Problem 2454: Next Greater Element IV

Problem Information

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

Problem Description

You are given a

0-indexed

array of non-negative integers

nums

. For each integer in

nums

, you must find its respective

second greater

integer.

The

second greater

integer of

nums[i]

is

nums[j]

such that:

j > i

nums[j] > nums[i]

There exists

exactly one

index

k

such that

nums[k] > nums[i]

and

i < k < j

.

If there is no such

nums[j]

, the second greater integer is considered to be

-1

.

For example, in the array

[1, 2, 4, 3]

, the second greater integer of

1

is

4

,

2

is

3

, and that of

3

and

4

is

-1

.

Return

an integer array

answer

, where

answer[i]

is the second greater integer of

nums[i]

.

Example 1:

Input:

nums = [2,4,0,9,6]

Output:

[9,6,6,-1,-1]

Explanation:

0th index: 4 is the first integer greater than 2, and 9 is the second integer greater than 2, to the right of 2. 1st index: 9 is the first, and 6 is the second integer greater than 4, to the right of 4. 2nd index: 9 is the first, and 6 is the second integer greater than 0, to the right of 0. 3rd index: There is no integer greater than 9 to its right, so the second greater integer is considered to be -1. 4th index: There is no integer greater than 6 to its right, so the second greater integer is considered to be -1. Thus, we return [9,6,6,-1,-1].

Example 2:

Input:

nums = [3,3]

Output:

[-1,-1]

Explanation:

We return [-1,-1] since neither integer has any integer greater than it.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> secondGreaterElement(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int[] secondGreaterElement(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function secondGreaterElement(nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] SecondGreaterElement(int[] nums) {

}
}
```

C:

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

}
```

Go:

```
func secondGreaterElement(nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun secondGreaterElement(nums: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func secondGreaterElement(_ nums: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn second_greater_element(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def second_greater_element(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function secondGreaterElement($nums) {

}
}
```

Dart:

```
class Solution {
List<int> secondGreaterElement(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def secondGreaterElement(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec second_greater_element(nums :: [integer]) :: [integer]
def second_greater_element(nums) do

end
end
```

Erlang:

```
-spec second_greater_element(Nums :: [integer()]) -> [integer()].
second_greater_element(Nums) ->
.
```

Racket:

```
(define/contract (second-greater-element nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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> secondGreaterElement(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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[] secondGreaterElement(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Greater Element IV
Difficulty: Hard
Tags: array, sort, search, stack, queue, heap

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 secondGreaterElement(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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
* @return {number[]}
*/
var secondGreaterElement = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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 secondGreaterElement(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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[] SecondGreaterElement(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Next Greater Element IV
* Difficulty: Hard
* Tags: array, sort, search, stack, queue, heap
*
* 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* secondGreaterElement(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Next Greater Element IV
// Difficulty: Hard
// Tags: array, sort, search, stack, queue, heap
//
// 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 secondGreaterElement(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun secondGreaterElement(nums: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func secondGreaterElement(_ nums: [Int]) -> [Int] {


}
}
```

Rust Solution:

```
// Problem: Next Greater Element IV
// Difficulty: Hard
// Tags: array, sort, search, stack, queue, heap
//
// 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 second_greater_element(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def second_greater_element(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function secondGreaterElement($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> secondGreaterElement(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def secondGreaterElement(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec second_greater_element(nums :: [integer]) :: [integer]
def second_greater_element(nums) do

end
end
```

Erlang Solution:

```
-spec second_greater_element(Nums :: [integer()]) -> [integer()].
second_greater_element(Nums) ->
.
```

Racket Solution:

```
(define/contract (second-greater-element nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```