Problem 3605: Minimum Stability Factor of Array

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

maxC

.

A

subarray

is called

stable

if the

highest common factor (HCF)

of all its elements is

greater than or equal to

2.

The

stability factor

of an array is defined as the length of its

longest

stable subarray.

You may modify

at most

maxC

elements of the array to any integer.

Return the

minimum

possible stability factor of the array after at most

maxC

modifications. If no stable subarray remains, return 0.

Note:

The

highest common factor (HCF)

of an array is the largest integer that evenly divides all the array elements.

A

subarray

of length 1 is stable if its only element is greater than or equal to 2, since

HCF([x]) = x

.

Example 1:

Input:

nums = [3,5,10], maxC = 1

Output:

1

Explanation:

The stable subarray

[5, 10]

has

HCF = 5

, which has a stability factor of 2.

Since

maxC = 1

, one optimal strategy is to change

nums[1]

to

7

, resulting in

nums = [3, 7, 10]

.

Now, no subarray of length greater than 1 has

HCF >= 2

. Thus, the minimum possible stability factor is 1.

Example 2:

Input:

nums = [2,6,8], maxC = 2

Output:

1

Explanation:

The subarray

[2, 6, 8]

has

HCF = 2

, which has a stability factor of 3.

Since

maxC = 2

, one optimal strategy is to change

nums[1]

to 3 and

nums[2]

to 5, resulting in

nums = [2, 3, 5]

.

Now, no subarray of length greater than 1 has

HCF >= 2

. Thus, the minimum possible stability factor is 1.

Example 3:

Input:

nums = [2,4,9,6], maxC = 1

Output:

2

Explanation:

The stable subarrays are:

[2, 4]

with

HCF = 2

and stability factor of 2.

[9, 6]

with

HCF = 3

and stability factor of 2.

Since

maxC = 1

, the stability factor of 2 cannot be reduced due to two separate stable subarrays. Thus, the minimum possible stability factor is 2.

Constraints:

1 <= n == nums.length <= 10

5

1 <= nums[i] <= 10

9

0 <= maxC <= n

Code Snippets

C++:

```
class Solution {
public:
int minStable(vector<int>& nums, int maxC) {

}
};
```

Java:

```
class Solution {
public int minStable(int[] nums, int maxC) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minStable(nums: number[], maxC: number): number {

};
```

C#:

```
public class Solution {
public int MinStable(int[] nums, int maxC) {

}
}
```

C:

```
int minStable(int* nums, int numsSize, int maxC) {

}
```

Go:

```
func minStable(nums []int, maxC int) int {

}
```

Kotlin:

```
class Solution {
fun minStable(nums: IntArray, maxC: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} max_c
# @return {Integer}
def min_stable(nums, max_c)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maxC
* @return Integer
*/
function minStable($nums, $maxC) {

}
}
```

Dart:

```
class Solution {
int minStable(List<int> nums, int maxC) {

}
}
```

Scala:

```
object Solution {
def minStable(nums: Array[Int], maxC: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec min_stable(nums :: [integer], max_c :: integer) :: integer
def min_stable(nums, max_c) do

end
end
```

Erlang:

```
-spec min_stable(Nums :: [integer()], MaxC :: integer()) -> integer().
min_stable(Nums, MaxC) ->
.
```

Racket:

```
(define/contract (min-stable nums maxC)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minStable(vector<int>& nums, int maxC) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minStable(int[] nums, int maxC) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Stability Factor of Array
Difficulty: Hard
Tags: array, tree, greedy, math, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minStable(nums: number[], maxC: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinStable(int[] nums, int maxC) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Stability Factor of Array
* Difficulty: Hard
* Tags: array, tree, greedy, math, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minStable(int* nums, int numsSize, int maxC) {

}
```

Go Solution:

```
// Problem: Minimum Stability Factor of Array
// Difficulty: Hard
```

```
// Tags: array, tree, greedy, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minStable(nums []int, maxC int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minStable(nums: IntArray, maxC: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Stability Factor of Array
// Difficulty: Hard
// Tags: array, tree, greedy, math, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_stable(nums: Vec<i32>, max_c: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} max_c
# @return {Integer}
def min_stable(nums, max_c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maxC
* @return Integer
*/
function minStable($nums, $maxC) {

}
}
```

Dart Solution:

```
class Solution {
int minStable(List<int> nums, int maxC) {

}
}
```

Scala Solution:

```
object Solution {
def minStable(nums: Array[Int], maxC: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_stable(nums :: [integer], max_c :: integer) :: integer
def min_stable(nums, max_c) do

end
end
```

Erlang Solution:

```
-spec min_stable(Nums :: [integer()], MaxC :: integer()) -> integer().
min_stable(Nums, MaxC) ->
.
```

Racket Solution:

```
(define/contract (min-stable nums maxC)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```