Problem 3574: Maximize Subarray GCD Score

Problem Information

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

Problem Description

You are given an array of positive integers

nums

and an integer

k

.

You may perform at most

k

operations. In each operation, you can choose one element in the array and

double

its value. Each element can be doubled

at most

once.

The

score

of a contiguous

subarray

is defined as the

product

of its length and the

greatest common divisor (GCD)

of all its elements.

Your task is to return the

maximum

score

that can be achieved by selecting a contiguous subarray from the modified array.

Note:

The

greatest common divisor (GCD)

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

Example 1:

Input:

nums = [2,4], k = 1

Output:

8

Explanation:

Double

nums[0]

to 4 using one operation. The modified array becomes

[4, 4]

.

The GCD of the subarray

[4, 4]

is 4, and the length is 2.

Thus, the maximum possible score is

2 × 4 = 8

.

Example 2:

Input:

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

Output:

14

Explanation:

Double

nums[2]

to 14 using one operation. The modified array becomes

[3, 5, 14]

.

The GCD of the subarray

[14]

is 14, and the length is 1.

Thus, the maximum possible score is

1 × 14 = 14

.

Example 3:

Input:

nums = [5,5,5], k = 1

Output:

15

Explanation:

The subarray

[5, 5, 5]

has a GCD of 5, and its length is 3.

Since doubling any element doesn't improve the score, the maximum score is

3 × 5 = 15

.

Constraints:

1 <= n == nums.length <= 1500

1 <= nums[i] <= 10

9

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
long long maxGCDScore(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long maxGCDScore(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxGCDScore(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MaxGCDScore(int[] nums, int k) {

}
}
```

C:

```
long long maxGCDScore(int* nums, int numsSize, int k) {

}
```

Go:

```
func maxGCDScore(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxGCDScore(nums: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_gcd_score(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
```

```
* @param Integer $k
* @return Integer
*/
function maxGCDScore($nums, $k) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-gcd-score nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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:
long long maxGCDScore(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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 long maxGCDScore(int[] nums, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximize Subarray GCD Score
Difficulty: Hard
Tags: array, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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} k
* @return {number}
*/
var maxGCDScore = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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 maxGCDScore(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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 long MaxGCDScore(int[] nums, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximize Subarray GCD Score
* Difficulty: Hard
* Tags: array, math
*
* 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
*/

long long maxGCDScore(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximize Subarray GCD Score
// Difficulty: Hard
// Tags: array, math
//
// 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 maxGCDScore(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxGCDScore(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Subarray GCD Score
// Difficulty: Hard
// Tags: array, math
//
// 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 max_gcd_score(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-gcd-score nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```