Problem 2022: Convert 1D Array Into 2D Array

Problem Information

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

Problem Description

You are given a

0-indexed

1-dimensional (1D) integer array

original

, and two integers,

m

and

n

. You are tasked with creating a 2-dimensional (2D) array with

m

rows and

n

columns using

all

the elements from

original

.

The elements from indices

0

to

n - 1

(

inclusive

) of

original

should form the first row of the constructed 2D array, the elements from indices

n

to

2 * n - 1

(

inclusive

) should form the second row of the constructed 2D array, and so on.

Return

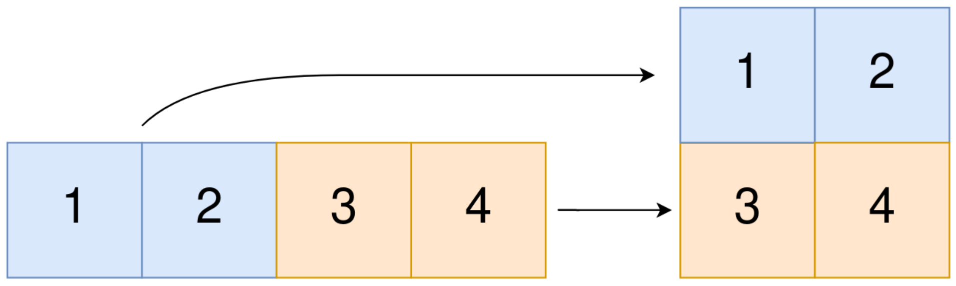
an

m x n

2D array constructed according to the above procedure, or an empty 2D array if it is impossible

.

Example 1:



Input:

original = [1,2,3,4], m = 2, n = 2

Output:

[[1,2],[3,4]]

Explanation:

The constructed 2D array should contain 2 rows and 2 columns. The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array. The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

Example 2:

Input:

original = [1,2,3], m = 1, n = 3

Output:

[[1,2,3]]

Explanation:

The constructed 2D array should contain 1 row and 3 columns. Put all three elements in original into the first row of the constructed 2D array.

Example 3:

Input:

original = [1,2], m = 1, n = 1

Output:

[]

Explanation:

There are 2 elements in original. It is impossible to fit 2 elements in a 1x1 2D array, so return an empty 2D array.

Constraints:

1 <= original.length <= 5 * 10

4

1 <= original[i] <= 10

5

1 <= m, n <= 4 * 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> construct2DArray(vector<int>& original, int m, int n) {

}
};
```

Java:

```
class Solution {
public int[][] construct2DArray(int[] original, int m, int n) {

}
}
```

Python3:

```
class Solution:
def construct2DArray(self, original: List[int], m: int, n: int) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def construct2DArray(self, original, m, n):
"""
:type original: List[int]
:type m: int
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[]} original
* @param {number} m
* @param {number} n
* @return {number[][]}
```

```
*/
var construct2DArray = function(original, m, n) {

};
```

TypeScript:

```
function construct2DArray(original: number[], m: number, n: number):
number[][] {

};
```

C#:

```
public class Solution {
public int[][] Construct2DArray(int[] original, int m, int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** construct2DArray(int* original, int originalSize, int m, int n, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func construct2DArray(original []int, m int, n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun construct2DArray(original: IntArray, m: Int, n: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func construct2DArray(_ original: [Int], _ m: Int, _ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn construct2_d_array(original: Vec<i32>, m: i32, n: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[]} original
# @param {Integer} m
# @param {Integer} n
# @return {Integer[][]}
def construct2_d_array(original, m, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $original
* @param Integer $m
* @param Integer $n
* @return Integer[][]
*/
function construct2DArray($original, $m, $n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> construct2DArray(List<int> original, int m, int n) {

}
}
```

Scala:

```
object Solution {
def construct2DArray(original: Array[Int], m: Int, n: Int): Array[Array[Int]]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec construct2_d_array(original :: [integer], m :: integer, n :: integer)
:: [[integer]]
def construct2_d_array(original, m, n) do

end
end
```

Erlang:

```
-spec construct2_d_array(Original :: [integer()], M :: integer(), N ::
integer()) -> [[integer()]].
construct2_d_array(Original, M, N) ->
.
```

Racket:

```
(define/contract (construct2-d-array original m n)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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<vector<int>> construct2DArray(vector<int>& original, int m, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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[][] construct2DArray(int[] original, int m, int n) {
}
}
```

Python3 Solution:

```
"""
Problem: Convert 1D Array Into 2D Array
Difficulty: Easy
Tags: array

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 construct2DArray(self, original: List[int], m: int, n: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def construct2DArray(self, original, m, n):
"""
:type original: List[int]
:type m: int
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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[]} original
* @param {number} m
* @param {number} n
* @return {number[][]}
*/
var construct2DArray = function(original, m, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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 construct2DArray(original: number[], m: number, n: number):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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[][] Construct2DArray(int[] original, int m, int n) {

}
}
```

C Solution:

```
/*
* Problem: Convert 1D Array Into 2D Array
* Difficulty: Easy
* Tags: array
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** construct2DArray(int* original, int originalSize, int m, int n, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Convert 1D Array Into 2D Array
// Difficulty: Easy
// Tags: array
//
// 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 construct2DArray(original []int, m int, n int) [][]int {
}
```

Kotlin Solution:

```
class Solution {
fun construct2DArray(original: IntArray, m: Int, n: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func construct2DArray(_ original: [Int], _ m: Int, _ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Convert 1D Array Into 2D Array
// Difficulty: Easy
// Tags: array
//
// 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 construct2_d_array(original: Vec<i32>, m: i32, n: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} original
# @param {Integer} m
# @param {Integer} n
# @return {Integer[][]}
def construct2_d_array(original, m, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $original
* @param Integer $m
* @param Integer $n
* @return Integer[][]
*/
function construct2DArray($original, $m, $n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> construct2DArray(List<int> original, int m, int n) {

}
}
```

Scala Solution:

```
object Solution {
def construct2DArray(original: Array[Int], m: Int, n: Int): Array[Array[Int]]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec construct2_d_array(original :: [integer], m :: integer, n :: integer)
```

```
:: [[integer]]
def construct2_d_array(original, m, n) do

end
end
```

Erlang Solution:

```
-spec construct2_d_array(Original :: [integer()], M :: integer(), N ::
integer()) -> [[integer()]].
construct2_d_array(Original, M, N) ->
.
```

Racket Solution:

```
(define/contract (construct2-d-array original m n)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof (listof
exact-integer?)))
)
```