Problem 3733: Minimum Time to Complete All Deliveries

Problem Information

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

Problem Description

You are given two integer arrays of size 2:

d = [d

1

, d

2

]

and

r = [r

1

, r

2

]

.

Two delivery drones are tasked with completing a specific number of deliveries. Drone

i

must complete

d

i

deliveries.

Each delivery takes

exactly

one hour and

only one

drone can make a delivery at any given hour.

Additionally, both drones require recharging at specific intervals during which they cannot make deliveries. Drone

i

must recharge every

r

i

hours (i.e. at hours that are multiples of

r

i

).

Return an integer denoting the

minimum

total time (in hours) required to complete all deliveries.

Example 1:

Input:

d = [3,1], r = [2,3]

Output:

5

Explanation:

The first drone delivers at hours 1, 3, 5 (recharges at hours 2, 4).

The second drone delivers at hour 2 (recharges at hour 3).

Example 2:

Input:

d = [1,3], r = [2,2]

Output:

7

Explanation:

The first drone delivers at hour 3 (recharges at hours 2, 4, 6).

The second drone delivers at hours 1, 5, 7 (recharges at hours 2, 4, 6).

Example 3:

Input:

d = [2,1], r = [3,4]

Output:

3

Explanation:

The first drone delivers at hours 1, 2 (recharges at hour 3).

The second drone delivers at hour 3.

Constraints:

d = [d

1

, d

2

]

1 <= d

i

<= 10

9

r = [r

1

, r

2

]

2 <= r

i

<= 3 * 10

4

Code Snippets

C++:

```
class Solution {
public:
long long minimumTime(vector<int>& d, vector<int>& r) {

}
};
```

Java:

```
class Solution {
public long minimumTime(int[] d, int[] r) {

}
}
```

Python3:

```
class Solution:
def minimumTime(self, d: List[int], r: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumTime(self, d, r):
"""
:type d: List[int]
:type r: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} d
* @param {number[]} r
* @return {number}
*/
var minimumTime = function(d, r) {

};
```

TypeScript:

```
function minimumTime(d: number[], r: number[]): number {

};
```

C#:

```
public class Solution {
public long MinimumTime(int[] d, int[] r) {

}
}
```

C:

```
long long minimumTime(int* d, int dSize, int* r, int rSize) {

}
```

Go:

```
func minimumTime(d []int, r []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumTime(d: IntArray, r: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minimumTime(_ d: [Int], _ r: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_time(d: Vec<i32>, r: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} d
# @param {Integer[]} r
# @return {Integer}
def minimum_time(d, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $d
```

```
* @param Integer[] $r
* @return Integer
*/
function minimumTime($d, $r) {

}
}
```

Dart:

```
class Solution {
int minimumTime(List<int> d, List<int> r) {

}
}
```

Scala:

```
object Solution {
def minimumTime(d: Array[Int], r: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time(d :: [integer], r :: [integer]) :: integer
def minimum_time(d, r) do

end
end
```

Erlang:

```
-spec minimum_time(D :: [integer()], R :: [integer()]) -> integer().
minimum_time(D, R) ->
.
```

Racket:

```
(define/contract (minimum-time d r)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Complete All Deliveries
* 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:
long long minimumTime(vector<int>& d, vector<int>& r) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Complete All Deliveries
* 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 long minimumTime(int[] d, int[] r) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Complete All Deliveries
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 minimumTime(self, d: List[int], r: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumTime(self, d, r):
"""
:type d: List[int]
:type r: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Complete All Deliveries
* 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[]} d
* @param {number[]} r
* @return {number}
*/
var minimumTime = function(d, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Complete All Deliveries
* 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 minimumTime(d: number[], r: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Complete All Deliveries
* 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 long MinimumTime(int[] d, int[] r) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Time to Complete All Deliveries
* 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
*/

long long minimumTime(int* d, int dSize, int* r, int rSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Complete All Deliveries
// 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 minimumTime(d []int, r []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTime(d: IntArray, r: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumTime(_ d: [Int], _ r: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Complete All Deliveries
// 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 minimum_time(d: Vec<i32>, r: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} d
# @param {Integer[]} r
# @return {Integer}
def minimum_time(d, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $d
* @param Integer[] $r
* @return Integer
*/
function minimumTime($d, $r) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int minimumTime(List<int> d, List<int> r) {

}
}
```

Scala Solution:

```
object Solution {
def minimumTime(d: Array[Int], r: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time(d :: [integer], r :: [integer]) :: integer
def minimum_time(d, r) do

end
end
```

Erlang Solution:

```
-spec minimum_time(D :: [integer()], R :: [integer()]) -> integer().
minimum_time(D, R) ->
.
```

Racket Solution:

```
(define/contract (minimum-time d r)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```