Problem 1064: Fixed Point

Problem Information

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

Problem Description

Given an array of distinct integers

arr

, where

arr

is sorted in

ascending order

, return the smallest index

i

that satisfies

arr[i] == i

. If there is no such index, return

-1

.

Example 1:

Input:

arr = [-10,-5,0,3,7]

Output:

3

Explanation:

For the given array,

arr[0] = -10, arr[1] = -5, arr[2] = 0, arr[3] = 3

, thus the output is 3.

Example 2:

Input:

arr = [0,2,5,8,17]

Output:

0

Explanation:

arr[0] = 0

, thus the output is 0.

Example 3:

Input:

arr = [-10,-5,3,4,7,9]

Output:

-1

Explanation:

There is no such

i

that

arr[i] == i

, thus the output is -1.

Constraints:

1 <= arr.length < 10

4

-10

9

<= arr[i] <= 10

9

Follow up:

The

O(n)

solution is very straightforward. Can we do better?

Code Snippets

C++:

```
class Solution {
public:
int fixedPoint(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int fixedPoint(int[] arr) {

}
}
```

Python3:

```
class Solution:
def fixedPoint(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def fixedPoint(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @return {number}
*/
var fixedPoint = function(arr) {

};
```

TypeScript:

```
function fixedPoint(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int FixedPoint(int[] arr) {

}
}
```

C:

```
int fixedPoint(int* arr, int arrSize) {

}
```

Go:

```
func fixedPoint(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun fixedPoint(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func fixedPoint(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn fixed_point(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def fixed_point(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function fixedPoint($arr) {

}
}
```

Dart:

```
class Solution {
int fixedPoint(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def fixedPoint(arr: Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec fixed_point(arr :: [integer]) :: integer
def fixed_point(arr) do

end
end
```

Erlang:

```
-spec fixed_point(Arr :: [integer()]) -> integer().
fixed_point(Arr) ->
.
```

Racket:

```
(define/contract (fixed-point arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Fixed Point
* Difficulty: Easy
* 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 fixedPoint(vector<int>& arr) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Fixed Point
* Difficulty: Easy
* 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 fixedPoint(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Fixed Point
Difficulty: Easy
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 fixedPoint(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def fixedPoint(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Fixed Point
* Difficulty: Easy
* 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[]} arr
* @return {number}
*/
var fixedPoint = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Fixed Point
* Difficulty: Easy
* 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 fixedPoint(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Fixed Point
* Difficulty: Easy
* 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 FixedPoint(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Fixed Point
* Difficulty: Easy
* 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 fixedPoint(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Fixed Point
// Difficulty: Easy
// 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 fixedPoint(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun fixedPoint(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func fixedPoint(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Fixed Point
// Difficulty: Easy
// 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 fixed_point(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def fixed_point(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function fixedPoint($arr) {

}
}
```

Dart Solution:

```
class Solution {
int fixedPoint(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def fixedPoint(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec fixed_point(arr :: [integer]) :: integer
def fixed_point(arr) do

end
```

```
end
```

Erlang Solution:

```
-spec fixed_point(Arr :: [integer()]) -> integer().
fixed_point(Arr) ->
.
```

Racket Solution:

```
(define/contract (fixed-point arr)
(-> (listof exact-integer?) exact-integer?)
)
```