Problem 3671: Sum of Beautiful Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

For every

positive

integer

g

, we define the

beauty

of

g

as the

product

of

g

and the number of

strictly increasing

subsequences

of

nums

whose greatest common divisor (GCD) is exactly

g

.

Return the

sum

of

beauty

values for all positive integers

g

.

Since the answer could be very large, return it modulo

10

9

+ 7

.

Example 1:

Input:

nums = [1,2,3]

Output:

10

Explanation:

All strictly increasing subsequences and their GCDs are:

Subsequence

GCD

[1]

1

[2]

2

[3]

3

[1,2]

1

[1,3]

1

[2,3]

1

[1,2,3]

1

Calculating beauty for each GCD:

GCD

Count of subsequences

Beauty (GCD × Count)

1

5

1 × 5 = 5

2

1

2 × 1 = 2

3

1

3 × 1 = 3

Total beauty is

5 + 2 + 3 = 10

.

Example 2:

Input:

nums = [4,6]

Output:

12

Explanation:

All strictly increasing subsequences and their GCDs are:

Subsequence

GCD

[4]

4

[6]

6

[4,6]

2

Calculating beauty for each GCD:

GCD

Count of subsequences

Beauty (GCD × Count)

2

1

2 × 1 = 2

4

1

4 × 1 = 4

6

1

6 × 1 = 6

Total beauty is

2 + 4 + 6 = 12

.

Constraints:

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

4

1 <= nums[i] <= 7 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int totalBeauty(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int totalBeauty(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var totalBeauty = function(nums) {

};
```

TypeScript:

```
function totalBeauty(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int TotalBeauty(int[] nums) {

}
}
```

C:

```
int totalBeauty(int* nums, int numsSize) {

}
```

Go:

```
func totalBeauty(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun totalBeauty(nums: IntArray): Int {

}
}
```

Swift:

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

Rust:

```
impl Solution {
pub fn total_beauty(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def total_beauty(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function totalBeauty($nums) {

}
}
```

Dart:

```
class Solution {
int totalBeauty(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def totalBeauty(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec total_beauty(nums :: [integer]) :: integer
def total_beauty(nums) do

end
end
```

Erlang:

```
-spec total_beauty(Nums :: [integer()]) -> integer().
total_beauty(Nums) ->
.
```

Racket:

```
(define/contract (total-beauty nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 totalBeauty(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 totalBeauty(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Beautiful Subsequences
Difficulty: Hard
Tags: array, tree, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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
* @return {number}
*/
var totalBeauty = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 totalBeauty(nums: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 TotalBeauty(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Beautiful Subsequences
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 totalBeauty(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Beautiful Subsequences
// Difficulty: Hard
// Tags: array, tree, math
//
// 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 totalBeauty(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun totalBeauty(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func totalBeauty(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum of Beautiful Subsequences
// Difficulty: Hard
// Tags: array, tree, math
//
// 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 total_beauty(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} nums
# @return {Integer}
def total_beauty(nums)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function totalBeauty($nums) {

}
}
```

Dart Solution:

```dart
class Solution {
int totalBeauty(List<int> nums) {

}
}
```

Scala Solution:

```scala
object Solution {
def totalBeauty(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec total_beauty(nums :: [integer]) :: integer
def total_beauty(nums) do
```

```
end
end
```

Erlang Solution:

```
-spec total_beauty(Nums :: [integer()]) -> integer().
total_beauty(Nums) ->
.
```

Racket Solution:

```
(define/contract (total-beauty nums)
(-> (listof exact-integer?) exact-integer?)
)
```