Problem 667: Beautiful Arrangement II

Problem Information

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

Problem Description

Given two integers

n

and

k

, construct a list

answer

that contains

n

different positive integers ranging from

1

to

n

and obeys the following requirement:

Suppose this list is

answer = [a

1

, a

2

, a

3

, ... , a

n

]

, then the list

[|a

1

- a

2

|, |a

2

- a

3

|, |a

3

- a

4

|, ... , |a

n-1

- a

n

|]

has exactly

k

distinct integers.

Return

the list

answer

. If there multiple valid answers, return

any of them

.

Example 1:

Input:

n = 3, k = 1

Output:

[1,2,3] Explanation: The [1,2,3] has three different positive integers ranging from 1 to 3, and the [1,1] has exactly 1 distinct integer: 1

Example 2:

Input:

n = 3, k = 2

Output:

[1,3,2] Explanation: The [1,3,2] has three different positive integers ranging from 1 to 3, and the [2,1] has exactly 2 distinct integers: 1 and 2.

Constraints:

1 <= k < n <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
```

```
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (construct-array n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Beautiful Arrangement II
* Difficulty: Medium
* Tags: array, math
*
* 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:
vector<int> constructArray(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Beautiful Arrangement II
```

```
* Difficulty: Medium
* Tags: array, math
*
* 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[] constructArray(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Beautiful Arrangement II
Difficulty: Medium
Tags: array, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Beautiful Arrangement II
* Difficulty: Medium
* Tags: array, math
*
* 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} n
* @param {number} k
* @return {number[]}
*/
var constructArray = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Beautiful Arrangement II
* Difficulty: Medium
* Tags: array, math
*
* 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 constructArray(n: number, k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Beautiful Arrangement II
* Difficulty: Medium
* Tags: array, math
```

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

}
}
```

C Solution:

```
/*
* Problem: Beautiful Arrangement II
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Beautiful Arrangement II
// Difficulty: Medium
// Tags: array, math
//
// 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 constructArray(n int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Beautiful Arrangement II
// Difficulty: Medium
// Tags: array, math
//
// 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 construct_array(n: i32, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[]}
```

```
def construct_array(n, k)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (construct-array n k)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```