Problem 1122: Relative Sort Array

Problem Information

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

Problem Description

Given two arrays

arr1

and

arr2

, the elements of

arr2

are distinct, and all elements in

arr2

are also in

arr1

.

Sort the elements of

arr1

such that the relative ordering of items in

arr1

are the same as in

arr2

. Elements that do not appear in

arr2

should be placed at the end of

arr1

in

ascending

order.

Example 1:

Input:

arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

Output:

[2,2,2,1,4,3,3,9,6,7,19]

Example 2:

Input:

arr1 = [28,6,22,8,44,17], arr2 = [22,28,8,6]

Output:

[22,28,8,6,17,44]

Constraints:

1 <= arr1.length, arr2.length <= 1000

0 <= arr1[i], arr2[i] <= 1000

All the elements of

arr2

are

distinct

.

Each

arr2[i]

is in

arr1

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java:

```
class Solution {
public int[] relativeSortArray(int[] arr1, int[] arr2) {

}
}
```

Python3:

```
class Solution:
def relativeSortArray(self, arr1: List[int], arr2: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def relativeSortArray(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} arr1
* @param {number[]} arr2
* @return {number[]}
*/
var relativeSortArray = function(arr1, arr2) {

};
```

TypeScript:

```
function relativeSortArray(arr1: number[], arr2: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] RelativeSortArray(int[] arr1, int[] arr2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* relativeSortArray(int* arr1, int arr1Size, int* arr2, int arr2Size, int*
returnSize) {

}
```

Go:

```
func relativeSortArray(arr1 []int, arr2 []int) []int {

}
```

Kotlin:

```
class Solution {
fun relativeSortArray(arr1: IntArray, arr2: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func relativeSortArray(_ arr1: [Int], _ arr2: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn relative_sort_array(arr1: Vec<i32>, arr2: Vec<i32>) -> Vec<i32> {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer[]}
def relative_sort_array(arr1, arr2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer[]
*/
function relativeSortArray($arr1, $arr2) {

}
}
```

Dart:

```
class Solution {
List<int> relativeSortArray(List<int> arr1, List<int> arr2) {

}
}
```

Scala:

```
object Solution {
def relativeSortArray(arr1: Array[Int], arr2: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec relative_sort_array(arr1 :: [integer], arr2 :: [integer]) :: [integer]
def relative_sort_array(arr1, arr2) do

end
end
```

Erlang:

```
-spec relative_sort_array(Arr1 :: [integer()], Arr2 :: [integer()]) ->
[integer()].
relative_sort_array(Arr1, Arr2) ->
.
```

Racket:

```
(define/contract (relative-sort-array arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Relative Sort Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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:
vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java Solution:

```
/**
* Problem: Relative Sort Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 int[] relativeSortArray(int[] arr1, int[] arr2) {

}
}
```

Python3 Solution:

```
"""
Problem: Relative Sort Array
Difficulty: Easy
Tags: array, hash, sort

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 relativeSortArray(self, arr1: List[int], arr2: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def relativeSortArray(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
```

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Relative Sort Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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[]} arr1
* @param {number[]} arr2
* @return {number[]}
*/
var relativeSortArray = function(arr1, arr2) {

};
```

TypeScript Solution:

```
/**
* Problem: Relative Sort Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 relativeSortArray(arr1: number[], arr2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Relative Sort Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 int[] RelativeSortArray(int[] arr1, int[] arr2) {

}
}
```

C Solution:

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

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* relativeSortArray(int* arr1, int arr1Size, int* arr2, int arr2Size, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Relative Sort Array
// Difficulty: Easy
// Tags: array, hash, sort
//
```

```
// 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 relativeSortArray(arr1 []int, arr2 []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun relativeSortArray(arr1: IntArray, arr2: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func relativeSortArray(_ arr1: [Int], _ arr2: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Relative Sort Array
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 relative_sort_array(arr1: Vec<i32>, arr2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer[]}
def relative_sort_array(arr1, arr2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer[]
*/
function relativeSortArray($arr1, $arr2) {

}
}
```

Dart Solution:

```
class Solution {
List<int> relativeSortArray(List<int> arr1, List<int> arr2) {

}
}
```

Scala Solution:

```
object Solution {
def relativeSortArray(arr1: Array[Int], arr2: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec relative_sort_array(arr1 :: [integer], arr2 :: [integer]) :: [integer]
def relative_sort_array(arr1, arr2) do
```

```
end
end
```

Erlang Solution:

```
-spec relative_sort_array(Arr1 :: [integer()], Arr2 :: [integer()]) ->
[integer()].
relative_sort_array(Arr1, Arr2) ->
.
```

Racket Solution:

```
(define/contract (relative-sort-array arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```