Problem 33: Search in Rotated Sorted Array

Problem Information

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

Problem Description

There is an integer array

nums

sorted in ascending order (with

distinct

values).

Prior to being passed to your function,

nums

is

possibly left rotated

at an unknown index

k

(

1 <= k < nums.length

) such that the resulting array is

[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]

(

0-indexed

). For example,

[0,1,2,4,5,6,7]

might be left rotated by

3

indices and become

[4,5,6,7,0,1,2]

.

Given the array

nums

after

the possible rotation and an integer

target

, return

the index of

target

if it is in

nums

, or

-1

if it is not in

nums

.

You must write an algorithm with

O(log n)

runtime complexity.

Example 1:

Input:

nums = [4,5,6,7,0,1,2], target = 0

Output:

4

Example 2:

Input:

nums = [4,5,6,7,0,1,2], target = 3

Output:

-1

Example 3:

Input:

nums = [1], target = 0

Output:

-1

Constraints:

1 <= nums.length <= 5000

-10

4

<= nums[i] <= 10

4

All values of

nums

are

unique

.

nums

is an ascending array that is possibly rotated.

-10

4

<= target <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int search(vector<int>& nums, int target) {

}
};
```

Java:

```
class Solution {
public int search(int[] nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function search(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int Search(int[] nums, int target) {

}
}
```

C:

```
int search(int* nums, int numsSize, int target) {

}
```

Go:

```
func search(nums []int, target int) int {

}
```

Kotlin:

```
class Solution {
fun search(nums: IntArray, target: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def search(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function search($nums, $target) {

}
}
```

Dart:

```
class Solution {
int search(List<int> nums, int target) {
```

```
}
}
```

Scala:

```
object Solution {
def search(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec search(nums :: [integer], target :: integer) :: integer
def search(nums, target) do

end
end
```

Erlang:

```
-spec search(Nums :: [integer()], Target :: integer()) -> integer().
search(Nums, Target) ->
.
```

Racket:

```
(define/contract (search nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Search in Rotated 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 search(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Search in Rotated 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 search(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Search in Rotated 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 search(self, nums: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Search in Rotated 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} target
* @return {number}
*/
var search = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Search in Rotated 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 search(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Search in Rotated 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 Search(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Search in Rotated 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 search(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Search in Rotated 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 search(nums []int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun search(nums: IntArray, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Search in Rotated 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 search(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def search(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function search($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int search(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def search(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec search(nums :: [integer], target :: integer) :: integer
def search(nums, target) do

end
end
```

Erlang Solution:

```
-spec search(Nums :: [integer()], Target :: integer()) -> integer().
search(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (search nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```