforward :

- Find an index such that `missing(idx - 1) < k <= missing(idx)` . In other words, that means that kth missing number is in-between `nums[idx - 1]` and `nums[idx]` .

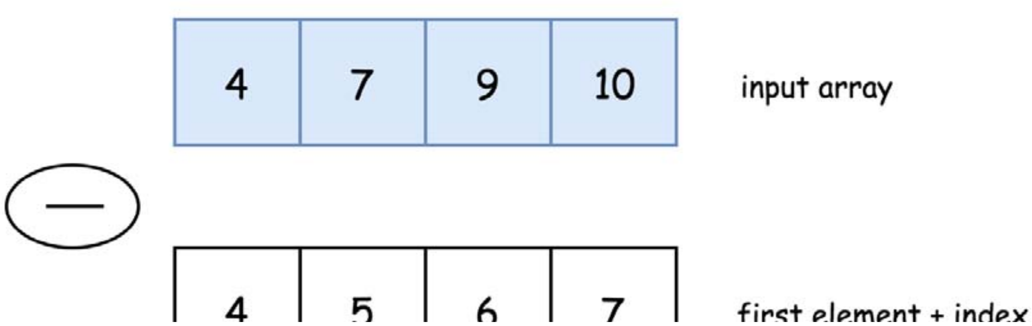
One even could compute a difference between kth missing number and `nums[idx - 1]`. First, there are `missing(idx - 1)` missing numbers until `nums[idx - 1]`. Second, all `k - missing(idx - 1)` missing numbers from `nums[idx - 1]` to kth missing are *consecutive ones*, because all of them are less than `nums[idx]` and hence there is nothing to separate them. Together that means that kth smallest is larger than `nums[idx - 1]` by `k - missing(idx - 1)`.

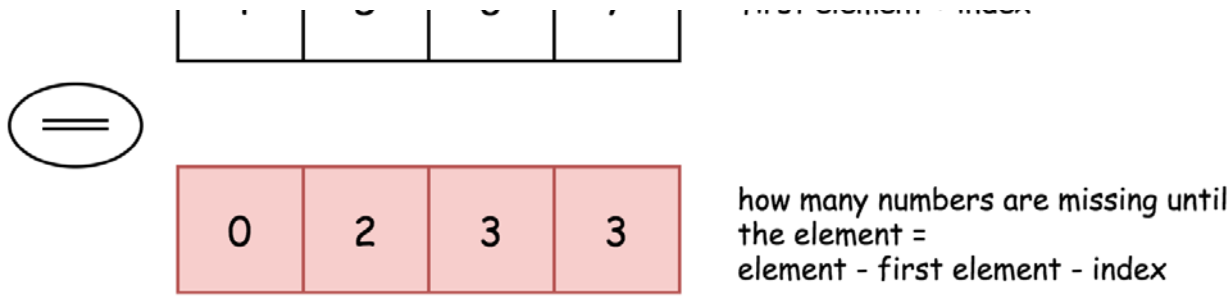
- Return kth smallest `nums[idx - 1] + k - missing(idx - 1)`.



> The last thing to discuss is how to implement `missing(idx)` function.

Let's consider an array element at index `idx`. If there is no numbers missing, the element should be equal to `nums[idx] = nums[0] + idx`. If k numbers are missing, the element should be equal to `nums[idx] = nums[0] + idx + k`. Hence the number of missing elements is equal to `nums[idx] - nums[0] - idx`.





Algorithm

- Implement `missing(idx)` function that returns how many numbers are missing until array element with index `idx`. Function returns `nums[idx] - nums[0] - idx`.
- Find an index such that `missing(idx - 1) < k <= missing(idx)` by a linear search.
- Return kth smallest `nums[idx - 1] + k - missing(idx - 1)`.

