Problem 81: Search in Rotated Sorted Array II

Problem Information

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

Problem Description

There is an integer array

nums

sorted in non-decreasing order (not necessarily with

distinct

values).

Before being passed to your function,

nums

is

rotated

at an unknown pivot index

k

(

0 <= 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,

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

might be rotated at pivot index

5

and become

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

.

Given the array

nums

after

the rotation and an integer

target

, return

true

if

target

is in

nums

, or

false

if it is not in

nums

.

You must decrease the overall operation steps as much as possible.

Example 1:

Input:

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

Output:

true

Example 2:

Input:

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

Output:

false

Constraints:

1 <= nums.length <= 5000

-10

4

<= nums[i] <= 10

4

nums

is guaranteed to be rotated at some pivot.

-10

4

<= target <= 10

4

Follow up:

This problem is similar to

Search in Rotated Sorted Array

, but

nums

may contain

duplicates

. Would this affect the runtime complexity? How and why?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

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

}
};
```

Java Solution:

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

}
}
```

Python3 Solution:

```
"""
Problem: Search in Rotated Sorted Array II
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) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Search in Rotated Sorted Array II
* 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 {boolean}
*/
var search = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Search in Rotated Sorted Array II
* 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): boolean {

};
```

C# Solution:

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

}
}
```

C Solution:

```
/*
* Problem: Search in Rotated Sorted Array II
* 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
*/

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

}
```

Go Solution:

```
// Problem: Search in Rotated Sorted Array II
// 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) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Search in Rotated Sorted Array II
// 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) -> bool {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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