Problem 2513: Minimize the Maximum of Two Arrays

Problem Information

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

Problem Description

We have two arrays

arr1

and

arr2

which are initially empty. You need to add positive integers to them such that they satisfy all the following conditions:

arr1

contains

uniqueCnt1

distinct

positive integers, each of which is

not divisible

by

divisor1

.

arr2

contains

uniqueCnt2

distinct

positive integers, each of which is

not divisible

by

divisor2

.

No

integer is present in both

arr1

and

arr2

.

Given

divisor1

,

divisor2

,

uniqueCnt1

, and

uniqueCnt2

, return

the

minimum possible maximum

integer that can be present in either array

.

Example 1:

Input:

divisor1 = 2, divisor2 = 7, uniqueCnt1 = 1, uniqueCnt2 = 3

Output:

4

Explanation:

We can distribute the first 4 natural numbers into arr1 and arr2. arr1 = [1] and arr2 = [2,3,4]. We can see that both arrays satisfy all the conditions. Since the maximum value is 4, we return it.

Example 2:

Input:

divisor1 = 3, divisor2 = 5, uniqueCnt1 = 2, uniqueCnt2 = 1

Output:

3

Explanation:

Here arr1 = [1,2], and arr2 = [3] satisfy all conditions. Since the maximum value is 3, we return it.

Example 3:

Input:

divisor1 = 2, divisor2 = 4, uniqueCnt1 = 8, uniqueCnt2 = 2

Output:

15

Explanation:

Here, the final possible arrays can be arr1 = [1,3,5,7,9,11,13,15], and arr2 = [2,6]. It can be shown that it is not possible to obtain a lower maximum satisfying all conditions.

Constraints:

2 <= divisor1, divisor2 <= 10

5

1 <= uniqueCnt1, uniqueCnt2 < 10

9

2 <= uniqueCnt1 + uniqueCnt2 <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
};
```

Java:

```
class Solution {
public int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int
uniqueCnt2) {

}
}
```

Python3:

```
class Solution:
def minimizeSet(self, divisor1: int, divisor2: int, uniqueCnt1: int,
uniqueCnt2: int) -> int:
```

Python:

```
class Solution(object):
def minimizeSet(self, divisor1, divisor2, uniqueCnt1, uniqueCnt2):
"""
:type divisor1: int
:type divisor2: int
:type uniqueCnt1: int
:type uniqueCnt2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} divisor1
* @param {number} divisor2
* @param {number} uniqueCnt1
* @param {number} uniqueCnt2
* @return {number}
*/
var minimizeSet = function(divisor1, divisor2, uniqueCnt1, uniqueCnt2) {

};
```

TypeScript:

```
function minimizeSet(divisor1: number, divisor2: number, uniqueCnt1: number,
uniqueCnt2: number): number {

};
```

C#:

```
public class Solution {
public int MinimizeSet(int divisor1, int divisor2, int uniqueCnt1, int
uniqueCnt2) {

}
}
```

C:

```
int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
```

Go:

```
func minimizeSet(divisor1 int, divisor2 int, uniqueCnt1 int, uniqueCnt2 int)
int {

}
```

Kotlin:

```
class Solution {
fun minimizeSet(divisor1: Int, divisor2: Int, uniqueCnt1: Int, uniqueCnt2:
Int): Int {

}
}
```

Swift:

```
class Solution {
func minimizeSet(_ divisor1: Int, _ divisor2: Int, _ uniqueCnt1: Int, _
uniqueCnt2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimize_set(divisor1: i32, divisor2: i32, unique_cnt1: i32,
unique_cnt2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} divisor1
# @param {Integer} divisor2
# @param {Integer} unique_cnt1
# @param {Integer} unique_cnt2
# @return {Integer}
def minimize_set(divisor1, divisor2, unique_cnt1, unique_cnt2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $divisor1
* @param Integer $divisor2
* @param Integer $uniqueCnt1
* @param Integer $uniqueCnt2
* @return Integer
*/
function minimizeSet($divisor1, $divisor2, $uniqueCnt1, $uniqueCnt2) {

}
}
```

Dart:

```
class Solution {
int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
}
```

Scala:

