Problem 2187: Minimum Time to Complete Trips

Problem Information

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

Problem Description

You are given an array

time

where

time[i]

denotes the time taken by the

i

th

bus to complete

one trip

.

Each bus can make multiple trips

successively

; that is, the next trip can start

immediately after

completing the current trip. Also, each bus operates

independently

; that is, the trips of one bus do not influence the trips of any other bus.

You are also given an integer

totalTrips

, which denotes the number of trips all buses should make

in total

. Return

the

minimum time

required for all buses to complete

at least

totalTrips

trips

.

Example 1:

Input:

time = [1,2,3], totalTrips = 5

Output:

3

Explanation:

- At time t = 1, the number of trips completed by each bus are [1,0,0]. The total number of trips completed is 1 + 0 + 0 = 1. - At time t = 2, the number of trips completed by each bus are [2,1,0]. The total number of trips completed is 2 + 1 + 0 = 3. - At time t = 3, the number of trips completed by each bus are [3,1,1]. The total number of trips completed is 3 + 1 + 1 = 5. So the minimum time needed for all buses to complete at least 5 trips is 3.

Example 2:

Input:

time = [2], totalTrips = 1

Output:

2

Explanation:

There is only one bus, and it will complete its first trip at t = 2. So the minimum time needed to complete 1 trip is 2.

Constraints:

1 <= time.length <= 10

5

1 <= time[i], totalTrips <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long minimumTime(int[] time, int totalTrips) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

}

TypeScript:

```
function minimumTime(time: number[], totalTrips: number): number {

};
```

C#:

```
public class Solution {
public long MinimumTime(int[] time, int totalTrips) {

}
}
```

C:

```
long long minimumTime(int* time, int timeSize, int totalTrips) {

}
```

Go:

```
func minimumTime(time []int, totalTrips int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumTime(time: IntArray, totalTrips: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} time
# @param {Integer} total_trips
# @return {Integer}
def minimum_time(time, total_trips)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $time
* @param Integer $totalTrips
* @return Integer
*/
function minimumTime($time, $totalTrips) {

}
}
```

Dart:

```
class Solution {
int minimumTime(List<int> time, int totalTrips) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_time(time :: [integer], total_trips :: integer) :: integer
def minimum_time(time, total_trips) do

end
end
```

Erlang:

```
-spec minimum_time(Time :: [integer()], TotalTrips :: integer()) ->
integer().
minimum_time(Time, TotalTrips) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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>& time, int totalTrips) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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[] time, int totalTrips) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Complete Trips
Difficulty: Medium
Tags: array, 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, time: List[int], totalTrips: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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[]} time
* @param {number} totalTrips
* @return {number}
*/
var minimumTime = function(time, totalTrips) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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(time: number[], totalTrips: number): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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[] time, int totalTrips) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Complete Trips
* Difficulty: Medium
* Tags: array, 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* time, int timeSize, int totalTrips) {

}
```

Go Solution:

```
// Problem: Minimum Time to Complete Trips
// Difficulty: Medium
// Tags: array, 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(time []int, totalTrips int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumTime(time: IntArray, totalTrips: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Complete Trips
// Difficulty: Medium
// Tags: array, 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(time: Vec<i32>, total_trips: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} time
# @param {Integer} total_trips
# @return {Integer}
def minimum_time(time, total_trips)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $time
* @param Integer $totalTrips
* @return Integer
*/
function minimumTime($time, $totalTrips) {

}
}
```

Dart Solution:

```
class Solution {
int minimumTime(List<int> time, int totalTrips) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_time(time :: [integer], total_trips :: integer) :: integer
def minimum_time(time, total_trips) do

end
end
```

Erlang Solution:

```
-spec minimum_time(Time :: [integer()], TotalTrips :: integer()) ->
integer().
minimum_time(Time, TotalTrips) ->
.
```

Racket Solution:

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