

# Problem 2269: Find the K-Beauty of a Number

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

The

k-beauty

of an integer

num

is defined as the number of

substrings

of

num

when it is read as a string that meet the following conditions:

It has a length of

k

.

It is a divisor of

num

Given integers

num

and

k

, return

the k-beauty of

num

Note:

Leading zeros

are allowed.

0

is not a divisor of any value.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

num = 240, k = 2

Output:

2

Explanation:

The following are the substrings of num of length k: - "24" from "

24

0": 24 is a divisor of 240. - "40" from "2

40

": 40 is a divisor of 240. Therefore, the k-beauty is 2.

Example 2:

Input:

num = 430043, k = 2

Output:

2

Explanation:

The following are the substrings of num of length k: - "43" from "

43

0043": 43 is a divisor of 430043. - "30" from "4

30

043": 30 is not a divisor of 430043. - "00" from "43

00

43": 0 is not a divisor of 430043. - "04" from "430

04

3": 4 is not a divisor of 430043. - "43" from "4300

43

": 43 is a divisor of 430043. Therefore, the k-beauty is 2.

Constraints:

$1 \leq num \leq 10^9$

9

$1 \leq k \leq num.length$

(taking

num

as a string)

## Code Snippets

C++:

```
class Solution {
public:
    int divisorSubstrings(int num, int k) {
        }
};
```

**Java:**

```
class Solution {  
    public int divisorSubstrings(int num, int k) {  
  
    }  
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

**TypeScript:**

```
function divisorSubstrings(num: number, k: number): number {  
  
};
```

**C#:**

```
public class Solution {  
    public int DivisorSubstrings(int num, int k) {  
  
    }  
}
```

**C:**

```
int divisorSubstrings(int num, int k) {  
  
}
```

**Go:**

```
func divisorSubstrings(num int, k int) int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun divisorSubstrings(num: Int, k: Int): Int {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func divisorSubstrings(_ num: Int, _ k: Int) -> Int {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn divisor_substrings(num: i32, k: i32) -> i32 {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer} num
# @param {Integer} k
# @return {Integer}
def divisor_substrings(num, k)

end
```

### PHP:

```
class Solution {

    /**
     * @param Integer $num
     * @param Integer $k
     * @return Integer
     */
    function divisorSubstrings($num, $k) {

    }
}
```

### Dart:

```
class Solution {
    int divisorSubstrings(int num, int k) {
    }
}
```

### Scala:

```
object Solution {
    def divisorSubstrings(num: Int, k: Int): Int = {
    }
}
```

### Elixir:

```
defmodule Solution do
  @spec divisor_substrings(non :: integer, k :: integer) :: integer
  def divisor_substrings(non, k) do
```

```
end  
end
```

### Erlang:

```
-spec divisor_substrings(Num :: integer(), K :: integer()) -> integer().  
divisor_substrings(Num, K) ->  
.
```

### Racket:

```
(define/contract (divisor-substrings num k)  
  (-> exact-integer? exact-integer? exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Find the K-Beauty of a Number  
 * Difficulty: Easy  
 * Tags: array, string, 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 divisorSubstrings(int num, int k) {  
  
    }  
};
```

### Java Solution:

```
/**  
 * Problem: Find the K-Beauty of a Number
```

```

* Difficulty: Easy
* Tags: array, string, 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 divisorSubstrings(int num, int k) {
}
}

```

### Python3 Solution:

```

"""
Problem: Find the K-Beauty of a Number
Difficulty: Easy
Tags: array, string, 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 divisorSubstrings(self, num: int, k: int) -> int:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def divisorSubstrings(self, num, k):
        """
        :type num: int
        :type k: int
        :rtype: int
        """

```

### JavaScript Solution:

```
/**  
 * Problem: Find the K-Beauty of a Number  
 * Difficulty: Easy  
 * Tags: array, string, 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} num  
 * @param {number} k  
 * @return {number}  
 */  
var divisorSubstrings = function(num, k) {  
  
};
```

### TypeScript Solution:

```
/**  
 * Problem: Find the K-Beauty of a Number  
 * Difficulty: Easy  
 * Tags: array, string, 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 divisorSubstrings(num: number, k: number): number {  
  
};
```

### C# Solution:

```
/*  
 * Problem: Find the K-Beauty of a Number  
 * Difficulty: Easy  
 * Tags: array, string, 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 DivisorSubstrings(int num, int k) {

    }
}

```

### C Solution:

```

/*
 * Problem: Find the K-Beauty of a Number
 * Difficulty: Easy
 * Tags: array, string, 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 divisorSubstrings(int num, int k) {
}

```

### Go Solution:

```

// Problem: Find the K-Beauty of a Number
// Difficulty: Easy
// Tags: array, string, 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 divisorSubstrings(num int, k int) int {
}

```

### Kotlin Solution:

```
class Solution {  
    fun divisorSubstrings(num: Int, k: Int): Int {  
  
    }  
}
```

### Swift Solution:

```
class Solution {  
    func divisorSubstrings(_ num: Int, _ k: Int) -> Int {  
  
    }  
}
```

### Rust Solution:

```
// Problem: Find the K-Beauty of a Number  
// Difficulty: Easy  
// Tags: array, string, 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 divisor_substrings(num: i32, k: i32) -> i32 {  
  
    }  
}
```

### Ruby Solution:

```
# @param {Integer} num  
# @param {Integer} k  
# @return {Integer}  
def divisor_substrings(num, k)  
  
end
```

### **PHP Solution:**

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

### **Dart Solution:**

```
class Solution {  
int divisorSubstrings(int num, int k) {  
  
}  
}
```

### **Scala Solution:**

```
object Solution {  
def divisorSubstrings(num: Int, k: Int): Int = {  
  
}  
}
```

### **Elixir Solution:**

```
defmodule Solution do  
@spec divisor_substrings(non :: integer, k :: integer) :: integer  
def divisor_substrings(non, k) do  
  
end  
end
```

### **Erlang Solution:**

```
-spec divisor_substrings(Num :: integer(), K :: integer()) -> integer().  
divisor_substrings(Num, K) ->  
. 
```

### Racket Solution:

```
(define/contract (divisor-substrings num k)  
(-> exact-integer? exact-integer? exact-integer?)  
) 
```