Average Rating: 3.45 (53 votes)

Premium

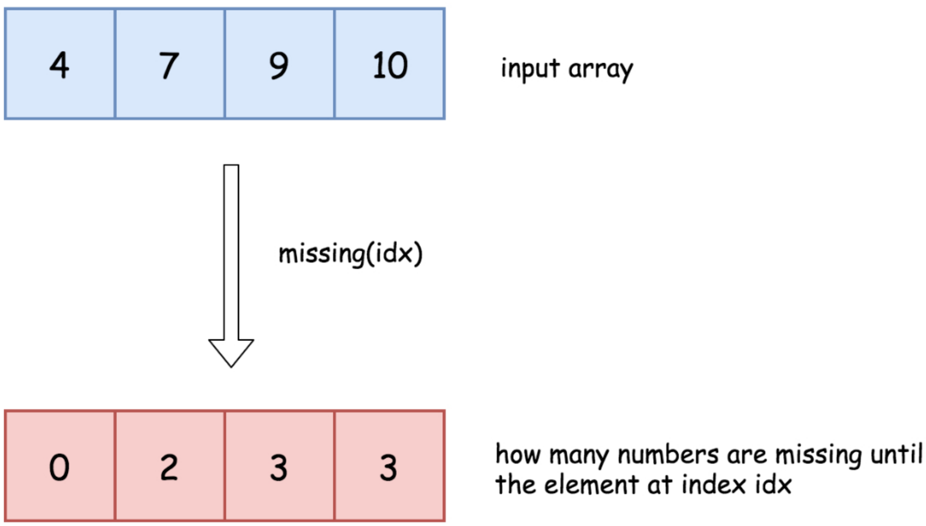
Solution

Approach 1: One Pass

Intuition

The problem is similar to 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`.

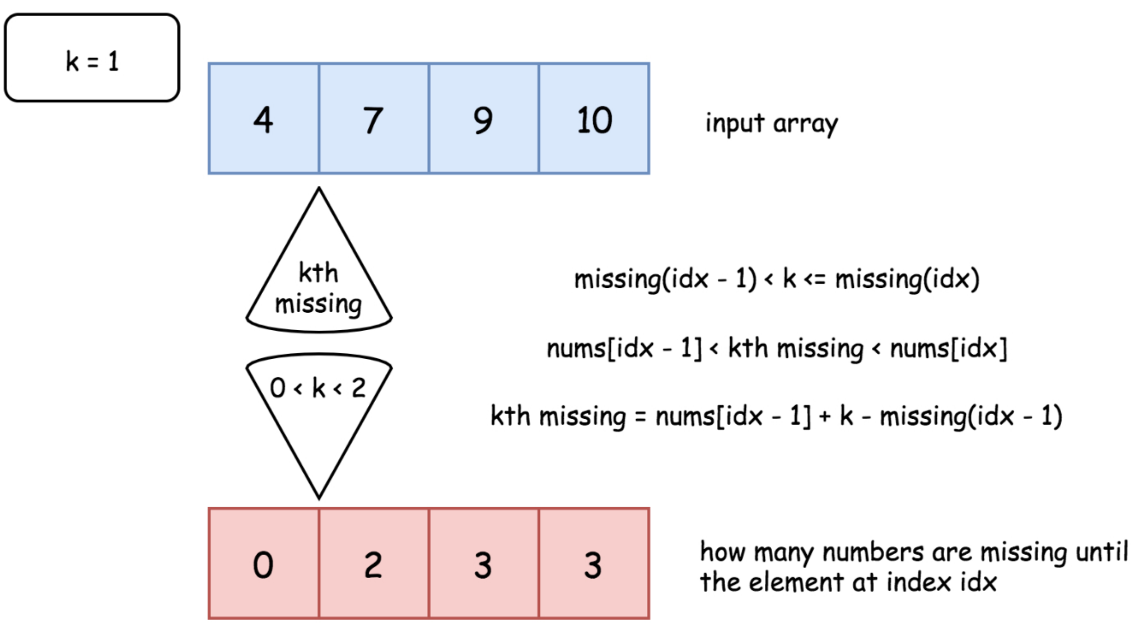


With the help of such a function the solution is straightforward :

- 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]`.

