Problem 1966: Binary Searchable Numbers in an Unsorted Array

Problem Information

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

Problem Description

Consider a function that implements an algorithm

similar

to

Binary Search

. The function has two input parameters:

sequence

is a sequence of integers, and

target

is an integer value. The purpose of the function is to find if the

target

exists in the

sequence

.

The pseudocode of the function is as follows:

func(sequence, target) while sequence is not empty

randomly

choose an element from sequence as the pivot if pivot = target, return

true

else if pivot < target, remove pivot and all elements to its left from the sequence else, remove pivot and all elements to its right from the sequence end while return

false

When the

sequence

is sorted, the function works correctly for

all

values. When the

sequence

is not sorted, the function does not work for all values, but may still work for

some

values.

Given an integer array

nums

, representing the

sequence

, that contains

unique

numbers and

may or may not be sorted

, return

the number of values that are

guaranteed

to be found using the function, for

every possible

pivot selection

.

Example 1:

Input:

nums = [7]

Output:

1

Explanation

: Searching for value 7 is guaranteed to be found. Since the sequence has only one element, 7 will be chosen as the pivot. Because the pivot equals the target, the function will return true.

Example 2:

Input:

nums = [-1,5,2]

Output:

1

Explanation

: Searching for value -1 is guaranteed to be found. If -1 was chosen as the pivot, the function would return true. If 5 was chosen as the pivot, 5 and 2 would be removed. In the next loop, the sequence would have only -1 and the function would return true. If 2 was chosen as the pivot, 2 would be removed. In the next loop, the sequence would have -1 and 5. No matter which number was chosen as the next pivot, the function would find -1 and return true.

Searching for value 5 is NOT guaranteed to be found. If 2 was chosen as the pivot, -1, 5 and 2 would be removed. The sequence would be empty and the function would return false.

Searching for value 2 is NOT guaranteed to be found. If 5 was chosen as the pivot, 5 and 2 would be removed. In the next loop, the sequence would have only -1 and the function would return false.

Because only -1 is guaranteed to be found, you should return 1.

Constraints:

1 <= nums.length <= 10

5

-10

5

<= nums[i] <= 10

5

All the values of

nums

are

unique

.

Follow-up:

If

nums

has

duplicates

, would you modify your algorithm? If so, how?

Code Snippets

C++:

```
class Solution {
public:
int binarySearchableNumbers(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int binarySearchableNumbers(int[] nums) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function binarySearchableNumbers(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int BinarySearchableNumbers(int[] nums) {

}
}
```

C:

```
int binarySearchableNumbers(int* nums, int numsSize) {

}
```

Go:

```
func binarySearchableNumbers(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun binarySearchableNumbers(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func binarySearchableNumbers(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn binary_searchable_numbers(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def binary_searchable_numbers(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function binarySearchableNumbers($nums) {

}
}
```

Dart:

```
class Solution {
int binarySearchableNumbers(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def binarySearchableNumbers(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec binary_searchable_numbers(nums :: [integer]) :: integer
def binary_searchable_numbers(nums) do

end
end
```

Erlang:

```
-spec binary_searchable_numbers(Nums :: [integer()]) -> integer().
binary_searchable_numbers(Nums) ->
.
```

Racket:

```
(define/contract (binary-searchable-numbers nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Binary Searchable Numbers in an Unsorted 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
* @return {number}
*/
var binarySearchableNumbers = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Binary Searchable Numbers in an Unsorted 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 BinarySearchableNumbers(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Binary Searchable Numbers in an Unsorted 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 binarySearchableNumbers(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun binarySearchableNumbers(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func binarySearchableNumbers(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Binary Searchable Numbers in an Unsorted 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 binary_searchable_numbers(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def binary_searchable_numbers(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function binarySearchableNumbers($nums) {

}
}
```

Dart Solution:

```
class Solution {
int binarySearchableNumbers(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def binarySearchableNumbers(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec binary_searchable_numbers(nums :: [integer]) :: integer
def binary_searchable_numbers(nums) do

end
end
```

Erlang Solution:

```
-spec binary_searchable_numbers(Nums :: [integer()]) -> integer().
binary_searchable_numbers(Nums) ->
.
```

Racket Solution:

```
(define/contract (binary-searchable-numbers nums)
(-> (listof exact-integer?) exact-integer?)
)
```