Problem 3018: Maximum Number of Removal Queries That Can Be Processed I

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

and a

0-indexed

array

queries

.

You can do the following operation at the beginning

at most once

:

Replace

nums

with a

subsequence

of

nums

.

We start processing queries in the given order; for each query, we do the following:

If the first

and

the last element of

nums

is

less than

queries[i]

, the processing of queries

ends

.

Otherwise, we choose either the first

or

the last element of

nums

if it is

greater than or equal to

queries[i]

, and we

remove

the chosen element from

nums

.

Return

the

maximum

number of queries that can be processed by doing the operation optimally.

Example 1:

Input:

nums = [1,2,3,4,5], queries = [1,2,3,4,6]

Output:

4

Explanation:

We don't do any operation and process the queries as follows: 1- We choose and remove nums[0] since 1 <= 1, then nums becomes [2,3,4,5]. 2- We choose and remove nums[0] since 2 <= 2, then nums becomes [3,4,5]. 3- We choose and remove nums[0] since 3 <= 3, then nums becomes [4,5]. 4- We choose and remove nums[0] since 4 <= 4, then nums becomes [5]. 5- We can not choose any elements from nums since they are not greater than or equal to 5. Hence, the answer is 4. It can be shown that we can't process more than 4 queries.

Example 2:

Input:

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

Output:

3

Explanation:

We don't do any operation and process the queries as follows: 1- We choose and remove nums[0] since 2 <= 2, then nums becomes [3,2]. 2- We choose and remove nums[1] since 2 <= 2, then nums becomes [3]. 3- We choose and remove nums[0] since 3 <= 3, then nums becomes []. Hence, the answer is 3. It can be shown that we can't process more than 3 queries.

Example 3:

Input:

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

Output:

2

Explanation:

First we replace nums with the subsequence of nums [4,3]. Then we can process the queries as follows: 1- We choose and remove nums[0] since 4 <= 4, then nums becomes [3]. 2- We choose and remove nums[0] since 3 <= 3, then nums becomes []. 3- We can not process any more queries since nums is empty. Hence, the answer is 2. It can be shown that we can't process more than 2 queries.

Constraints:

1 <= nums.length <= 1000

1 <= queries.length <= 1000

1 <= nums[i], queries[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maximumProcessableQueries(int* nums, int numsSize, int* queries, int
queriesSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun maximumProcessableQueries(nums: IntArray, queries: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-processable-queries nums queries)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

Python3 Solution:

```
"""
Problem: Maximum Number of Removal Queries That Can Be Processed I
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumProcessableQueries(nums: number[], queries: number[]): number
{

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Removal Queries That Can Be Processed I
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maximumProcessableQueries(int* nums, int numsSize, int* queries, int
queriesSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Removal Queries That Can Be Processed I
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumProcessableQueries(nums []int, queries []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumProcessableQueries(nums: IntArray, queries: IntArray): Int {


}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Removal Queries That Can Be Processed I
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

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

Racket Solution:

```
(define/contract (maximum-processable-queries nums queries)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```