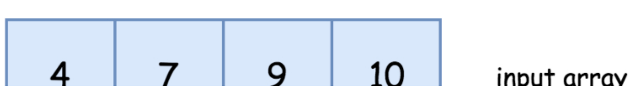
  One even could compute a difference between kth missing number and `nums[idx - 1]`. First, there are `missing(idx - 1)` missing numbers until `nums[idx - 1]`. Second, all `k - missing(idx - 1)` missing numbers from `nums[idx - 1]` to kth missing are *consecutive ones*, because all of them are less than `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`.



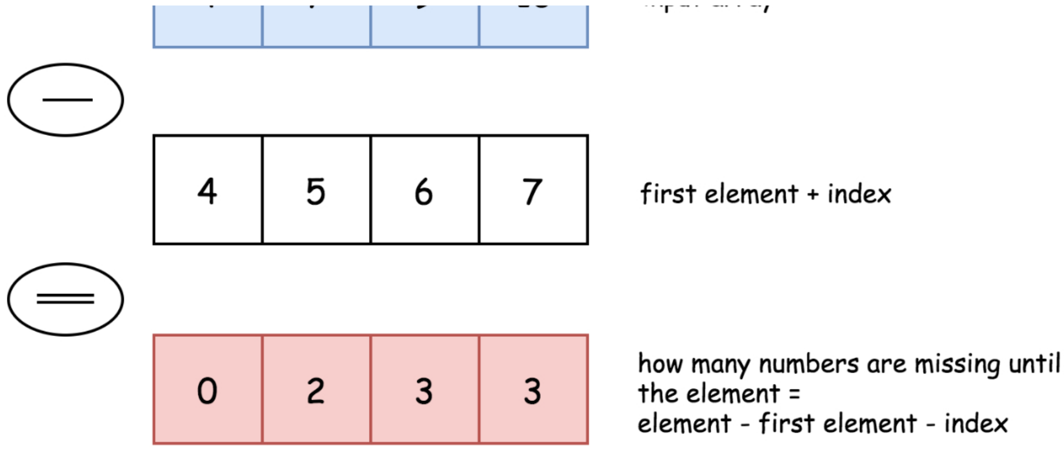
C++

Autocomplete

```cpp
class Solution {
public:
    // Return how many numbers are missing until nums[idx]
    int missing(int idx, vector<int> &nums) {
        return nums[idx] - nums[0] - idx;
    }

    int kth(int idx, vector<int> &nums, int k) {
        return nums[idx] + k - missing(idx, nums);
    }

    int missingElement(vector<int> &nums, int k) {
        int n = nums.size();
        // kth missing number after last number?
        if (k > missing(n - 1, nums))
            return kth(n-1, nums, k);

        int start = 0, end = n - 1, lower_bound = -1;
        // find start = end index such that
        // missing(start - 1) < k <= missing(start)
        while (start <= end) {
            int mid = start + (end - start) / 2;

            if (missing(mid, nums) < k)
                start = mid + 1;
            else
                end = mid-1, lower_bound = mid;
        }

        // kth missing number is
```

