Problem 503: Next Greater Element II

Problem Information

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

Problem Description

Given a circular integer array

nums

(i.e., the next element of

nums[nums.length - 1]

is

nums[0]

), return

the

next greater number

for every element in

nums

.

The

next greater number

of a number

x

is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return

-1

for this number.

Example 1:

Input:

nums = [1,2,1]

Output:

[2,-1,2] Explanation: The first 1's next greater number is 2; The number 2 can't find next greater number. The second 1's next greater number needs to search circularly, which is also 2.

Example 2:

Input:

nums = [1,2,3,4,3]

Output:

[2,3,4,-1,4]

Constraints:

1 <= nums.length <= 10

4

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (next-greater-elements nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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> nextGreaterElements(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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[] nextGreaterElements(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Greater Element II
Difficulty: Medium
Tags: array, search, stack
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 nextGreaterElements = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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 nextGreaterElements(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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[] NextGreaterElements(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Next Greater Element II
* Difficulty: Medium
* Tags: array, search, stack
*
* 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* nextGreaterElements(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Next Greater Element II
// Difficulty: Medium
// Tags: array, search, stack
//
// 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 nextGreaterElements(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Next Greater Element II
// Difficulty: Medium
// Tags: array, search, stack
//
// 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 next_greater_elements(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (next-greater-elements nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```