Problem 1808: Maximize Number of Nice Divisors

Problem Information

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

Problem Description

You are given a positive integer

primeFactors

. You are asked to construct a positive integer

n

that satisfies the following conditions:

The number of prime factors of

n

(not necessarily distinct) is

at most

primeFactors

.

The number of nice divisors of

n

is maximized. Note that a divisor of

n

is

nice

if it is divisible by every prime factor of

n

. For example, if

n = 12

, then its prime factors are

[2,2,3]

, then

6

and

12

are nice divisors, while

3

and

4

are not.

Return

the number of nice divisors of

n

. Since that number can be too large, return it

modulo

10

9

+ 7

.

Note that a prime number is a natural number greater than

1

that is not a product of two smaller natural numbers. The prime factors of a number

n

is a list of prime numbers such that their product equals

n

.

Example 1:

Input:

primeFactors = 5

Output:

6

Explanation:

200 is a valid value of n. It has 5 prime factors: [2,2,2,5,5], and it has 6 nice divisors: [10,20,40,50,100,200]. There is not other value of n that has at most 5 prime factors and more nice divisors.

Example 2:

Input:

primeFactors = 8

Output:

18

Constraints:

1 <= primeFactors <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxNiceDivisors(int primeFactors) {

}
};
```

Java:

```
class Solution {
public int maxNiceDivisors(int primeFactors) {
```

```
}
}
```

Python3:

```
class Solution:
def maxNiceDivisors(self, primeFactors: int) -> int:
```

Python:

```
class Solution(object):
def maxNiceDivisors(self, primeFactors):
"""
:type primeFactors: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} primeFactors
* @return {number}
*/
var maxNiceDivisors = function(primeFactors) {

};
```

TypeScript:

```
function maxNiceDivisors(primeFactors: number): number {

};
```

C#:

```
public class Solution {
public int MaxNiceDivisors(int primeFactors) {

}
}
```

C:

```
int maxNiceDivisors(int primeFactors){

}
```

Go:

```
func maxNiceDivisors(primeFactors int) int {

}
```

Kotlin:

```
class Solution {
fun maxNiceDivisors(primeFactors: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxNiceDivisors(_ primeFactors: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_nice_divisors(prime_factors: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} prime_factors
# @return {Integer}
def max_nice_divisors(prime_factors)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $primeFactors
* @return Integer
*/
function maxNiceDivisors($primeFactors) {

}
}
```

Scala:

```
object Solution {
def maxNiceDivisors(primeFactors: Int): Int = {

}
}
```

Racket:

```
(define/contract (max-nice-divisors primeFactors)
(-> exact-integer? exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maxNiceDivisors(int primeFactors) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxNiceDivisors(int primeFactors) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Number of Nice Divisors
Difficulty: Hard
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""
```

```
class Solution:
def maxNiceDivisors(self, primeFactors: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxNiceDivisors(self, primeFactors):
"""
:type primeFactors: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} primeFactors
* @return {number}
*/
var maxNiceDivisors = function(primeFactors) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
```

```
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxNiceDivisors(primeFactors: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaxNiceDivisors(int primeFactors) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Number of Nice Divisors
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
int maxNiceDivisors(int primeFactors){

}
```

Go Solution:

```
// Problem: Maximize Number of Nice Divisors
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func maxNiceDivisors(primeFactors int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxNiceDivisors(primeFactors: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxNiceDivisors(_ primeFactors: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Number of Nice Divisors
// Difficulty: Hard
// Tags: math
//
```

```
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_nice_divisors(prime_factors: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} prime_factors
# @return {Integer}
def max_nice_divisors(prime_factors)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $primeFactors
* @return Integer
*/
function maxNiceDivisors($primeFactors) {

}
}
```

Scala Solution:

```
object Solution {
def maxNiceDivisors(primeFactors: Int): Int = {

}
}
```

Racket Solution:

```
(define/contract (max-nice-divisors primeFactors)
(-> exact-integer? exact-integer?)

)
```