Your previous code was restored from your local storage

Console Contribute



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

```java
class Solution {
  // Return how many numbers are missing until nums[idx]
  int missing(int idx, int[] nums) {
    return nums[idx] - nums[0] - idx;
  }

  public int missingElement(int[] nums, int k) {
    int n = nums.length;
    // If kth missing number is larger than
    // the last element of the array
    if (k > missing(n - 1, nums))
      return nums[n - 1] + k - missing(n - 1, nums);

    int idx = 1;
    // find idx such that
    // missing(idx - 1) < k <= missing(idx)
    while (missing(idx, nums) < k) idx++;

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

Complexity Analysis

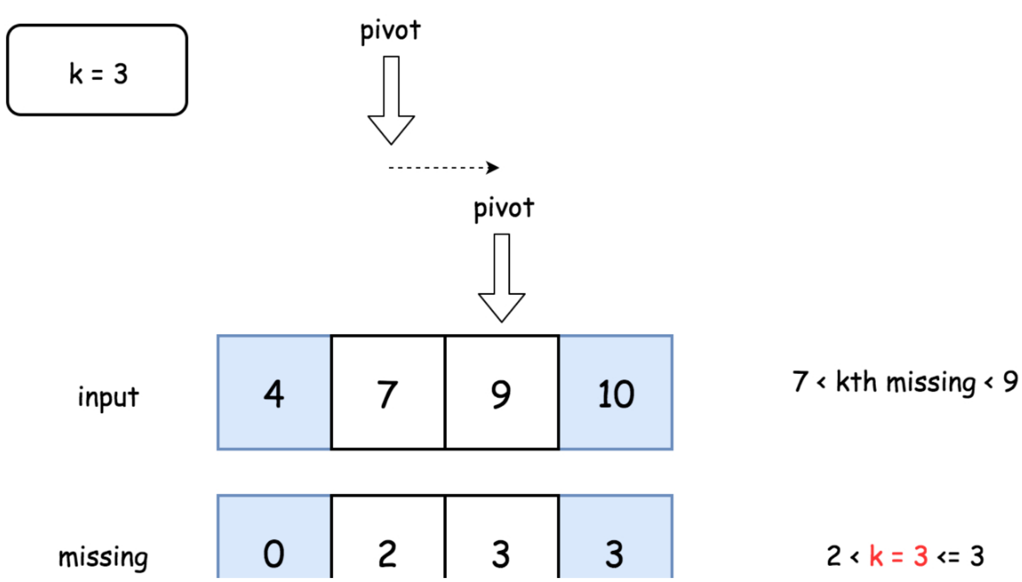
- Time complexity: $\mathcal{O}(N)$ since in the worst case it's one pass along the array.
- Space complexity: $\mathcal{O}(1)$ since it's a constant space solution.

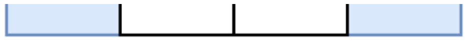
Approach 2: Binary Search

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 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

```java
class Solution {
  // Return how many numbers are missing until nums[idx]
  int missing(int idx, int[] nums) {
    return nums[idx] - nums[0] - idx;
  }

  public int missingElement(int[] nums, int k) {
    int n = nums.length;
    // If kth missing number is larger than
    // the last element of the array
    if (k > missing(n - 1, nums))
      return nums[n - 1] + k - missing(n - 1, nums);

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

      if (missing(pivot, nums) < k) left = pivot + 1;
      else right = pivot;
    }

    // kth missing number is greater than nums[idx - 1]
    // and less than nums[idx]
    return nums[left - 1] + k - missing(left - 1, nums);
  }
}
```

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.

Report Article Issue

Comments: 32

Best | Most Votes | Newest to Oldest | Oldest to Newest

Type comment here... (Markdown is supported)

Post

ltbtb_rise ★ 1039 Last Edit: June 29, 2020 5:34 AM

approach one solution was made like a rocket science while a simple for loop could just do the job:

```cpp
class Solution {
public:
    int missingElement(vector<int>& nums, int k) {
        int n=nums.size(),diff=0;
        for(int i=1;i<n;i++)
        {
            diff=nums[i]-nums[i-1]-1;
            if(diff>=k) return nums[i-1]+k;
            k-=diff;
        }
        return nums[n-1]+k;
    }
};
```

153 Show 4 replies Reply

montabano1 ★ 46 April 21, 2020 7:52 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?

44 Show 1 reply Reply

bhushan55 ★ 231 July 28, 2019 6:36 PM

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

35 Show 3 replies Reply

sutirtho ★ 315 July 5, 2020 9:39 AM

This question should be marked hard.

49 Show 3 replies Reply

ywen1995 ★ 758 August 11, 2019 5:25 PM

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

12 Reply

nlackx ★ 137 November 15, 2019 8:55 PM

> 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**

11 Reply

nyc_coder_84 ★ 35 March 1, 2021 2:45 PM

Was asked this question in my Facebook onsite interview, 2nd question after I completed the first one in under 20 min, took a long time to arrive at the binary search solution with the help of the interviewer, but didn't have time in the end to code it all. If you haven't seen this question before, it is very difficult to see that binary search pattern especially in an interview setting.

9 Show 6 replies Reply

steffi_keran 198 July 14, 2020 10:46 AM

What is the intuition behind `nums[idx - 1] + k - missing(idx - 1)` ?

6 Show 3 replies Reply

__thalaiva__ 915 September 7, 2020 8:53 PM

Analysis is fine. But, it needs more explanation than just - "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 :" There should be more emphasis on how to arrive to the above step from the problem statement. It is neither easy nor straightforward. Before calling out something as straightforward, please evaluate if the insight can be derived on the spot in interview without having seen similar problems before.

3 Reply

ilkercankaya 166 Last Edit: December 20, 2019 6:21 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!

6 Reply