Problem 2447: Number of Subarrays With GCD Equal to K

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, return

the number of

subarrays

of

nums

where the greatest common divisor of the subarray's elements is

k

.

A

subarray

is a contiguous non-empty sequence of elements within an array.

The

greatest common divisor of an array

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

Example 1:

Input:

nums = [9,3,1,2,6,3], k = 3

Output:

4

Explanation:

The subarrays of nums where 3 is the greatest common divisor of all the subarray's elements are: - [9,

3

,1,2,6,3] - [9,3,1,2,6,

3

] - [

9,3

,1,2,6,3] - [9,3,1,2,

6,3

]

Example 2:

Input:

nums = [4], k = 7

Output:

0

Explanation:

There are no subarrays of nums where 7 is the greatest common divisor of all the subarray's elements.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i], k <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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


}
}
```

C:

```
int subarrayGCD(int* nums, int numsSize, int k) {

}
```

Go:

```
func subarrayGCD(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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:
int subarrayGCD(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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 int subarrayGCD(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Subarrays With GCD Equal to K
Difficulty: Medium
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 subarrayGCD(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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 subarrayGCD = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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 subarrayGCD(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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 int SubarrayGCD(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Subarrays With GCD Equal to K
* Difficulty: Medium
* 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
*/

int subarrayGCD(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Number of Subarrays With GCD Equal to K
// Difficulty: Medium
// 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 subarrayGCD(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Subarrays With GCD Equal to K
// Difficulty: Medium
// 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 subarray_gcd(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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