Problem 3506: Find Time Required to Eliminate Bacterial Strains

Problem Information

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

Problem Description

You are given an integer array

timeReq

and an integer

splitTime

.

In the microscopic world of the human body, the immune system faces an extraordinary challenge: combatting a rapidly multiplying bacterial colony that threatens the body's survival.

Initially, only one

white blood cell

(

WBC

) is deployed to eliminate the bacteria. However, the lone WBC quickly realizes it cannot keep up with the bacterial growth rate.

The WBC devises a clever strategy to fight the bacteria:

The

i

th

bacterial strain takes

timeReq[i]

units of time to be eliminated.

A single WBC can eliminate

only one

bacterial strain. Afterwards, the WBC is exhausted and cannot perform any other tasks.

A WBC can split itself into two WBCs, but this requires

splitTime

units of time. Once split, the two WBCs can work in

parallel

on eliminating the bacteria.

Only one

WBC can work on a single bacterial strain. Multiple WBCs

cannot

attack one strain in parallel.

You must determine the

minimum

time required to eliminate all the bacterial strains.

Note

that the bacterial strains can be eliminated in any order.

Example 1:

Input:

timeReq = [10,4,5], splitTime = 2

Output:

12

Explanation:

The elimination process goes as follows:

Initially, there is a single WBC. The WBC splits into 2 WBCs after 2 units of time.

One of the WBCs eliminates strain 0 at a time

t = 2 + 10 = 12.

The other WBC splits again, using 2 units of time.

The 2 new WBCs eliminate the bacteria at times

t = 2 + 2 + 4

and

t = 2 + 2 + 5

.

Example 2:

Input:

timeReq = [10,4], splitTime = 5

Output:

15

Explanation:

The elimination process goes as follows:

Initially, there is a single WBC. The WBC splits into 2 WBCs after 5 units of time.

The 2 new WBCs eliminate the bacteria at times

t = 5 + 10

and

t = 5 + 4

.

Constraints:

2 <= timeReq.length <= 10

5

1 <= timeReq[i] <= 10

9

1 <= splitTime <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long minEliminationTime(vector<int>& timeReq, int splitTime) {

}
};
```

Java:

```
class Solution {
public long minEliminationTime(int[] timeReq, int splitTime) {

}
}
```

Python3:

```
class Solution:
def minEliminationTime(self, timeReq: List[int], splitTime: int) -> int:
```

Python:

```
class Solution(object):
def minEliminationTime(self, timeReq, splitTime):
"""
:type timeReq: List[int]
:type splitTime: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} timeReq
* @param {number} splitTime
* @return {number}
*/
var minEliminationTime = function(timeReq, splitTime) {

};
```

TypeScript:

```
function minEliminationTime(timeReq: number[], splitTime: number): number {

};
```

C#:

```
public class Solution {
public long MinEliminationTime(int[] timeReq, int splitTime) {

}
}
```

C:

```
long long minEliminationTime(int* timeReq, int timeReqSize, int splitTime) {

}
```

Go:

```
func minEliminationTime(timeReq []int, splitTime int) int64 {

}
```

Kotlin:

```
class Solution {
fun minEliminationTime(timeReq: IntArray, splitTime: Int): Long {

}
}
```

Swift:

```
class Solution {
func minEliminationTime(_ timeReq: [Int], _ splitTime: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_elimination_time(time_req: Vec<i32>, split_time: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} time_req
# @param {Integer} split_time
# @return {Integer}
def min_elimination_time(time_req, split_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $timeReq
* @param Integer $splitTime
* @return Integer
*/
function minEliminationTime($timeReq, $splitTime) {

}
}
```

Dart:

```
class Solution {
int minEliminationTime(List<int> timeReq, int splitTime) {

}
}
```

Scala:

```
object Solution {
def minEliminationTime(timeReq: Array[Int], splitTime: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_elimination_time(time_req :: [integer], split_time :: integer) ::
integer
def min_elimination_time(time_req, split_time) do

end
end
```

Erlang:

```
-spec min_elimination_time(TimeReq :: [integer()], SplitTime :: integer()) ->
integer().
min_elimination_time(TimeReq, SplitTime) ->
.
```

Racket:

```
(define/contract (min-elimination-time timeReq splitTime)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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 minEliminationTime(vector<int>& timeReq, int splitTime) {

}
};
```

Java Solution:

```
/**
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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 minEliminationTime(int[] timeReq, int splitTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Time Required to Eliminate Bacterial Strains
Difficulty: Hard
Tags: array, greedy, math, queue, heap

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 minEliminationTime(self, timeReq: List[int], splitTime: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minEliminationTime(self, timeReq, splitTime):
"""
:type timeReq: List[int]
:type splitTime: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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[]} timeReq
* @param {number} splitTime
* @return {number}
*/
var minEliminationTime = function(timeReq, splitTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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 minEliminationTime(timeReq: number[], splitTime: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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 MinEliminationTime(int[] timeReq, int splitTime) {

}
}
```

C Solution:

```
/*
* Problem: Find Time Required to Eliminate Bacterial Strains
* Difficulty: Hard
* Tags: array, greedy, math, queue, heap
*
* 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 minEliminationTime(int* timeReq, int timeReqSize, int splitTime) {

}
```

Go Solution:

```
// Problem: Find Time Required to Eliminate Bacterial Strains
// Difficulty: Hard
// Tags: array, greedy, math, queue, heap
//
// 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 minEliminationTime(timeReq []int, splitTime int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minEliminationTime(timeReq: IntArray, splitTime: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func minEliminationTime(_ timeReq: [Int], _ splitTime: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Time Required to Eliminate Bacterial Strains
// Difficulty: Hard
// Tags: array, greedy, math, queue, heap
```

```
//
// 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_elimination_time(time_req: Vec<i32>, split_time: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} time_req
# @param {Integer} split_time
# @return {Integer}
def min_elimination_time(time_req, split_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $timeReq
* @param Integer $splitTime
* @return Integer
*/
function minEliminationTime($timeReq, $splitTime) {

}
}
```

Dart Solution:

```
class Solution {
int minEliminationTime(List<int> timeReq, int splitTime) {

}
}
```

Scala Solution:

```
object Solution {
def minEliminationTime(timeReq: Array[Int], splitTime: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_elimination_time(time_req :: [integer], split_time :: integer) :: integer
def min_elimination_time(time_req, split_time) do

end
end
```

Erlang Solution:

```
-spec min_elimination_time(TimeReq :: [integer()], SplitTime :: integer()) -> integer().
min_elimination_time(TimeReq, SplitTime) ->
.
```

Racket Solution:

```
(define/contract (min-elimination-time timeReq splitTime)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```