

Problem 2892: Minimizing Array After Replacing Pairs With Their Product

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an integer array

nums

and an integer

k

, you can perform the following operation on the array any number of times:

Select two

adjacent

elements of the array like

x

and

y

, such that

$x * y \leq k$

, and replace both of them with a

single element

with value

$x * y$

(e.g. in one operation the array

[1, 2, 2, 3]

with

$k = 5$

can become

[1, 4, 3]

or

[2, 2, 3]

, but can't become

[1, 2, 6]

).

Return

the

minimum

possible length of

nums

after any number of operations

.

Example 1:

Input:

nums = [2,3,3,7,3,5], k = 20

Output:

3

Explanation:

We perform these operations: 1. [

2,3

,3,7,3,5] -> [

6

,3,7,3,5] 2. [

6,3

,7,3,5] -> [

18

,7,3,5] 3. [18,7,

3,5

] -> [18,7,

15

] It can be shown that 3 is the minimum length possible to achieve with the given operation.

Example 2:

Input:

nums = [3,3,3,3], k = 6

Output:

4

Explanation:

We can't perform any operations since the product of every two adjacent elements is greater than 6. Hence, the answer is 4.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$0 \leq \text{nums}[i] \leq 10$

9

$1 \leq k \leq 10$

9

Code Snippets

C++:

```
class Solution {  
public:  
    int minArrayLength(vector<int>& nums, int k) {  
  
    }  
};
```

Java:

```
class Solution {  
public int minArrayLength(int[] nums, int k) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {number}  
 */  
var minArrayLength = function(nums, k) {  
  
};
```

TypeScript:

```
function minArrayLength(nums: number[], k: number): number {  
}  
};
```

C#:

```
public class Solution {  
    public int MinArrayLength(int[] nums, int k) {  
        }  
    }  
}
```

C:

```
int minArrayLength(int* nums, int numsSize, int k) {  
}  
}
```

Go:

```
func minArrayLength(nums []int, k int) int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun minArrayLength(nums: IntArray, k: Int): Int {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func minArrayLength(_ nums: [Int], _ k: Int) -> Int {  
        }  
    }  
}
```

Rust:

```
impl Solution {
    pub fn min_array_length(nums: Vec<i32>, k: i32) -> i32 {
        }
    }
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_array_length(nums, k)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function minArrayLength($nums, $k) {

    }
}
```

Dart:

```
class Solution {
    int minArrayLength(List<int> nums, int k) {
        }
    }
```

Scala:

```
object Solution {
    def minArrayLength(nums: Array[Int], k: Int): Int = {
        }
```

```
}
```

Elixir:

```
defmodule Solution do
  @spec min_array_length(nums :: [integer], k :: integer) :: integer
  def min_array_length(nums, k) do
    end
  end
```

Erlang:

```
-spec min_array_length(Nums :: [integer()], K :: integer()) -> integer().
min_array_length(Nums, K) ->
  .
```

Racket:

```
(define/contract (min-array-length nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimizing Array After Replacing Pairs With Their Product
 * Difficulty: Medium
 * Tags: array, dp, greedy
 *
 * 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 minArrayLength(vector<int>& nums, int k) {
```

```
}
```

```
} ;
```

Java Solution:

```
/**  
 * Problem: Minimizing Array After Replacing Pairs With Their Product  
 * Difficulty: Medium  
 * Tags: array, dp, greedy  
 *  
 * 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 minArrayLength(int[] nums, int k) {  
        // Implementation  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Minimizing Array After Replacing Pairs With Their Product  
Difficulty: Medium  
Tags: array, dp, greedy  
  
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 minArrayLength(self, nums: List[int], k: int) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

JavaScript Solution:

```
/**  
 * Problem: Minimizing Array After Replacing Pairs With Their Product  
 * Difficulty: Medium  
 * Tags: array, dp, greedy  
 *  
 * 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} k  
 * @return {number}  
 */  
var minArrayLength = function(nums, k) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Minimizing Array After Replacing Pairs With Their Product  
 * Difficulty: Medium  
 * Tags: array, dp, greedy  
 *  
 * 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 minArrayLength(nums: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
 * Problem: Minimizing Array After Replacing Pairs With Their Product
 * Difficulty: Medium
 * Tags: array, dp, greedy
 *
 * 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 MinArrayLength(int[] nums, int k) {
        return 0;
    }
}
```

C Solution:

```
/*
 * Problem: Minimizing Array After Replacing Pairs With Their Product
 * Difficulty: Medium
 * Tags: array, dp, greedy
 *
 * 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 minArrayLength(int* nums, int numsSize, int k) {
    return 0;
}
```

Go Solution:

```
// Problem: Minimizing Array After Replacing Pairs With Their Product
// Difficulty: Medium
```

```

// Tags: array, dp, greedy
//
// 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 minArrayLength(nums []int, k int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun minArrayLength(nums: IntArray, k: Int): Int {
        return 0
    }
}

```

Swift Solution:

```

class Solution {
    func minArrayLength(_ nums: [Int], _ k: Int) -> Int {
        return 0
    }
}

```

Rust Solution:

```

// Problem: Minimizing Array After Replacing Pairs With Their Product
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 min_array_length(nums: Vec<i32>, k: i32) -> i32 {
        return 0
    }
}

```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_array_length(nums, k)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function minArrayLength($nums, $k) {

    }
}
```

Dart Solution:

```
class Solution {
  int minArrayLength(List<int> nums, int k) {
    }
}
```

Scala Solution:

```
object Solution {
  def minArrayLength(nums: Array[Int], k: Int): Int = {
    }
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_array_length(nums :: [integer], k :: integer) :: integer
def min_array_length(nums, k) do

end
end
```

Erlang Solution:

```
-spec min_array_length(Nums :: [integer()], K :: integer()) -> integer().
min_array_length(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (min-array-length nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?))
```