Problem 1950: Maximum of Minimum Values in All Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

. You are asked to solve

n

queries for each integer

i

in the range

0 <= i < n

.

To solve the

i

th

query:

Find the

minimum value

in each possible subarray of size

i + 1

of the array

nums

.

Find the

maximum

of those minimum values. This maximum is the

answer

to the query.

Return

a

0-indexed

integer array

ans

of size

n

such that

ans[i]

is the answer to the

i

th

query

.

A

subarray

is a contiguous sequence of elements in an array.

Example 1:

Input:

nums = [0,1,2,4]

Output:

[4,2,1,0]

Explanation:

i=0: - The subarrays of size 1 are [0], [1], [2], [4]. The minimum values are 0, 1, 2, 4. - The maximum of the minimum values is 4. i=1: - The subarrays of size 2 are [0,1], [1,2], [2,4]. The minimum values are 0, 1, 2. - The maximum of the minimum values is 2. i=2: - The subarrays

of size 3 are [0,1,2], [1,2,4]. The minimum values are 0, 1. - The maximum of the minimum values is 1. i=3: - There is one subarray of size 4, which is [0,1,2,4]. The minimum value is 0. - There is only one value, so the maximum is 0.

Example 2:

Input:

nums = [10,20,50,10]

Output:

[50,20,10,10]

Explanation:

i=0: - The subarrays of size 1 are [10], [20], [50], [10]. The minimum values are 10, 20, 50, 10. - The maximum of the minimum values is 50. i=1: - The subarrays of size 2 are [10,20], [20,50], [50,10]. The minimum values are 10, 20, 10. - The maximum of the minimum values is 20. i=2: - The subarrays of size 3 are [10,20,50], [20,50,10]. The minimum values are 10, 10. - The maximum of the minimum values is 10. i=3: - There is one subarray of size 4, which is [10,20,50,10]. The minimum value is 10. - There is only one value, so the maximum is 10.

Constraints:

n == nums.length

1 <= n <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-maximums nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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> findMaximums(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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[] findMaximums(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum of Minimum Values in All Subarrays
Difficulty: Medium
Tags: array, 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 findMaximums(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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 findMaximums = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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 findMaximums(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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[] FindMaximums(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum of Minimum Values in All Subarrays
* Difficulty: Medium
* Tags: array, 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* findMaximums(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum of Minimum Values in All Subarrays
// Difficulty: Medium
// Tags: array, 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 findMaximums(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum of Minimum Values in All Subarrays
// Difficulty: Medium
// Tags: array, 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 find_maximums(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-maximums nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```