Problem 1054: Distant Barcodes

Problem Information

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

Problem Description

In a warehouse, there is a row of barcodes, where the

i

th

barcode is

barcodes[i]

.

Rearrange the barcodes so that no two adjacent barcodes are equal. You may return any answer, and it is guaranteed an answer exists.

Example 1:

Input:

barcodes = [1,1,1,2,2,2]

Output:

[2,1,2,1,2,1]

Example 2:

Input:

barcodes = [1,1,1,1,2,2,3,3]

Output:

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

Constraints:

1 <= barcodes.length <= 10000

1 <= barcodes[i] <= 10000

Code Snippets

C++:

```
class Solution {
public:
vector<int> rearrangeBarcodes(vector<int>& barcodes) {

}
};
```

Java:

```
class Solution {
public int[] rearrangeBarcodes(int[] barcodes) {

}
}
```

Python3:

```
class Solution:
def rearrangeBarcodes(self, barcodes: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def rearrangeBarcodes(self, barcodes):
"""
:type barcodes: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} barcodes
* @return {number[]}
*/
var rearrangeBarcodes = function(barcodes) {

};
```

TypeScript:

```
function rearrangeBarcodes(barcodes: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] RearrangeBarcodes(int[] barcodes) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* rearrangeBarcodes(int* barcodes, int barcodesSize, int* returnSize) {

}
```

Go:

```
func rearrangeBarcodes(barcodes []int) []int {

}
```

Kotlin:

```
class Solution {
fun rearrangeBarcodes(barcodes: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func rearrangeBarcodes(_ barcodes: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn rearrange_barcodes(barcodes: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} barcodes
# @return {Integer[]}
def rearrange_barcodes(barcodes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $barcodes
* @return Integer[]
*/
function rearrangeBarcodes($barcodes) {

}
}
```

Dart:

```
class Solution {
List<int> rearrangeBarcodes(List<int> barcodes) {

}
}
```

Scala:

```
object Solution {
def rearrangeBarcodes(barcodes: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec rearrange_barcodes(barcodes :: [integer]) :: [integer]
def rearrange_barcodes(barcodes) do

end
end
```

Erlang:

```
-spec rearrange_barcodes(Barcodes :: [integer()]) -> [integer()].
rearrange_barcodes(Barcodes) ->
.
```

Racket:

```
(define/contract (rearrange-barcodes barcodes)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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> rearrangeBarcodes(vector<int>& barcodes) {

}
};
```

Java Solution:

```
/**
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[] rearrangeBarcodes(int[] barcodes) {

}
}
```

Python3 Solution:

```
"""
Problem: Distant Barcodes
Difficulty: Medium
Tags: array, greedy, hash, sort, queue, heap

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 rearrangeBarcodes(self, barcodes: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rearrangeBarcodes(self, barcodes):
"""
:type barcodes: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[]} barcodes
* @return {number[]}
*/
var rearrangeBarcodes = function(barcodes) {
};
```

TypeScript Solution:

```
/**
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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 rearrangeBarcodes(barcodes: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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[] RearrangeBarcodes(int[] barcodes) {

}
}
```

C Solution:

```
/*
* Problem: Distant Barcodes
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* 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* rearrangeBarcodes(int* barcodes, int barcodesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Distant Barcodes
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// 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 rearrangeBarcodes(barcodes []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun rearrangeBarcodes(barcodes: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func rearrangeBarcodes(_ barcodes: [Int]) -> [Int] {
}
}
```

Rust Solution:

```
// Problem: Distant Barcodes
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// 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 rearrange_barcodes(barcodes: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} barcodes
# @return {Integer[]}
def rearrange_barcodes(barcodes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $barcodes
* @return Integer[]
*/
function rearrangeBarcodes($barcodes) {

}
}
```

Dart Solution:

```
class Solution {
List<int> rearrangeBarcodes(List<int> barcodes) {

}
}
```

Scala Solution:

```
object Solution {
def rearrangeBarcodes(barcodes: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec rearrange_barcodes(barcodes :: [integer]) :: [integer]
def rearrange_barcodes(barcodes) do

end
end
```

Erlang Solution:

```
-spec rearrange_barcodes(Barcodes :: [integer()]) -> [integer()].
rearrange_barcodes(Barcodes) ->
.
```

Racket Solution:

```
(define/contract (rearrange-barcodes barcodes)
(-> (listof exact-integer?) (listof exact-integer?))
)
```