Problem 452: Minimum Number of Arrows to Burst Balloons

Problem Information

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

Problem Description

There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array

points

where

points[i] = [x

start

, x

end

]

denotes a balloon whose

horizontal diameter

stretches between

x

start

and

x

end

. You do not know the exact y-coordinates of the balloons.

Arrows can be shot up

directly vertically

(in the positive y-direction) from different points along the x-axis. A balloon with

x

start

and

x

end

is

burst

by an arrow shot at

x

if

x

start

<= x <= x

end

. There is

no limit

to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path.

Given the array

points

, return

the

minimum

number of arrows that must be shot to burst all balloons

.

Example 1:

Input:

points = [[10,16],[2,8],[1,6],[7,12]]

Output:

2

Explanation:

The balloons can be burst by 2 arrows: - Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6]. - Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12].

Example 2:

Input:

points = [[1,2],[3,4],[5,6],[7,8]]

Output:

4

Explanation:

One arrow needs to be shot for each balloon for a total of 4 arrows.

Example 3:

Input:

points = [[1,2],[2,3],[3,4],[4,5]]

Output:

2

Explanation:

The balloons can be burst by 2 arrows: - Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3]. - Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5].

Constraints:

1 <= points.length <= 10

5

points[i].length == 2

-2

31

<= x

start

< x

end

<= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
int findMinArrowShots(vector<vector<int>>& points) {

}
};
```

Java:

```
class Solution {
public int findMinArrowShots(int[][] points) {

}
}
```

Python3:

```
class Solution:
def findMinArrowShots(self, points: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def findMinArrowShots(self, points):
"""
:type points: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} points
* @return {number}
*/
var findMinArrowShots = function(points) {

};
```

TypeScript:

```
function findMinArrowShots(points: number[][]): number {

};
```

C#:

```
public class Solution {
public int FindMinArrowShots(int[][] points) {

}
}
```

C:

```
int findMinArrowShots(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func findMinArrowShots(points [][]int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun findMinArrowShots(points: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func findMinArrowShots(_ points: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_min_arrow_shots(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @return {Integer}
def find_min_arrow_shots(points)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
```

```
function findMinArrowShots($points) {

}
}
```

Dart:

```
class Solution {
int findMinArrowShots(List<List<int>> points) {

}
}
```

Scala:

```
object Solution {
def findMinArrowShots(points: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_min_arrow_shots(points :: [[integer]]) :: integer
def find_min_arrow_shots(points) do

end
end
```

Erlang:

```
-spec find_min_arrow_shots(Points :: [[integer()]]) -> integer().
find_min_arrow_shots(Points) ->
.
```

Racket:

```
(define/contract (find-min-arrow-shots points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 findMinArrowShots(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 findMinArrowShots(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Arrows to Burst Balloons
Difficulty: Medium
Tags: array, greedy, sort

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 findMinArrowShots(self, points: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMinArrowShots(self, points):
"""
:type points: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[][]} points
* @return {number}
*/
var findMinArrowShots = function(points) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 findMinArrowShots(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 FindMinArrowShots(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Arrows to Burst Balloons
* Difficulty: Medium
```

```
* Tags: array, greedy, sort
*
* 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 findMinArrowShots(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Arrows to Burst Balloons
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 findMinArrowShots(points [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMinArrowShots(points: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMinArrowShots(_ points: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Arrows to Burst Balloons
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 find_min_arrow_shots(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @return {Integer}
def find_min_arrow_shots(points)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function findMinArrowShots($points) {

}
}
```

Dart Solution:

```
class Solution {
int findMinArrowShots(List<List<int>> points) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def findMinArrowShots(points: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_min_arrow_shots(points :: [[integer]]) :: integer
def find_min_arrow_shots(points) do

end
end
```

Erlang Solution:

```
-spec find_min_arrow_shots(Points :: [[integer()]]) -> integer().
find_min_arrow_shots(Points) ->
.
```

Racket Solution:

```
(define/contract (find-min-arrow-shots points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```