Problem 2089: Find Target Indices After Sorting Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and a target element

target

.

A

target index

is an index

i

such that

nums[i] == target

.

Return

a list of the target indices of

nums

after

sorting

nums

in

non-decreasing

order

. If there are no target indices, return

an

empty

list

. The returned list must be sorted in

increasing

order.

Example 1:

Input:

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

Output:

[1,2]

Explanation:

After sorting, nums is [1,

2

,

2

,3,5]. The indices where nums[i] == 2 are 1 and 2.

Example 2:

Input:

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

Output:

[3]

Explanation:

After sorting, nums is [1,2,2,

3

,5]. The index where nums[i] == 3 is 3.

Example 3:

Input:

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

Output:

[4]

Explanation:

After sorting, nums is [1,2,2,3,

5

]. The index where nums[i] == 5 is 4.

Constraints:

1 <= nums.length <= 100

1 <= nums[i], target <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> targetIndices(int[] nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> TargetIndices(int[] nums, int target) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun targetIndices(nums: IntArray, target: Int): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find Target Indices After Sorting Array
* 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:
vector<int> targetIndices(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Find Target Indices After Sorting Array
* 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 List<Integer> targetIndices(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Target Indices After Sorting Array
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 targetIndices(self, nums: List[int], target: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```javascript
/**
* Problem: Find Target Indices After Sorting Array
* 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 targetIndices = function(nums, target) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Find Target Indices After Sorting Array
* 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 targetIndices(nums: number[], target: number): number[] {

};
```

C# Solution:

```csharp
/*
* Problem: Find Target Indices After Sorting Array
* 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 IList<int> TargetIndices(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Find Target Indices After Sorting Array
* 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
*/

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

}
```

Go Solution:

```
// Problem: Find Target Indices After Sorting Array
// 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 targetIndices(nums []int, target int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun targetIndices(nums: IntArray, target: Int): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Target Indices After Sorting Array
// 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 target_indices(nums: Vec<i32>, target: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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