Implementation

Java Python Copy

```python
class Solution:
    def missingElement(self, nums: List[int], k: int) -> int:
        # Return how many numbers are missing until nums[idx]
        missing = lambda idx: nums[idx] - nums[0] - idx

        n = len(nums)
        # If kth missing number is larger than
        # the last element of the array
        if k > missing(n - 1):
            return nums[-1] + k - missing(n - 1)

        idx = 1
        # find idx such that
        # missing(idx - 1) < k <= missing(idx)
        while missing(idx) < k:
            idx += 1

        # kth missing number is greater than nums[idx - 1]
        # and less than nums[idx]
        return nums[idx - 1] + k - missing(idx - 1)
```

Complexity Analysis

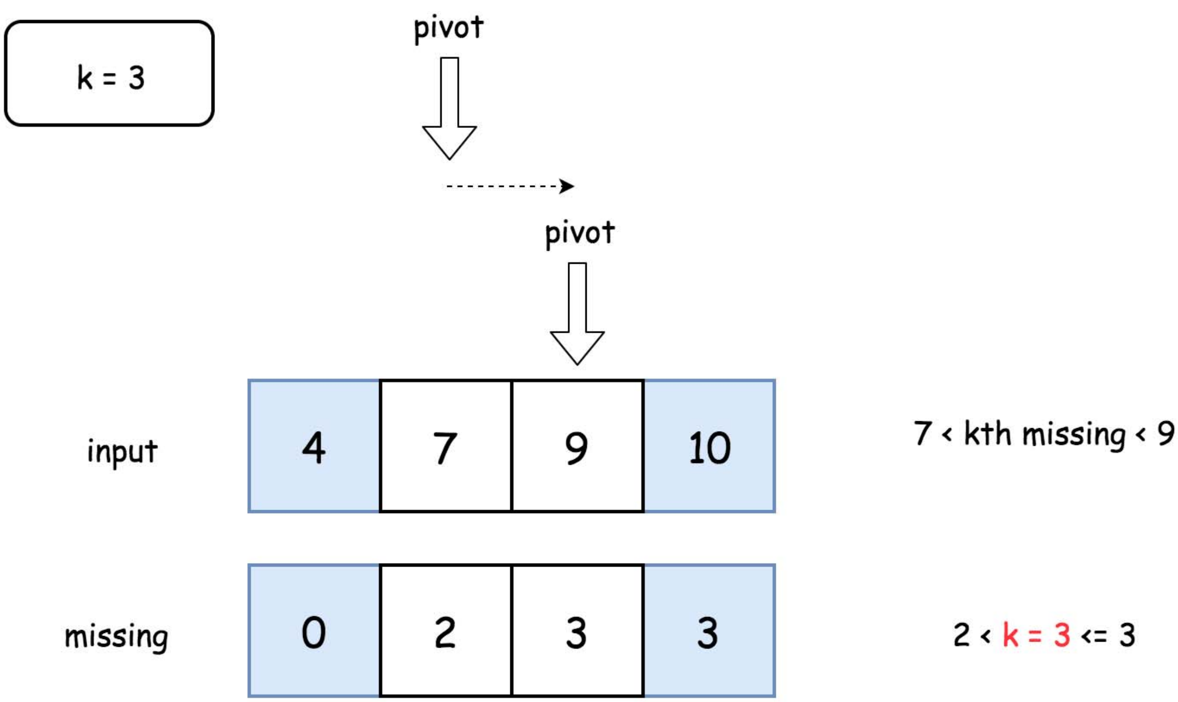
- Time complexity: $\mathcal{O}(N)$ since in the worst case it's one pass along the array.
- Space complexity: $\mathcal{O}(1)$ since it's a constant space solution.

Approach 2: Binary Search

Intuition

Approach 1 uses the linear search and doesn't profit from the fact that array is *sorted*. One could replace the linear search by a binary one (https://leetcode.com/articles/binary-search/) and reduce the time complexity from $\mathcal{O}(N)$ down to $\mathcal{O}(\log N)$.

> The idea is to find the leftmost element such that the number of missing numbers until this element is less or equal to k.