```
object Solution {
def minimizeSet(divisor1: Int, divisor2: Int, uniqueCnt1: Int, uniqueCnt2:
Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimize_set(divisor1 :: integer, divisor2 :: integer, unique_cnt1 ::
integer, unique_cnt2 :: integer) :: integer
def minimize_set(divisor1, divisor2, unique_cnt1, unique_cnt2) do

end
end
```

Erlang:

```
-spec minimize_set(Divisor1 :: integer(), Divisor2 :: integer(), UniqueCnt1
:: integer(), UniqueCnt2 :: integer()) -> integer().
minimize_set(Divisor1, Divisor2, UniqueCnt1, UniqueCnt2) ->
.
```

Racket:

```
(define/contract (minimize-set divisor1 divisor2 uniqueCnt1 uniqueCnt2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
*
* 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 minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
*
* 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 minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int
uniqueCnt2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize the Maximum of Two Arrays
Difficulty: Medium
Tags: array, math, search

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 minimizeSet(self, divisor1: int, divisor2: int, uniqueCnt1: int,
uniqueCnt2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimizeSet(self, divisor1, divisor2, uniqueCnt1, uniqueCnt2):
"""
:type divisor1: int
:type divisor2: int
:type uniqueCnt1: int
:type uniqueCnt2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
*
* 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} divisor1
* @param {number} divisor2
* @param {number} uniqueCnt1
* @param {number} uniqueCnt2
* @return {number}
*/
var minimizeSet = function(divisor1, divisor2, uniqueCnt1, uniqueCnt2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
*
* 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 minimizeSet(divisor1: number, divisor2: number, uniqueCnt1: number,
uniqueCnt2: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
```

```
*
* 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 MinimizeSet(int divisor1, int divisor2, int uniqueCnt1, int
uniqueCnt2) {

}
}
```

C Solution:

```
/*
* Problem: Minimize the Maximum of Two Arrays
* Difficulty: Medium
* Tags: array, math, search
*
* 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
*/

int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
```

Go Solution:

```
// Problem: Minimize the Maximum of Two Arrays
// Difficulty: Medium
// Tags: array, math, search
//
// 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 minimizeSet(divisor1 int, divisor2 int, uniqueCnt1 int, uniqueCnt2 int)
int {
}
```

Kotlin Solution:

```
class Solution {
fun minimizeSet(divisor1: Int, divisor2: Int, uniqueCnt1: Int, uniqueCnt2:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimizeSet(_ divisor1: Int, _ divisor2: Int, _ uniqueCnt1: Int, _
uniqueCnt2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize the Maximum of Two Arrays
// Difficulty: Medium
// Tags: array, math, search
//
// 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 minimize_set(divisor1: i32, divisor2: i32, unique_cnt1: i32,
unique_cnt2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} divisor1
# @param {Integer} divisor2
# @param {Integer} unique_cnt1
# @param {Integer} unique_cnt2
# @return {Integer}
def minimize_set(divisor1, divisor2, unique_cnt1, unique_cnt2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $divisor1
* @param Integer $divisor2
* @param Integer $uniqueCnt1
* @param Integer $uniqueCnt2
* @return Integer
*/
function minimizeSet($divisor1, $divisor2, $uniqueCnt1, $uniqueCnt2) {

}
}
```

Dart Solution:

```
class Solution {
int minimizeSet(int divisor1, int divisor2, int uniqueCnt1, int uniqueCnt2) {

}
}
```

Scala Solution:

```
object Solution {
def minimizeSet(divisor1: Int, divisor2: Int, uniqueCnt1: Int, uniqueCnt2:
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimize_set(divisor1 :: integer, divisor2 :: integer, unique_cnt1 ::
integer, unique_cnt2 :: integer) :: integer
def minimize_set(divisor1, divisor2, unique_cnt1, unique_cnt2) do

end
end
```

Erlang Solution:

```
-spec minimize_set(Divisor1 :: integer(), Divisor2 :: integer(), UniqueCnt1
:: integer(), UniqueCnt2 :: integer()) -> integer().
minimize_set(Divisor1, Divisor2, UniqueCnt1, UniqueCnt2) ->
.
```

Racket Solution:

```
(define/contract (minimize-set divisor1 divisor2 uniqueCnt1 uniqueCnt2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```