Problem 3494: Find the Minimum Amount of Time to Brew Potions

Problem Information

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

Problem Description

You are given two integer arrays,

skill

and

mana

, of length

n

and

m

, respectively.

In a laboratory,

n

wizards must brew

m

potions

in order

. Each potion has a mana capacity

mana[j]

and

must

pass through

all

the wizards sequentially to be brewed properly. The time taken by the

i

th

wizard on the

j

th

potion is

time

ij

= skill[i] * mana[j]

.

Since the brewing process is delicate, a potion

must

be passed to the next wizard immediately after the current wizard completes their work. This means the timing must be

synchronized

so that each wizard begins working on a potion

exactly

when it arrives.

Return the

minimum

amount of time required for the potions to be brewed properly.

Example 1:

Input:

skill = [1,5,2,4], mana = [5,1,4,2]

Output:

110

Explanation:

Potion Number

Start time

Wizard 0 done by

Wizard 1 done by

Wizard 2 done by

Wizard 3 done by

0

0

5

30

40

60

1

52

53

58

60

64

2

54

58

78

86

102

3

86

88

98

102

110

As an example for why wizard 0 cannot start working on the 1

st

potion before time

t = 52

, consider the case where the wizards started preparing the 1

st

potion at time

t = 50

. At time

t = 58

, wizard 2 is done with the 1

st

potion, but wizard 3 will still be working on the 0

th

potion till time

t = 60

.

Example 2:

Input:

skill = [1,1,1], mana = [1,1,1]

Output:

5

Explanation:

Preparation of the 0

th

potion begins at time

t = 0

, and is completed by time

t = 3

.

Preparation of the 1

st

potion begins at time

t = 1

, and is completed by time

t = 4

.

Preparation of the 2

nd

potion begins at time

t = 2

, and is completed by time

t = 5

.

Example 3:

Input:

skill = [1,2,3,4], mana = [1,2]

Output:

21

Constraints:

n == skill.length

m == mana.length

1 <= n, m <= 5000

1 <= mana[i], skill[i] <= 5000

Code Snippets

C++:

```
class Solution {
public:
long long minTime(vector<int>& skill, vector<int>& mana) {

}
};
```

Java:

```
class Solution {
public long minTime(int[] skill, int[] mana) {

}
}
```

Python3:

```
class Solution:
def minTime(self, skill: List[int], mana: List[int]) -> int:
```

Python:

```
class Solution(object):
def minTime(self, skill, mana):
"""
:type skill: List[int]
:type mana: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} skill
* @param {number[]} mana
* @return {number}
*/
var minTime = function(skill, mana) {

};
```

TypeScript:

```
function minTime(skill: number[], mana: number[]): number {

};
```

C#:

```
public class Solution {
public long MinTime(int[] skill, int[] mana) {

}
}
```

C:

```
long long minTime(int* skill, int skillSize, int* mana, int manaSize) {

}
```

Go:

```
func minTime(skill []int, mana []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minTime(skill: IntArray, mana: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minTime(_ skill: [Int], _ mana: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_time(skill: Vec<i32>, mana: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} skill
# @param {Integer[]} mana
# @return {Integer}
def min_time(skill, mana)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $skill
* @param Integer[] $mana
* @return Integer
*/
function minTime($skill, $mana) {

}
}
```

Dart:

```
class Solution {
int minTime(List<int> skill, List<int> mana) {
```

```
}
}
```

Scala:

```
object Solution {
def minTime(skill: Array[Int], mana: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_time(skill :: [integer], mana :: [integer]) :: integer
def min_time(skill, mana) do

end
end
```

Erlang:

```
-spec min_time(Skill :: [integer()], Mana :: [integer()]) -> integer().
min_time(Skill, Mana) ->
.
```

Racket:

```
(define/contract (min-time skill mana)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
```

```
* Tags: array
*
* 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:
long long minTime(vector<int>& skill, vector<int>& mana) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
* Tags: array
*
* 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 long minTime(int[] skill, int[] mana) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Minimum Amount of Time to Brew Potions
Difficulty: Medium
Tags: array

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 minTime(self, skill: List[int], mana: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minTime(self, skill, mana):
"""
:type skill: List[int]
:type mana: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
* Tags: array
*
* 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[]} skill
* @param {number[]} mana
* @return {number}
*/
var minTime = function(skill, mana) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
* Tags: array
*
* 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 minTime(skill: number[], mana: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
* Tags: array
*
* 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 long MinTime(int[] skill, int[] mana) {

}
}
```

C Solution:

```
/*
* Problem: Find the Minimum Amount of Time to Brew Potions
* Difficulty: Medium
* Tags: array
*
* 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
```

```
*/

long long minTime(int* skill, int skillSize, int* mana, int manaSize) {

}
```

Go Solution:

```
// Problem: Find the Minimum Amount of Time to Brew Potions
// Difficulty: Medium
// Tags: array
//
// 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 minTime(skill []int, mana []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minTime(skill: IntArray, mana: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minTime(_ skill: [Int], _ mana: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Minimum Amount of Time to Brew Potions
// Difficulty: Medium
// Tags: array
```

```
//
// 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 min_time(skill: Vec<i32>, mana: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} skill
# @param {Integer[]} mana
# @return {Integer}
def min_time(skill, mana)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $skill
* @param Integer[] $mana
* @return Integer
*/
function minTime($skill, $mana) {

}
}
```

Dart Solution:

```
class Solution {
int minTime(List<int> skill, List<int> mana) {

}
}
```

Scala Solution:

```
object Solution {
def minTime(skill: Array[Int], mana: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_time(skill :: [integer], mana :: [integer]) :: integer
def min_time(skill, mana) do

end
end
```

Erlang Solution:

```
-spec min_time(Skill :: [integer()], Mana :: [integer()]) -> integer().
min_time(Skill, Mana) ->
.
```

Racket Solution:

```
(define/contract (min-time skill mana)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```