Problem 1870: Minimum Speed to Arrive on Time

Problem Information

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

Problem Description

You are given a floating-point number

hour

, representing the amount of time you have to reach the office. To commute to the office, you must take

n

trains in sequential order. You are also given an integer array

dist

of length

n

, where

dist[i]

describes the distance (in kilometers) of the

i

th

train ride.

Each train can only depart at an integer hour, so you may need to wait in between each train ride.

For example, if the

1

st

train ride takes

1.5

hours, you must wait for an additional

0.5

hours before you can depart on the

2

nd

train ride at the 2 hour mark.

Return

the

minimum positive integer

speed

(in kilometers per hour)

that all the trains must travel at for you to reach the office on time, or

-1

if it is impossible to be on time

.

Tests are generated such that the answer will not exceed

10

7

and

hour

will have

at most two digits after the decimal point

.

Example 1:

Input:

dist = [1,3,2], hour = 6

Output:

1

Explanation:

At speed 1: - The first train ride takes 1/1 = 1 hour. - Since we are already at an integer hour, we depart immediately at the 1 hour mark. The second train takes 3/1 = 3 hours. - Since we are already at an integer hour, we depart immediately at the 4 hour mark. The third train takes

2/1 = 2 hours. - You will arrive at exactly the 6 hour mark.

Example 2:

Input:

dist = [1,3,2], hour = 2.7

Output:

3

Explanation:

At speed 3: - The first train ride takes 1/3 = 0.33333 hours. - Since we are not at an integer hour, we wait until the 1 hour mark to depart. The second train ride takes 3/3 = 1 hour. - Since we are already at an integer hour, we depart immediately at the 2 hour mark. The third train takes 2/3 = 0.66667 hours. - You will arrive at the 2.66667 hour mark.

Example 3:

Input:

dist = [1,3,2], hour = 1.9

Output:

-1

Explanation:

It is impossible because the earliest the third train can depart is at the 2 hour mark.

Constraints:

n == dist.length

1 <= n <= 10

5

1 <= dist[i] <= 10

5

1 <= hour <= 10

9

There will be at most two digits after the decimal point in

hour

.

Code Snippets

C++:

```
class Solution {
public:
int minSpeedOnTime(vector<int>& dist, double hour) {

}
};
```

Java:

```
class Solution {
public int minSpeedOnTime(int[] dist, double hour) {

}
}
```

Python3:

```
class Solution:
def minSpeedOnTime(self, dist: List[int], hour: float) -> int:
```

Python:

```
class Solution(object):
def minSpeedOnTime(self, dist, hour):
"""
:type dist: List[int]
:type hour: float
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} dist
* @param {number} hour
* @return {number}
*/
var minSpeedOnTime = function(dist, hour) {

};
```

TypeScript:

```
function minSpeedOnTime(dist: number[], hour: number): number {

};
```

C#:

```
public class Solution {
public int MinSpeedOnTime(int[] dist, double hour) {

}
}
```

C:

```
int minSpeedOnTime(int* dist, int distSize, double hour) {

}
```

Go:

```
func minSpeedOnTime(dist []int, hour float64) int {

}
```

Kotlin:

```
class Solution {
fun minSpeedOnTime(dist: IntArray, hour: Double): Int {

}
}
```

Swift:

```
class Solution {
func minSpeedOnTime(_ dist: [Int], _ hour: Double) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_speed_on_time(dist: Vec<i32>, hour: f64) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} dist
# @param {Float} hour
# @return {Integer}
def min_speed_on_time(dist, hour)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $dist
* @param Float $hour
* @return Integer
*/
function minSpeedOnTime($dist, $hour) {

}
}
```

Dart:

```
class Solution {
int minSpeedOnTime(List<int> dist, double hour) {

}
}
```

Scala:

```
object Solution {
def minSpeedOnTime(dist: Array[Int], hour: Double): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_speed_on_time(dist :: [integer], hour :: float) :: integer
def min_speed_on_time(dist, hour) do

end
end
```

Erlang:

```
-spec min_speed_on_time(Dist :: [integer()], Hour :: float()) -> integer().
min_speed_on_time(Dist, Hour) ->
.
```

Racket:

```
(define/contract (min-speed-on-time dist hour)
(-> (listof exact-integer?) flonum? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Speed to Arrive on Time
* 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:
int minSpeedOnTime(vector<int>& dist, double hour) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Speed to Arrive on Time
* 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 int minSpeedOnTime(int[] dist, double hour) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Speed to Arrive on Time
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 minSpeedOnTime(self, dist: List[int], hour: float) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSpeedOnTime(self, dist, hour):
"""
:type dist: List[int]
:type hour: float
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Speed to Arrive on Time
* 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[]} dist
* @param {number} hour
* @return {number}
*/
var minSpeedOnTime = function(dist, hour) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Speed to Arrive on Time
* 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 minSpeedOnTime(dist: number[], hour: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Speed to Arrive on Time
* 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 int MinSpeedOnTime(int[] dist, double hour) {

}

}
```

C Solution:

```
/*
* Problem: Minimum Speed to Arrive on Time
* 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
*/

int minSpeedOnTime(int* dist, int distSize, double hour) {

}
```

Go Solution:

```
// Problem: Minimum Speed to Arrive on Time
// 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 minSpeedOnTime(dist []int, hour float64) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSpeedOnTime(dist: IntArray, hour: Double): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSpeedOnTime(_ dist: [Int], _ hour: Double) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Speed to Arrive on Time
// 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 min_speed_on_time(dist: Vec<i32>, hour: f64) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} dist
# @param {Float} hour
# @return {Integer}
def min_speed_on_time(dist, hour)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $dist
* @param Float $hour
* @return Integer
*/
function minSpeedOnTime($dist, $hour) {

}
}
```

Dart Solution:

```
class Solution {
int minSpeedOnTime(List<int> dist, double hour) {

}
}
```

Scala Solution:

```
object Solution {
def minSpeedOnTime(dist: Array[Int], hour: Double): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_speed_on_time(dist :: [integer], hour :: float) :: integer
def min_speed_on_time(dist, hour) do

end
end
```

Erlang Solution:

```
-spec min_speed_on_time(Dist :: [integer()], Hour :: float()) -> integer().
min_speed_on_time(Dist, Hour) ->
.
```

Racket Solution:

```
(define/contract (min-speed-on-time dist hour)
(-> (listof exact-integer?) flonum? exact-integer?)
)
```