Problem 1883: Minimum Skips to Arrive at Meeting On Time

Problem Information

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

Problem Description

You are given an integer

hoursBefore

, the number of hours you have to travel to your meeting. To arrive at your meeting, you have to travel through

n

roads. The road lengths are given as an integer array

dist

of length

n

, where

dist[i]

describes the length of the

i

th

road in

kilometers

. In addition, you are given an integer

speed

, which is the speed (in

km/h

) you will travel at.

After you travel road

i

, you must rest and wait for the

next integer hour

before you can begin traveling on the next road. Note that you do not have to rest after traveling the last road because you are already at the meeting.

For example, if traveling a road takes

1.4

hours, you must wait until the

2

hour mark before traveling the next road. If traveling a road takes exactly

2

hours, you do not need to wait.

However, you are allowed to

skip

some rests to be able to arrive on time, meaning you do not need to wait for the next integer hour. Note that this means you may finish traveling future roads at different hour marks.

For example, suppose traveling the first road takes

1.4

hours and traveling the second road takes

0.6

hours. Skipping the rest after the first road will mean you finish traveling the second road right at the

2

hour mark, letting you start traveling the third road immediately.

Return

the

minimum number of skips required

to arrive at the meeting on time, or

-1

if it is

impossible

.

Example 1:

Input:

dist = [1,3,2], speed = 4, hoursBefore = 2

Output:

1

Explanation:

Without skipping any rests, you will arrive in (1/4 + 3/4) + (3/4 + 1/4) + (2/4) = 2.5 hours. You can skip the first rest to arrive in ((1/4 +

0

) + (3/4 + 0)) + (2/4) = 1.5 hours. Note that the second rest is shortened because you finish traveling the second road at an integer hour due to skipping the first rest.

Example 2:

Input:

dist = [7,3,5,5], speed = 2, hoursBefore = 10

Output:

2

Explanation:

Without skipping any rests, you will arrive in (7/2 + 1/2) + (3/2 + 1/2) + (5/2 + 1/2) + (5/2) = 11.5 hours. You can skip the first and third rest to arrive in ((7/2 +

0

) + (3/2 + 0)) + ((5/2 +

0

) + (5/2)) = 10 hours.

Example 3:

Input:

dist = [7,3,5,5], speed = 1, hoursBefore = 10

Output:

-1

Explanation:

It is impossible to arrive at the meeting on time even if you skip all the rests.

Constraints:

n == dist.length

1 <= n <= 1000

1 <= dist[i] <= 10

5

1 <= speed <= 10

6

1 <= hoursBefore <= 10

7

Code Snippets

C++:

```
class Solution {
public:
int minSkips(vector<int>& dist, int speed, int hoursBefore) {

}
};
```

Java:

```
class Solution {
public int minSkips(int[] dist, int speed, int hoursBefore) {

}
}
```

Python3:

```
class Solution:
def minSkips(self, dist: List[int], speed: int, hoursBefore: int) -> int:
```

Python:

```
class Solution(object):
def minSkips(self, dist, speed, hoursBefore):
"""
:type dist: List[int]
:type speed: int
:type hoursBefore: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} dist
* @param {number} speed
* @param {number} hoursBefore
* @return {number}
*/
var minSkips = function(dist, speed, hoursBefore) {
```

```
};
```

TypeScript:

```
function minSkips(dist: number[], speed: number, hoursBefore: number): number
{

};
```

C#:

```
public class Solution {
public int MinSkips(int[] dist, int speed, int hoursBefore) {

}
}
```

C:

```
int minSkips(int* dist, int distSize, int speed, int hoursBefore) {

}
```

Go:

```
func minSkips(dist []int, speed int, hoursBefore int) int {

}
```

Kotlin:

```
class Solution {
fun minSkips(dist: IntArray, speed: Int, hoursBefore: Int): Int {

}
}
```

Swift:

```
class Solution {
func minSkips(_ dist: [Int], _ speed: Int, _ hoursBefore: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn min_skips(dist: Vec<i32>, speed: i32, hours_before: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} dist
# @param {Integer} speed
# @param {Integer} hours_before
# @return {Integer}
def min_skips(dist, speed, hours_before)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $dist
* @param Integer $speed
* @param Integer $hoursBefore
* @return Integer
*/
function minSkips($dist, $speed, $hoursBefore) {

}
}
```

Dart:

```
class Solution {
int minSkips(List<int> dist, int speed, int hoursBefore) {

}
```

```
}
```

Scala:

```
object Solution {
def minSkips(dist: Array[Int], speed: Int, hoursBefore: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_skips(dist :: [integer], speed :: integer, hours_before :: integer)
:: integer
def min_skips(dist, speed, hours_before) do

end
end
```

Erlang:

```
-spec min_skips(Dist :: [integer()], Speed :: integer(), HoursBefore ::
integer()) -> integer().
min_skips(Dist, Speed, HoursBefore) ->
.
```

Racket:

```
(define/contract (min-skips dist speed hoursBefore)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
```

```
* Tags: array, dp
*
* 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 minSkips(vector<int>& dist, int speed, int hoursBefore) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
* Tags: array, dp
*
* 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 minSkips(int[] dist, int speed, int hoursBefore) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Skips to Arrive at Meeting On Time
Difficulty: Hard
Tags: array, dp

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 minSkips(self, dist: List[int], speed: int, hoursBefore: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSkips(self, dist, speed, hoursBefore):
"""
:type dist: List[int]
:type speed: int
:type hoursBefore: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
* Tags: array, dp
*
* 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[]} dist
* @param {number} speed
* @param {number} hoursBefore
* @return {number}
*/
var minSkips = function(dist, speed, hoursBefore) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
* Tags: array, dp
*
* 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 minSkips(dist: number[], speed: number, hoursBefore: number): number
{

};
```

C# Solution:

```
/*
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
* Tags: array, dp
*
* 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 MinSkips(int[] dist, int speed, int hoursBefore) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Skips to Arrive at Meeting On Time
* Difficulty: Hard
* Tags: array, dp
*
* 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 minSkips(int* dist, int distSize, int speed, int hoursBefore) {

}
```

Go Solution:

```
// Problem: Minimum Skips to Arrive at Meeting On Time
// Difficulty: Hard
// Tags: array, dp
//
// 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 minSkips(dist []int, speed int, hoursBefore int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSkips(dist: IntArray, speed: Int, hoursBefore: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSkips(_ dist: [Int], _ speed: Int, _ hoursBefore: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Skips to Arrive at Meeting On Time
// Difficulty: Hard
// Tags: array, dp
//
// 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 min_skips(dist: Vec<i32>, speed: i32, hours_before: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} dist
# @param {Integer} speed
# @param {Integer} hours_before
# @return {Integer}
def min_skips(dist, speed, hours_before)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $dist
* @param Integer $speed
* @param Integer $hoursBefore
* @return Integer
*/
function minSkips($dist, $speed, $hoursBefore) {

}
}
```

Dart Solution:

```
class Solution {
int minSkips(List<int> dist, int speed, int hoursBefore) {

}
}
```

Scala Solution:

```
object Solution {
def minSkips(dist: Array[Int], speed: Int, hoursBefore: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_skips(dist :: [integer], speed :: integer, hours_before :: integer)
:: integer
def min_skips(dist, speed, hours_before) do

end
end
```

Erlang Solution:

```
-spec min_skips(Dist :: [integer()], Speed :: integer(), HoursBefore ::
integer()) -> integer().
min_skips(Dist, Speed, HoursBefore) ->
.
```

Racket Solution:

```
(define/contract (min-skips dist speed hoursBefore)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```