Problem 3376: Minimum Time to Break Locks I

Problem Information

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

Problem Description

Bob is stuck in a dungeon and must break

n

locks, each requiring some amount of

energy

to break. The required energy for each lock is stored in an array called

strength

where

strength[i]

indicates the energy needed to break the

i

th

lock.

To break a lock, Bob uses a sword with the following characteristics:

The initial energy of the sword is 0.

The initial factor

x

by which the energy of the sword increases is 1.

Every minute, the energy of the sword increases by the current factor

x

.

To break the

i

th

lock, the energy of the sword must reach

at least

strength[i]

.

After breaking a lock, the energy of the sword resets to 0, and the factor

x

increases by a given value

k

.

Your task is to determine the

minimum

time in minutes required for Bob to break all

n

locks and escape the dungeon.

Return the

minimum

time required for Bob to break all

n

locks.

Example 1:

Input:

strength = [3,4,1], k = 1

Output:

4

Explanation:

Time

Energy

x

Action

Updated x

0

0

1

Nothing

1

1

1

1

Break 3

rd

Lock

2

2

2

2

Nothing

2

3

4

2

Break 2

nd

Lock

3

4

3

3

Break 1

st

Lock

3

The locks cannot be broken in less than 4 minutes; thus, the answer is 4.

Example 2:

Input:

strength = [2,5,4], k = 2

Output:

5

Explanation:

Time

Energy

x

Action

Updated x

0

0

1

Nothing

1

1

1

1

Nothing

1

2

2

1

Break 1

st

Lock

3

3

3

3

Nothing

3

4

6

3

Break 2

n

d

Lock

5

5

5

5

Break 3

r

d

Lock

7

The locks cannot be broken in less than 5 minutes; thus, the answer is 5.

Constraints:

n == strength.length

1 <= n <= 8

1 <= K <= 10

1 <= strength[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int findMinimumTime(vector<int>& strength, int k) {

}
};
```

Java:

```
class Solution {
public int findMinimumTime(List<Integer> strength, int k) {

}
```

```
}
```

Python3:

```
class Solution:
def findMinimumTime(self, strength: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def findMinimumTime(self, strength, k):
"""
:type strength: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} strength
* @param {number} k
* @return {number}
*/
var findMinimumTime = function(strength, k) {

};
```

TypeScript:

```
function findMinimumTime(strength: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int FindMinimumTime(IList<int> strength, int k) {

}
}
```

C:

```
int findMinimumTime(int* strength, int strengthSize, int k) {

}
```

Go:

```
func findMinimumTime(strength []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findMinimumTime(strength: List<Int>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func findMinimumTime(_ strength: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_minimum_time(strength: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} strength
# @param {Integer} k
# @return {Integer}
def find_minimum_time(strength, k)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $strength
* @param Integer $k
* @return Integer
*/
function findMinimumTime($strength, $k) {

}
}
```

Dart:

```
class Solution {
int findMinimumTime(List<int> strength, int k) {

}
}
```

Scala:

```
object Solution {
def findMinimumTime(strength: List[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_minimum_time(strength :: [integer], k :: integer) :: integer
def find_minimum_time(strength, k) do

end
end
```

Erlang:

```
-spec find_minimum_time(Strength :: [integer()], K :: integer()) ->
integer().
find_minimum_time(Strength, K) ->
.
```

Racket:

```
(define/contract (find-minimum-time strength k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int findMinimumTime(vector<int>& strength, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int findMinimumTime(List<Integer> strength, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Break Locks I
Difficulty: Medium
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findMinimumTime(self, strength: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMinimumTime(self, strength, k):
"""
:type strength: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
```

```
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} strength
* @param {number} k
* @return {number}
*/
var findMinimumTime = function(strength, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findMinimumTime(strength: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int FindMinimumTime(IList<int> strength, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Break Locks I
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int findMinimumTime(int* strength, int strengthSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Time to Break Locks I
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func findMinimumTime(strength []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMinimumTime(strength: List<Int>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMinimumTime(_ strength: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Break Locks I
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_minimum_time(strength: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} strength
# @param {Integer} k
# @return {Integer}
def find_minimum_time(strength, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $strength
* @param Integer $k
* @return Integer
*/
function findMinimumTime($strength, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findMinimumTime(List<int> strength, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findMinimumTime(strength: List[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_minimum_time(strength :: [integer], k :: integer) :: integer
def find_minimum_time(strength, k) do

end
end
```

Erlang Solution:

```
-spec find_minimum_time(Strength :: [integer()], K :: integer()) ->
integer().
find_minimum_time(Strength, K) ->
```

```
.
```

Racket Solution:

```
(define/contract (find-minimum-time strength k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```