

Problem 625: Minimum Factorization

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a positive integer num, return

the smallest positive integer

x

whose multiplication of each digit equals

num

. If there is no answer or the answer is not fit in

32-bit

signed integer, return

0

.

Example 1:

Input:

num = 48

Output:

68

Example 2:

Input:

num = 15

Output:

35

Constraints:

1 <= num <= 2

31

- 1

Code Snippets

C++:

```
class Solution {  
public:  
    int smallestFactorization(int num) {  
  
    }  
};
```

Java:

```
class Solution {  
public int smallestFactorization(int num) {  
  
}
```

```
}
```

Python3:

```
class Solution:  
    def smallestFactorization(self, num: int) -> int:
```

Python:

```
class Solution(object):  
    def smallestFactorization(self, num):  
        """  
        :type num: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number} num  
 * @return {number}  
 */  
var smallestFactorization = function(num) {  
  
};
```

TypeScript:

```
function smallestFactorization(num: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int SmallestFactorization(int num) {  
  
    }  
}
```

C:

```
int smallestFactorization(int num) {  
  
}
```

Go:

```
func smallestFactorization(num int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun smallestFactorization(num: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func smallestFactorization(_ num: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn smallest_factorization(num: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} num  
# @return {Integer}  
def smallest_factorization(num)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $num  
     * @return Integer  
     */  
    function smallestFactorization($num) {  
  
    }  
}
```

Dart:

```
class Solution {  
int smallestFactorization(int num) {  
  
}  
}
```

Scala:

```
object Solution {  
def smallestFactorization(num: Int): Int = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec smallest_factorization(non_neg_integer()) :: integer()  
def smallest_factorization(num) do  
  
end  
end
```

Erlang:

```
-spec smallest_factorization(non_neg_integer()) -> integer().  
smallest_factorization(Num) ->  
.
```

Racket:

```
(define/contract (smallest-factorization num)
  (-> exact-integer? exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimum Factorization
 * Difficulty: Medium
 * Tags: greedy, math
 *
 * Approach: Greedy algorithm with local optimal choices
 * 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 smallestFactorization(int num) {

    }
};
```

Java Solution:

```
/**
 * Problem: Minimum Factorization
 * Difficulty: Medium
 * Tags: greedy, math
 *
 * Approach: Greedy algorithm with local optimal choices
 * 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 smallestFactorization(int num) {

    }
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Factorization
Difficulty: Medium
Tags: greedy, math

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

    def smallestFactorization(self, num: int) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):

    def smallestFactorization(self, num):
        """
        :type num: int
        :rtype: int
        """
```

JavaScript Solution:

```
/**
 * Problem: Minimum Factorization
 * Difficulty: Medium
 * Tags: greedy, math
 *
 * Approach: Greedy algorithm with local optimal choices
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
```

```
* @param {number} num
* @return {number}
*/
var smallestFactorization = function(num) {

};
```

TypeScript Solution:

```
/** 
* Problem: Minimum Factorization
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
function smallestFactorization(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Factorization
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* 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 SmallestFactorization(int num) {
        return 0;
    }
}
```

C Solution:

```
/*
 * Problem: Minimum Factorization
 * Difficulty: Medium
 * Tags: greedy, math
 *
 * Approach: Greedy algorithm with local optimal choices
 * Time Complexity: O(n) to O(n^2) depending on approach
 * Space Complexity: O(1) to O(n) depending on approach
 */

int smallestFactorization(int num) {

}
```

Go Solution:

```
// Problem: Minimum Factorization
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func smallestFactorization(num int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun smallestFactorization(num: Int): Int {
        return 0
    }
}
```

Swift Solution:

```
class Solution {
    func smallestFactorization(_ num: Int) -> Int {
```

```
}
```

```
}
```

Rust Solution:

```
// Problem: Minimum Factorization
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// 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 smallest_factorization(num: i32) -> i32 {
        ...
    }
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer}
def smallest_factorization(num)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer $num
     * @return Integer
     */
    function smallestFactorization($num) {

    }
}
```

Dart Solution:

```
class Solution {  
    int smallestFactorization(int num) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def smallestFactorization(num: Int) = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec smallest_factorization(non_neg_integer) :: [non_neg_integer]  
    def smallest_factorization(num) do  
  
    end  
end
```

Erlang Solution:

```
-spec smallest_factorization(non_neg_integer()) -> [non_neg_integer].  
smallest_factorization(Num) ->  
.
```

Racket Solution:

```
(define/contract (smallest-factorization num)  
  (-> exact-integer? exact-integer?)  
)
```