

Problem 34: Find First and Last Position of Element in Sorted Array

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an array of integers

nums

sorted in non-decreasing order, find the starting and ending position of a given

target

value.

If

target

is not found in the array, return

`[-1, -1]`

.

You must write an algorithm with

$O(\log n)$

runtime complexity.

Example 1:

Input:

nums = [5,7,7,8,8,10], target = 8

Output:

[3,4]

Example 2:

Input:

nums = [5,7,7,8,8,10], target = 6

Output:

[-1,-1]

Example 3:

Input:

nums = [], target = 0

Output:

[-1,-1]

Constraints:

$0 \leq \text{nums.length} \leq 10$

5

-10

9

`<= nums[i] <= 10`

9

nums

is a non-decreasing array.

-10

9

`<= target <= 10`

9

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

```

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

```

Python:

```

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

```

JavaScript:

```

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

};

```

TypeScript:

```

function searchRange(nums: number[], target: number): number[] {

};

```

C#:

```

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

    }
}

```

C:

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */

```

```
int* searchRange(int* nums, int numsSize, int target, int* returnSize) {  
  
}
```

Go:

```
func searchRange(nums []int, target int) []int {  
  
}
```

Kotlin:

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

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer $target  
     * @return Integer[]  
     */  
    function searchRange($nums, $target) {  
  
    }  
}
```

Dart:

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

Scala:

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

Elixir:

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

Erlang:

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

.

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Find First and Last Position of Element in 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:
    vector<int> searchRange(vector<int>& nums, int target) {

    }
};
```

Java Solution:

```
/**
 * Problem: Find First and Last Position of Element in 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[] searchRange(int[] nums, int target) {

}

}

```

Python3 Solution:

```

"""
Problem: Find First and Last Position of Element in 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 searchRange(self, nums: List[int], target: int) -> List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Find First and Last Position of Element in 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 searchRange = function(nums, target) {

};

```

TypeScript Solution:

```

/**
 * Problem: Find First and Last Position of Element in 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 searchRange(nums: number[], target: number): number[] {

};

```

C# Solution:

```

/*
 * Problem: Find First and Last Position of Element in 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[] SearchRange(int[] nums, int target) {

    }
}

```

C Solution:

```

/*
 * Problem: Find First and Last Position of Element in 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
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* searchRange(int* nums, int numsSize, int target, int* returnSize) {

}

```

Go Solution:

```

// Problem: Find First and Last Position of Element in 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 searchRange(nums []int, target int) []int {

}

```

Kotlin Solution:

```

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

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Find First and Last Position of Element in 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_range(nums: Vec<i32>, target: i32) -> Vec<i32> {

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

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

  }

}

```

Dart Solution:

```

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

  }

}

```

Scala Solution:

```

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

  }

}

```

Elixir Solution:

```

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

  end

end

```

Erlang Solution:

```

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

.

```

Racket Solution:

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