Algorithm

- Implement `missing(idx)` function that returns how many numbers are missing until array element with index `idx`. Function returns `nums[idx] - nums[0] - idx`.
- Find an index such that `missing(idx - 1) < k <= missing(idx)` by a *binary search*.
- Return kth smallest `nums[idx - 1] + k - missing(idx - 1)`.

Implementation

Java Python Copy

```
class Solution:
    def missingElement(self, nums: List[int], k: int) -> int:
        # Return how many numbers are missing until nums[idx]
        missing = lambda idx: nums[idx] - nums[0] - idx

        n = len(nums)
        # If kth missing number is larger than
        # the last element of the array
        if k > missing(n - 1):
            return nums[-1] + k - missing(n - 1)

        left, right = 0, n - 1
        # find left = right index such that
        # missing(left - 1) < k <= missing(left)
        while left != right:
            pivot = left + (right - left) // 2

            if missing(pivot) < k:
                left = pivot + 1
            else:
                right = pivot

        # kth missing number is greater than nums[left - 1]
        # and less than nums[left]
        return nums[left - 1] + k - missing(left - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(\log N)$ since it's a binary search algorithm in the worst case when the missing number is less than the last element of the array.
- Space complexity : $\mathcal{O}(1)$ since it's a constant space solution.

Rate this article:

Previous (/articles/binary-search-tree-iterator/) Next (/articles/ipo/)

Comments: 10 Sort By

Type comment here... (Markdown is supported)

Preview Post

bhushan55 (/bhushan55) ★ 93 July 28, 2019 9:36 PM

the time complexity is not constant, you should only put for the worst case here.

(/bhushan55)

31 Share Reply

SHOW 2 REPLIES

ywen1995 (/ywen1995) ★ 318 August 11, 2019 8:25 PM

The constant complexity is very misleading (or over simplified) here.

(/ywen1995)

11 Share Reply

ilkercankaya (/ilkercankaya) ★ 89 December 20, 2019 9:21 AM

Time Complexity is calculated by taking the worst-case into account. Saying that it is O(1) is extremely misleading for a lot of people. Please change it!

(/ilkercankaya)

6 Share Reply

nlackx (/nlackx) ★ 71 November 15, 2019 11:55 PM

(/nlackx)

> The idea is to find the leftmost element such that the number of missing numbers until this element is **smaller** or equal to k.

should be **greater**

3 Share Reply

montabano1 (/montabano1) ★ 2 April 21, 2020 10:52 AM

I think using the phrase that approach 1 "doesn't profit from the fact that array is sorted." is wrong. If the array wasnt sorted we would not be able to find what numbers were missing?

(/montabano1)

1 Share Reply

civabhusal (/civabhusal) ★ 2 April 3, 2020 9:28 AM

How does the second approach take O(logN) time when missing(idx) takes O(N) time ? Could you please kindly clarify ?

(/civabhusal)

1 Share Reply

SHOW 1 REPLY

Prashanth_123 (/prashanth_123) ★ 0 April 21, 2020 5:27 PM

(/prashanth_123)

```
public int missingElement(int[] nums, int k) {
       int temp = k, index = Integer.MIN_VALUE, numMissingElements
= 0;
```

Read More

0 Share Reply

(/pbu)

pbu (/pbu) ★ 261 March 4, 2020 5:57 AM

java, one pass:

```
class Solution {
    public int missingElement(int[] nums, int k) {
        int n = nums.length;
```

Read More

0 Share Reply

(/letsgo99)

letsGo99 (/letsgo99) ★ 4 March 16, 2020 11:38 PM

The time complexity is O(n). You use the lambda function which would iterate n times through the list before the binary search

-2 Share Reply

SHOW 2 REPLIES

(/fishfly)

FishFly (/fishfly) ★ -2 March 13, 2020 12:54 AM

The time complexity of lambda func is already O(N)...

-2 Share Reply

 Help Center (/support/) | Terms (/terms/) | Privacy (/privacy/) United States (/region/)