Problem 1891: Cutting Ribbons

Problem Information

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

Problem Description

You are given an integer array

ribbons

, where

ribbons[i]

represents the length of the

i

th

ribbon, and an integer

k

. You may cut any of the ribbons into any number of segments of

positive integer

lengths, or perform no cuts at all.

For example, if you have a ribbon of length

4

, you can:

Keep the ribbon of length

4

,

Cut it into one ribbon of length

3

and one ribbon of length

1

,

Cut it into two ribbons of length

2

,

Cut it into one ribbon of length

2

and two ribbons of length

1

, or

Cut it into four ribbons of length

1

.

Your task is to determine the

maximum

length of ribbon,

x

, that allows you to cut

at least

k

ribbons, each of length

x

. You can discard any leftover ribbon from the cuts. If it is

impossible

to cut

k

ribbons of the same length, return 0.

Example 1:

Input:

ribbons = [9,7,5], k = 3

Output:

5

Explanation:

- Cut the first ribbon to two ribbons, one of length 5 and one of length 4. - Cut the second ribbon to two ribbons, one of length 5 and one of length 2. - Keep the third ribbon as it is. Now you have 3 ribbons of length 5.

Example 2:

Input:

ribbons = [7,5,9], k = 4

Output:

4

Explanation:

- Cut the first ribbon to two ribbons, one of length 4 and one of length 3. - Cut the second ribbon to two ribbons, one of length 4 and one of length 1. - Cut the third ribbon to three ribbons, two of length 4 and one of length 1. Now you have 4 ribbons of length 4.

Example 3:

Input:

ribbons = [5,7,9], k = 22

Output:

0

Explanation:

You cannot obtain k ribbons of the same positive integer length.

Constraints:

1 <= ribbons.length <= 10

5

1 <= ribbons[i] <= 10

5

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxLength(vector<int>& ribbons, int k) {

}
};
```

Java:

```
class Solution {
public int maxLength(int[] ribbons, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maxLength(self, ribbons, k):
```

```
"""
:type ribbons: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxLength(ribbons: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxLength(int[] ribbons, int k) {

}
}
```

C:

```
int maxLength(int* ribbons, int ribbonsSize, int k) {

}
```

Go:

```
func maxLength(ribbons []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxLength(ribbons: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} ribbons
# @param {Integer} k
# @return {Integer}
def max_length(ribbons, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $ribbons
* @param Integer $k
* @return Integer
*/
function maxLength($ribbons, $k) {
}
}
```

Dart:

```
class Solution {
int maxLength(List<int> ribbons, int k) {

}
}
```

Scala:

```
object Solution {
def maxLength(ribbons: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_length(ribbons :: [integer], k :: integer) :: integer
def max_length(ribbons, k) do

end
end
```

Erlang:

```
-spec max_length(Ribbons :: [integer()], K :: integer()) -> integer().
max_length(Ribbons, K) ->
.
```

Racket:

```
(define/contract (max-length ribbons k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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 maxLength(vector<int>& ribbons, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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 maxLength(int[] ribbons, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Cutting Ribbons
```

```
Difficulty: Medium
Tags: array, search

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 maxLength(self, ribbons: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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[]} ribbons
* @param {number} k
* @return {number}
*/
var maxLength = function(ribbons, k) {
};
```

TypeScript Solution:

```
/**
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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 maxLength(ribbons: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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 MaxLength(int[] ribbons, int k) {

}
}
```

C Solution:

```
/*
* Problem: Cutting Ribbons
* Difficulty: Medium
* Tags: array, search
*
* 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 maxLength(int* ribbons, int ribbonsSize, int k) {

}
```

Go Solution:

```
// Problem: Cutting Ribbons
// Difficulty: Medium
// Tags: array, search
//
// 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 maxLength(ribbons []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxLength(ribbons: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Cutting Ribbons
// Difficulty: Medium
// Tags: array, search
//
// 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 max_length(ribbons: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} ribbons
# @param {Integer} k
# @return {Integer}
def max_length(ribbons, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $ribbons
* @param Integer $k
* @return Integer
*/
function maxLength($ribbons, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxLength(List<int> ribbons, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxLength(ribbons: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_length(ribbons :: [integer], k :: integer) :: integer
def max_length(ribbons, k) do

end
end
```

Erlang Solution:

```
-spec max_length(Ribbons :: [integer()], K :: integer()) -> integer().
max_length(Ribbons, K) ->
.
```

Racket Solution:

```
(define/contract (max-length ribbons k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```