Problem 704: Binary Search

Problem Information

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

Problem Description

Given an array of integers

nums

which is sorted in ascending order, and an integer

target

, write a function to search

target

in

nums

. If

target

exists, then return its index. Otherwise, return

-1

.

You must write an algorithm with

O(log n)

runtime complexity.

Example 1:

Input:

nums = [-1,0,3,5,9,12], target = 9

Output:

4

Explanation:

9 exists in nums and its index is 4

Example 2:

Input:

nums = [-1,0,3,5,9,12], target = 2

Output:

-1

Explanation:

2 does not exist in nums so return -1

Constraints:

1 <= nums.length <= 10

4

-10

4

< nums[i], target < 10

4

All the integers in

nums

are

unique

.

nums

is sorted in ascending order.

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:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```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: Binary Search
* 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 search(vector<int>& nums, int target) {

}
};
```

Java Solution:

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

}
}
```

Python3 Solution:

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

};
```

TypeScript Solution:

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

};
```

C# Solution:

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

}
}
```

C Solution:

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

}
```

Go Solution:

```
// Problem: Binary Search
// 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 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: Binary Search
// 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 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?)
)
```