Problem 1936: Add Minimum Number of Rungs

Problem Information

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

Problem Description

You are given a

strictly increasing

integer array

rungs

that represents the

height

of rungs on a ladder. You are currently on the

floor

at height

0

, and you want to reach the last rung.

You are also given an integer

dist

. You can only climb to the next highest rung if the distance between where you are currently at (the floor or on a rung) and the next rung is

at most

dist

. You are able to insert rungs at any positive

integer

height if a rung is not already there.

Return

the

minimum

number of rungs that must be added to the ladder in order for you to climb to the last rung.

Example 1:

Input:

rungs = [1,3,5,10], dist = 2

Output:

2

Explanation:

You currently cannot reach the last rung. Add rungs at heights 7 and 8 to climb this ladder. The ladder will now have rungs at [1,3,5,

7

,

8

,10].

Example 2:

Input:

rungs = [3,6,8,10], dist = 3

Output:

0

Explanation:

This ladder can be climbed without adding additional rungs.

Example 3:

Input:

rungs = [3,4,6,7], dist = 2

Output:

1

Explanation:

You currently cannot reach the first rung from the ground. Add a rung at height 1 to climb this ladder. The ladder will now have rungs at [

1

,3,4,6,7].

Constraints:

1 <= rungs.length <= 10

5

1 <= rungs[i] <= 10

9

1 <= dist <= 10

9

rungs

is

strictly increasing

.

Code Snippets

C++:

```
class Solution {
public:
int addRungs(vector<int>& rungs, int dist) {

}
};
```

Java:

```
class Solution {
public int addRungs(int[] rungs, int dist) {

}
}
```

Python3:

```
class Solution:
def addRungs(self, rungs: List[int], dist: int) -> int:
```

Python:

```
class Solution(object):
def addRungs(self, rungs, dist):
"""
:type rungs: List[int]
:type dist: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function addRungs(rungs: number[], dist: number): number {

};
```

C#:

```
public class Solution {
public int AddRungs(int[] rungs, int dist) {

}
}
```

C:

```
int addRungs(int* rungs, int rungsSize, int dist) {

}
```

Go:

```
func addRungs(rungs []int, dist int) int {

}
```

Kotlin:

```
class Solution {
fun addRungs(rungs: IntArray, dist: Int): Int {

}
}
```

Swift:

```
class Solution {
func addRungs(_ rungs: [Int], _ dist: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn add_rungs(rungs: Vec<i32>, dist: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} rungs
# @param {Integer} dist
# @return {Integer}
def add_rungs(rungs, dist)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $rungs
* @param Integer $dist
* @return Integer
*/
function addRungs($rungs, $dist) {

}
}
```

Dart:

```
class Solution {
int addRungs(List<int> rungs, int dist) {

}
}
```

Scala:

```
object Solution {
def addRungs(rungs: Array[Int], dist: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec add_rungs(rungs :: [integer], dist :: integer) :: integer
def add_rungs(rungs, dist) do

end
end
```

Erlang:

```
-spec add_rungs(Rungs :: [integer()], Dist :: integer()) -> integer().
add_rungs(Rungs, Dist) ->
```

```
.
```

Racket:

```
(define/contract (add-rungs rungs dist)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
* 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 addRungs(vector<int>& rungs, int dist) {

}
};
```

Java Solution:

```
/**
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
* 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 addRungs(int[] rungs, int dist) {

}
}
```

Python3 Solution:

```
"""
Problem: Add Minimum Number of Rungs
Difficulty: Medium
Tags: array, greedy

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 addRungs(self, rungs: List[int], dist: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def addRungs(self, rungs, dist):
"""
:type rungs: List[int]
:type dist: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
* 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 addRungs(rungs: number[], dist: number): number {

};
```

C# Solution:

```
/*
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
* 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 AddRungs(int[] rungs, int dist) {

}
}
```

C Solution:

```
/*
* Problem: Add Minimum Number of Rungs
* Difficulty: Medium
* Tags: array, greedy
*
* 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 addRungs(int* rungs, int rungsSize, int dist) {

}
```

Go Solution:

```
// Problem: Add Minimum Number of Rungs
// Difficulty: Medium
// Tags: array, greedy
//
// 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 addRungs(rungs []int, dist int) int {

}
```

Kotlin Solution:

```
class Solution {
fun addRungs(rungs: IntArray, dist: Int): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func addRungs(_ rungs: [Int], _ dist: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Add Minimum Number of Rungs
// Difficulty: Medium
// Tags: array, greedy
//
// 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 add_rungs(rungs: Vec<i32>, dist: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} rungs
# @param {Integer} dist
# @return {Integer}
def add_rungs(rungs, dist)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $rungs
* @param Integer $dist
* @return Integer
*/
function addRungs($rungs, $dist) {

}
}
```

Dart Solution:

```
class Solution {
int addRungs(List<int> rungs, int dist) {

}
}
```

Scala Solution:

```
object Solution {
def addRungs(rungs: Array[Int], dist: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec add_rungs(rungs :: [integer], dist :: integer) :: integer
def add_rungs(rungs, dist) do

end
end
```

Erlang Solution:

```
-spec add_rungs(Rungs :: [integer()], Dist :: integer()) -> integer().
add_rungs(Rungs, Dist) ->
.
```

Racket Solution:

```
(define/contract (add-rungs rungs dist)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```