Problem 2605: Form Smallest Number From Two Digit Arrays

Problem Information

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

Problem Description

Given two arrays of

unique

digits

nums1

and

nums2

, return

the

smallest

number that contains

at least

one digit from each array

.

Example 1:

Input:

nums1 = [4,1,3], nums2 = [5,7]

Output:

15

Explanation:

The number 15 contains the digit 1 from nums1 and the digit 5 from nums2. It can be proven that 15 is the smallest number we can have.

Example 2:

Input:

nums1 = [3,5,2,6], nums2 = [3,1,7]

Output:

3

Explanation:

The number 3 contains the digit 3 which exists in both arrays.

Constraints:

1 <= nums1.length, nums2.length <= 9

1 <= nums1[i], nums2[i] <= 9

All digits in each array are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int minNumber(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int minNumber(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def minNumber(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def minNumber(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
```

```
 * @return {number}
 */
var minNumber = function(nums1, nums2) {

};
```

TypeScript:

```
function minNumber(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int MinNumber(int[] nums1, int[] nums2) {

}
}
```

C:

```
int minNumber(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func minNumber(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun minNumber(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minNumber(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_number(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_number(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minNumber($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int minNumber(List<int> nums1, List<int> nums2) {

}
```

```
}
```

Scala:

```
object Solution {
def minNumber(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_number(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_number(nums1, nums2) do

end
end
```

Erlang:

```
-spec min_number(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
min_number(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (min-number nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Form Smallest Number From Two Digit Arrays
* 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:
int minNumber(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Form Smallest Number From Two Digit Arrays
* 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 int minNumber(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Form Smallest Number From Two Digit Arrays
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 minNumber(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minNumber(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Form Smallest Number From Two Digit Arrays
* 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minNumber = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Form Smallest Number From Two Digit Arrays
```

```
* 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 minNumber(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Form Smallest Number From Two Digit Arrays
* 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 int MinNumber(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Form Smallest Number From Two Digit Arrays
* 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
*/
```

```
int minNumber(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Form Smallest Number From Two Digit Arrays
// 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 minNumber(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minNumber(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minNumber(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Form Smallest Number From Two Digit Arrays
// 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 min_number(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_number(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minNumber($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int minNumber(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def minNumber(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_number(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_number(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec min_number(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
min_number(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (min-number nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```