Problem 2948: Make Lexicographically Smallest Array by Swapping Elements

Problem Information

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

Problem Description

You are given a

0-indexed

array of

positive

integers

nums

and a

positive

integer

limit

.

In one operation, you can choose any two indices

i

and

j

and swap

nums[i]

and

nums[j]

if

|nums[i] - nums[j]| <= limit

.

Return

the

lexicographically smallest array

that can be obtained by performing the operation any number of times

.

An array

a

is lexicographically smaller than an array

b

if in the first position where

a

and

b

differ, array

a

has an element that is less than the corresponding element in

b

. For example, the array

[2,10,3]

is lexicographically smaller than the array

[10,2,3]

because they differ at index

0

and

2 < 10

.

Example 1:

Input:

nums = [1,5,3,9,8], limit = 2

Output:

[1,3,5,8,9]

Explanation:

Apply the operation 2 times: - Swap nums[1] with nums[2]. The array becomes [1,3,5,9,8] - Swap nums[3] with nums[4]. The array becomes [1,3,5,8,9] We cannot obtain a lexicographically smaller array by applying any more operations. Note that it may be possible to get the same result by doing different operations.

Example 2:

Input:

nums = [1,7,6,18,2,1], limit = 3

Output:

[1,6,7,18,1,2]

Explanation:

Apply the operation 3 times: - Swap nums[1] with nums[2]. The array becomes [1,6,7,18,2,1] - Swap nums[0] with nums[4]. The array becomes [2,6,7,18,1,1] - Swap nums[0] with nums[5]. The array becomes [1,6,7,18,1,2] We cannot obtain a lexicographically smaller array by applying any more operations.

Example 3:

Input:

nums = [1,7,28,19,10], limit = 3

Output:

[1,7,28,19,10]

Explanation:

[1,7,28,19,10] is the lexicographically smallest array we can obtain because we cannot apply the operation on any two indices.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= limit <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> lexicographicallySmallestArray(vector<int>& nums, int limit) {

}
};
```

Java:

```
class Solution {
public int[] lexicographicallySmallestArray(int[] nums, int limit) {

}
}
```

Python3:

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

```
List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function lexicographicallySmallestArray(nums: number[], limit: number):
number[] {

};
```

C#:

```
public class Solution {
public int[] LexicographicallySmallestArray(int[] nums, int limit) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* lexicographicallySmallestArray(int* nums, int numsSize, int limit, int*
returnSize) {

}
```

Go:

```
func lexicographicallySmallestArray(nums []int, limit int) []int {

}
```

Kotlin:

```
class Solution {
fun lexicographicallySmallestArray(nums: IntArray, limit: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @return {Integer[]}
def lexicographically_smallest_array(nums, limit)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @return Integer[]
*/
function lexicographicallySmallestArray($nums, $limit) {

}
}
```

Dart:

```
class Solution {
List<int> lexicographicallySmallestArray(List<int> nums, int limit) {

}
}
```

Scala:

```
object Solution {
def lexicographicallySmallestArray(nums: Array[Int], limit: Int): Array[Int]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec lexicographically_smallest_array(nums :: [integer], limit :: integer)
:: [integer]
def lexicographically_smallest_array(nums, limit) do

end
```

```
end
```

Erlang:

```
-spec lexicographically_smallest_array(Nums :: [integer()], Limit ::
integer()) -> [integer()].
lexicographically_smallest_array(Nums, Limit) ->
.
```

Racket:

```
(define/contract (lexicographically-smallest-array nums limit)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Lexicographically Smallest Array by Swapping Elements
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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> lexicographicallySmallestArray(vector<int>& nums, int limit) {

}
};
```

Java Solution:

```
/**
* Problem: Make Lexicographically Smallest Array by Swapping Elements
```

```
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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[] lexicographicallySmallestArray(int[] nums, int limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Lexicographically Smallest Array by Swapping Elements
Difficulty: Medium
Tags: array, graph, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Lexicographically Smallest Array by Swapping Elements
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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} limit
* @return {number[]}
*/
var lexicographicallySmallestArray = function(nums, limit) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Lexicographically Smallest Array by Swapping Elements
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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 lexicographicallySmallestArray(nums: number[], limit: number):
number[] {

};
```

C# Solution:

```
/*
* Problem: Make Lexicographically Smallest Array by Swapping Elements
```

```
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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[] LexicographicallySmallestArray(int[] nums, int limit) {

}
}
```

C Solution:

```
/*
* Problem: Make Lexicographically Smallest Array by Swapping Elements
* Difficulty: Medium
* Tags: array, graph, sort
*
* 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* lexicographicallySmallestArray(int* nums, int numsSize, int limit, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Make Lexicographically Smallest Array by Swapping Elements
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 lexicographicallySmallestArray(nums []int, limit int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun lexicographicallySmallestArray(nums: IntArray, limit: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Lexicographically Smallest Array by Swapping Elements
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 lexicographically_smallest_array(nums: Vec<i32>, limit: i32) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @return {Integer[]}
def lexicographically_smallest_array(nums, limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @return Integer[]
*/
function lexicographicallySmallestArray($nums, $limit) {

}
}
```

Dart Solution:

```
class Solution {
List<int> lexicographicallySmallestArray(List<int> nums, int limit) {

}
}
```

Scala Solution:

```
object Solution {
def lexicographicallySmallestArray(nums: Array[Int], limit: Int): Array[Int]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec lexicographically_smallest_array(nums :: [integer], limit :: integer)
```

```
:: [integer]
def lexicographically_smallest_array(nums, limit) do

end
end
```

Erlang Solution:

```
-spec lexicographically_smallest_array(Nums :: [integer()], Limit ::
integer()) -> [integer()].
lexicographically_smallest_array(Nums, Limit) ->
.
```

Racket Solution:

```
(define/contract (lexicographically-smallest-array nums limit)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```