Problem 1060: Missing Element in Sorted Array

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an integer array

nums

which is sorted in

ascending order

and all of its elements are

unique

and given also an integer

k

, return the

k

th

missing number starting from the leftmost number of the array.

Example 1:

Input:

nums = [4,7,9,10], k = 1

Output:

5

Explanation:

The first missing number is 5.

Example 2:

Input:

nums = [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:

nums = [1,2,4], k = 3

Output:

6

Explanation:

The missing numbers are [3,5,6,7,...], hence the third missing number is 6.

Constraints:

1 <= nums.length <= 5 * 10

4

1 <= nums[i] <= 10

7

nums

is sorted in

ascending order,

and all the elements are

unique

.

1 <= k <= 10

8

Follow up:

Can you find a logarithmic time complexity (i.e.,

O(log(n))

) solution?

Code Snippets

C++:

```
class Solution {
public:
int missingElement(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int missingElement(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def missingElement(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def missingElement(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var missingElement = function(nums, k) {

};
```

TypeScript:

```
function missingElement(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MissingElement(int[] nums, int k) {

}
}
```

C:

```
int missingElement(int* nums, int numsSize, int k) {

}
```

Go:

```
func missingElement(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun missingElement(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func missingElement(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn missing_element(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def missing_element(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function missingElement($nums, $k) {

}
}
```

Dart:

```
class Solution {
int missingElement(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def missingElement(nums: Array[Int], k: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec missing_element(nums :: [integer], k :: integer) :: integer
def missing_element(nums, k) do

end
end
```

Erlang:

```
-spec missing_element(Nums :: [integer()], K :: integer()) -> integer().
missing_element(Nums, K) ->
.
```

Racket:

```
(define/contract (missing-element nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int missingElement(vector<int>& nums, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int missingElement(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Missing Element in Sorted Array
Difficulty: Medium
Tags: array, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def missingElement(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def missingElement(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var missingElement = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function missingElement(nums: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MissingElement(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Missing Element in Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int missingElement(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Missing Element in Sorted Array
// Difficulty: Medium
```

```
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func missingElement(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun missingElement(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func missingElement(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Missing Element in Sorted Array
// Difficulty: Medium
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn missing_element(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def missing_element(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function missingElement($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int missingElement(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def missingElement(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec missing_element(nums :: [integer], k :: integer) :: integer
def missing_element(nums, k) do

end
end
```

Erlang Solution:

```
-spec missing_element(Nums :: [integer()], K :: integer()) -> integer().
missing_element(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (missing-element nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```