Problem 2113: Elements in Array After Removing and Replacing Elements

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. Initially on minute

0

, the array is unchanged. Every minute, the

leftmost

element in

nums

is removed until no elements remain. Then, every minute, one element is appended to the

end

of

nums

, in the order they were removed in, until the original array is restored. This process repeats indefinitely.

For example, the array

[0,1,2]

would change as follows:

[0,1,2] → [1,2] → [2] → [] → [0] → [0,1] → [0,1,2] → [1,2] → [2] → [] → [0] → [0,1] → [0,1,2] → ...

You are also given a 2D integer array

queries

of size

n

where

queries[j] = [time

j

, index

j

]

. The answer to the

j

th

query is:

nums[index

j

]

if

index

j

< nums.length

at minute

time

j

-1

if

index

j

>= nums.length

at minute

time

j

Return

an integer array

ans

of size

n

where

ans[j]

is the answer to the

j

th

query

.

Example 1:

Input:

nums = [0,1,2], queries = [[0,2],[2,0],[3,2],[5,0]]

Output:

[2,2,-1,0]

Explanation:

Minute 0: [0,1,2] - All elements are in the nums. Minute 1: [1,2] - The leftmost element, 0, is removed. Minute 2: [2] - The leftmost element, 1, is removed. Minute 3: [] - The leftmost element, 2, is removed. Minute 4: [0] - 0 is added to the end of nums. Minute 5: [0,1] - 1 is

added to the end of nums.

At minute 0, nums[2] is 2. At minute 2, nums[0] is 2. At minute 3, nums[2] does not exist. At minute 5, nums[0] is 0.

Example 2:

Input:

nums = [2], queries = [[0,0],[1,0],[2,0],[3,0]]

Output:

[2,-1,2,-1] Minute 0: [2] - All elements are in the nums. Minute 1: [] - The leftmost element, 2, is removed. Minute 2: [2] - 2 is added to the end of nums. Minute 3: [] - The leftmost element, 2, is removed.

At minute 0, nums[0] is 2. At minute 1, nums[0] does not exist. At minute 2, nums[0] is 2. At minute 3, nums[0] does not exist.

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 100

n == queries.length

1 <= n <= 10

5

queries[j].length == 2

0 <= time

j

<= 10

5

0 <= index

j

< nums.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> elementInNums(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] elementInNums(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

```
function elementInNums(nums: number[], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ElementInNums(int[] nums, int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func elementInNums(nums []int, queries [][]int) []int {
```

```
}
```

Kotlin:

```
class Solution {
fun elementInNums(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def element_in_nums(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
```

```
* @return Integer[]
*/
function elementInNums($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> elementInNums(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def elementInNums(nums: Array[Int], queries: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec element_in_nums(nums :: [integer], queries :: [[integer]]) :: [integer]
def element_in_nums(nums, queries) do

end
end
```

Erlang:

```
-spec element_in_nums(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
element_in_nums(Nums, Queries) ->
.
```

Racket:

```
(define/contract (element-in-nums nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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> elementInNums(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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[] elementInNums(int[] nums, int[][] queries) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Elements in Array After Removing and Replacing Elements
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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[][]} queries
* @return {number[]}
*/
var elementInNums = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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 elementInNums(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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[] ElementInNums(int[] nums, int[][] queries) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Elements in Array After Removing and Replacing Elements
* Difficulty: Medium
* Tags: array
*
* 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* elementInNums(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Elements in Array After Removing and Replacing Elements
// Difficulty: Medium
// Tags: array
//
// 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 elementInNums(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun elementInNums(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Elements in Array After Removing and Replacing Elements
// Difficulty: Medium
// Tags: array
//
// 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 element_in_nums(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def element_in_nums(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function elementInNums($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> elementInNums(List<int> nums, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def elementInNums(nums: Array[Int], queries: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec element_in_nums(nums :: [integer], queries :: [[integer]]) :: [integer]
def element_in_nums(nums, queries) do

end
end
```

Erlang Solution:

```
-spec element_in_nums(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
```

```
element_in_nums(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (element-in-nums nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```