<https://leetcode.com/problems/kth-missing-positive-number/description/>

Given an array arr of positive integers sorted in a **strictly increasing order**, and an integer k.

Return *the* kth ***positive*** *integer that is* ***missing*** *from this array.*

**Example 1:**

Input: arr = [2,3,4,7,11], k = 5

Output: 9

Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9.

**Example 2:**

Input: arr = [1,2,3,4], k = 2

Output: 6

Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive integer is 6.

**Constraints:**

* 1 <= arr.length <= 1000
* 1 <= arr[i] <= 1000
* 1 <= k <= 1000
* arr[i] < arr[j] for 1 <= i < j <= arr.length

**Follow up:**

Could you solve this problem in less than O(n) complexity?

**Attempt 1: 2023-09-18**

**Solution 1:  Hash Table (10 min)**

class Solution {

public int findKthPositive(int[] arr, int k) {

Set<Integer> set = new HashSet<>();

for(int num : arr) {

set.add(num);

}

int i = 1;

int result = 0;

while(k > 0) {

if(!set.contains(i)) {

k--;

result = i;

}

i++;

}

return result;

}

}

===============================================================

No need extra variable 'result'

class Solution {

public int findKthPositive(int[] arr, int k) {

Set<Integer> set = new HashSet<>();

for(int num : arr) {

set.add(num);

}

int i = 1;

// When k == 0, i additionally increase one more time,

// if removing 'result' we have to decrease that

// additional one more add

//int result = 0;

while(k > 0) {

if(!set.contains(i)) {

k--;

//result = i;

}

i++;

}

//return result;

return i - 1;

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Solution 2:  nums[i] and (i + 1) relation (30 min, difficult to think about)**

class Solution {

public int findKthPositive(int[] arr, int k) {

int n = arr.length;

// 因为题目给的是一个升序的正整数数组，而且数组理论上

// 是从1开始。所以理想情况下，数字nums[i]和它对应的

// 下标i的关系应该是nums[i] = i + 1。所以在扫描input

// 数组的时候，如果发觉数字和他对应的下标的差大于1了，

// 则说明中间开始缺失数字了。当这个差距大于等于K的时候，

// 则找到了第K个缺失的数字

for(int i = 0; i < n; i++) {

if(arr[i] - i - 1 >= k) {

return i + k;

}

}

// e.g arr = [1,2,3,4], k = 2, expect = 6

// e.g arr = [1,2,4,5], k = 2, expect = 6

return n + k;

}

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://www.cnblogs.com/cnoodle/p/13632810.html>

**一个比较优化的线性解法是，因为题目给的是一个升序的正整数数组，而且数组理论上是从1开始。所以理想情况下，数字nums[i]和它对应的下标i的关系应该是nums[i] = i + 1。所以在扫描input数组的时候，如果发觉数字和他对应的下标的差大于1了，则说明中间开始缺失数字了。当这个差距大于等于K的时候，则找到了第K个缺失的数字。**

时间O(n)

空间O(1)

class Solution {

public int findKthPositive(int[] arr, int k) {

int len = arr.length;

for (int i = 0; i < len; i++) {

if (arr[i] - i - 1 >= k) {

return k + i;

}

}

return k + len;

}

}

**Solution 3:  Binary Search (30 min, difficult on getting this idea)**

**使用L704.Binary Search的找下界模版 (Find the first number able to produce k as difference between its value and its index)**

class Solution {

public int findKthPositive(int[] arr, int k) {

// Find the first number able to produce k as

// difference between its value and its index

int lo = 0;

int hi = arr.length - 1;

while(lo <= hi) {

int mid = lo + (hi - lo) / 2;

int diff = arr[mid] - (mid + 1);

if(diff >= k) {

hi = mid - 1;

} else {

lo = mid + 1;

}

}

return lo + k;

}

}

Time Complexity: O(logN)

Space Complexity: O(1)

**Refer to**

<https://www.cnblogs.com/cnoodle/p/13632810.html>

二分法。**因为input数组是有序的，所以如果面试官不满意O(n)级别的思路的话，可以试图往二分法上靠。**mid找到中点，还是跟第二种做法类似，比较当前这个index上的数字和index的差值。如果差值小于K，则往数组的右半边找；反之则往数组的左半边找。

时间O(logn)

空间O(1)

class Solution {

public int findKthPositive(int[] arr, int k) {

int len = arr.length;

for (int i = 0; i < len; i++) {

if (arr[i] - i - 1 >= k) {

return k + i;

}

}

return k + len;

}

}

**Refer to**

<https://zxi.mytechroad.com/blog/algorithms/binary-search/leetcode-1539-kth-missing-positive-number/>

We can find the smallest index l using binary search, s.t. arr[l] – l + 1 >= k, which means we missed at least k numbers at index l. And the answer will be l + k.

Time complexity: O(logn)

Space complexity: O(1)

class Solution {

public int findKthPositive(int[] arr, int k) {

int l = 0;

int r = arr.length;

while (l < r) {

int m = l + (r - l) / 2;

if (arr[m] - (m + 1) >= k)

r = m;

else

l = m + 1;

}

return l + k;

}

}