# Problem 1640: Check Array Formation Through Concatenation

## Problem Information

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

## Problem Description

You are given an array of

distinct

integers

arr

and an array of integer arrays

pieces

, where the integers in

pieces

are

distinct

. Your goal is to form

arr

by concatenating the arrays in

pieces

in any order

. However, you are

not

allowed to reorder the integers in each array

pieces[i]

.

Return

true

if it is possible

to form the array

arr

from

pieces

. Otherwise, return

false

.

Example 1:

Input:

arr = [15,88], pieces = [[88],[15]]

Output:

true

Explanation:

Concatenate [15] then [88]

Example 2:

Input:

arr = [49,18,16], pieces = [[16,18,49]]

Output:

false

Explanation:

Even though the numbers match, we cannot reorder pieces[0].

Example 3:

Input:

arr = [91,4,64,78], pieces = [[78],[4,64],[91]]

Output:

true

Explanation:

Concatenate [91] then [4,64] then [78]

Constraints:

1 <= pieces.length <= arr.length <= 100

sum(pieces[i].length) == arr.length

1 <= pieces[i].length <= arr.length

1 <= arr[i], pieces[i][j] <= 100

The integers in

arr

are

distinct

.

The integers in

pieces

are

distinct

(i.e., If we flatten pieces in a 1D array, all the integers in this array are distinct).

## Code Snippets

**C++:**

```
class Solution {
public:
bool canFormArray(vector<int>& arr, vector<vector<int>>& pieces) {

}
};
```

**Java:**

```
class Solution {
public boolean canFormArray(int[] arr, int[][] pieces) {

}
}
```

**Python3:**

```
class Solution:
def canFormArray(self, arr: List[int], pieces: List[List[int]]) -> bool:
```

**Python:**

```
class Solution(object):
def canFormArray(self, arr, pieces):
"""
:type arr: List[int]
:type pieces: List[List[int]]
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number[]} arr
* @param {number[][]} pieces
* @return {boolean}
*/
var canFormArray = function(arr, pieces) {

};
```

**TypeScript:**

```
function canFormArray(arr: number[], pieces: number[][]): boolean {

};
```

**C#:**

```
public class Solution {
public bool CanFormArray(int[] arr, int[][] pieces) {

}
}
```

**C:**

```
bool canFormArray(int* arr, int arrSize, int** pieces, int piecesSize, int*
piecesColSize) {

}
```

**Go:**

```
func canFormArray(arr []int, pieces [][]int) bool {

}
```

**Kotlin:**

```
class Solution {
fun canFormArray(arr: IntArray, pieces: Array<IntArray>): Boolean {

}
}
```

**Swift:**

```
class Solution {
func canFormArray(_ arr: [Int], _ pieces: [[Int]]) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn can_form_array(arr: Vec<i32>, pieces: Vec<Vec<i32>>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[]} arr
# @param {Integer[][]} pieces
# @return {Boolean}
def can_form_array(arr, pieces)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer[][] $pieces
* @return Boolean
*/
function canFormArray($arr, $pieces) {

}
}
```

**Dart:**

```
class Solution {
bool canFormArray(List<int> arr, List<List<int>> pieces) {

}
}
```

**Scala:**

```
object Solution {
def canFormArray(arr: Array[Int], pieces: Array[Array[Int]]): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec can_form_array(arr :: [integer], pieces :: [[integer]]) :: boolean
```

```
def can_form_array(arr, pieces) do

end
end
```

**Erlang:**

```
-spec can_form_array(Arr :: [integer()], Pieces :: [[integer()]]) ->
boolean().
can_form_array(Arr, Pieces) ->
.
```

**Racket:**

```
(define/contract (can-form-array arr pieces)
(-> (listof exact-integer?) (listof (listof exact-integer?)) boolean?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool canFormArray(vector<int>& arr, vector<vector<int>>& pieces) {

}
};
```

**Java Solution:**

```
/**
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean canFormArray(int[] arr, int[][] pieces) {

}
}
```

### Python3 Solution:

```
"""
Problem: Check Array Formation Through Concatenation
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def canFormArray(self, arr: List[int], pieces: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def canFormArray(self, arr, pieces):
"""
:type arr: List[int]
:type pieces: List[List[int]]
:rtype: bool
"""
```

### JavaScript Solution:

```
/**
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} arr
* @param {number[][]} pieces
* @return {boolean}
*/
var canFormArray = function(arr, pieces) {

};
```

### TypeScript Solution:

```
/**
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function canFormArray(arr: number[], pieces: number[][]): boolean {

};
```

### C# Solution:

```
/*
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
```

```
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool CanFormArray(int[] arr, int[][] pieces) {

}
}
```

### C Solution:

```
/*
* Problem: Check Array Formation Through Concatenation
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool canFormArray(int* arr, int arrSize, int** pieces, int piecesSize, int*
piecesColSize) {

}
```

### Go Solution:

```
// Problem: Check Array Formation Through Concatenation
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func canFormArray(arr []int, pieces [][]int) bool {
```

```
}
```

### Kotlin Solution:

```
class Solution {
fun canFormArray(arr: IntArray, pieces: Array<IntArray>): Boolean {

}
}
```

### Swift Solution:

```
class Solution {
func canFormArray(_ arr: [Int], _ pieces: [[Int]]) -> Bool {

}
}
```

### Rust Solution:

```
// Problem: Check Array Formation Through Concatenation
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn can_form_array(arr: Vec<i32>, pieces: Vec<Vec<i32>>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer[][]} pieces
# @return {Boolean}
def can_form_array(arr, pieces)
```

```
end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer[][] $pieces
* @return Boolean
*/
function canFormArray($arr, $pieces) {

}
}
```

**Dart Solution:**

```
class Solution {
bool canFormArray(List<int> arr, List<List<int>> pieces) {

}
}
```

**Scala Solution:**

```
object Solution {
def canFormArray(arr: Array[Int], pieces: Array[Array[Int]]): Boolean = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec can_form_array(arr :: [integer], pieces :: [[integer]]) :: boolean
def can_form_array(arr, pieces) do

end
end
```

### Erlang Solution:

```
-spec can_form_array(Arr :: [integer()], Pieces :: [[integer()]]) ->
boolean().
can_form_array(Arr, Pieces) ->
.
```

### Racket Solution:

```
(define/contract (can-form-array arr pieces)
(-> (listof exact-integer?) (listof (listof exact-integer?)) boolean?)
)
```