Problem 3669: Balanced K-Factor Decomposition

Problem Information

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

Problem Description

Given two integers

n

and

k

, split the number

n

into exactly

k

positive integers such that the

product

of these integers is equal to

n

.

Return

any

one

split in which the

maximum

difference between any two numbers is

minimized

. You may return the result in

any order

.

Example 1:

Input:

n = 100, k = 2

Output:

[10,10]

Explanation:

The split

[10, 10]

yields

10 * 10 = 100

and a max-min difference of 0, which is minimal.

Example 2:

Input:

n = 44, k = 3

Output:

[2,2,11]

Explanation:

Split

[1, 1, 44]

yields a difference of 43

Split

[1, 2, 22]

yields a difference of 21

Split

[1, 4, 11]

yields a difference of 10

Split

[2, 2, 11]

yields a difference of 9

Therefore,

[2, 2, 11]

is the optimal split with the smallest difference 9.

Constraints:

4 <= n <= 10

5

2 <= k <= 5

k

is strictly less than the total number of positive divisors of

n

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> minDifference(int n, int k) {

}
};
```

Java:

```
class Solution {
public int[] minDifference(int n, int k) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minDifference(n: number, k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MinDifference(int n, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minDifference(int n, int k, int* returnSize) {

}
```

Go:

```
func minDifference(n int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun minDifference(n: Int, k: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
```

```
# @return {Integer[]}
def min_difference(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[]
*/
function minDifference($n, $k) {

}
}
```

Dart:

```
class Solution {
List<int> minDifference(int n, int k) {

}
}
```

Scala:

```
object Solution {
def minDifference(n: Int, k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_difference(n :: integer, k :: integer) :: [integer]
def min_difference(n, k) do

end
```

```
end
```

Erlang:

```
-spec min_difference(N :: integer(), K :: integer()) -> [integer()].
min_difference(N, K) ->
.
```

Racket:

```
(define/contract (min-difference n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
* 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:
vector<int> minDifference(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
```

```
* 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[] minDifference(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Balanced K-Factor Decomposition
Difficulty: Medium
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 minDifference(self, n: int, k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
* 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} n
* @param {number} k
* @return {number[]}
*/
var minDifference = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
* 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 minDifference(n: number, k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
* 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[] MinDifference(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Balanced K-Factor Decomposition
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minDifference(int n, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Balanced K-Factor Decomposition
// Difficulty: Medium
// 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 minDifference(n int, k int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minDifference(n: Int, k: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Balanced K-Factor Decomposition
// Difficulty: Medium
// 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 min_difference(n: i32, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[]}
def min_difference(n, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[]
*/
function minDifference($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minDifference(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minDifference(n: Int, k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_difference(n :: integer, k :: integer) :: [integer]
def min_difference(n, k) do

end
end
```

Erlang Solution:

```
-spec min_difference(N :: integer(), K :: integer()) -> [integer()].
min_difference(N, K) ->
.
```

Racket Solution:

```
(define/contract (min-difference n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```