# Problem 3111: Minimum Rectangles to Cover Points

## Problem Information

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

## Problem Description

You are given a 2D integer array

points

, where

points[i] = [x

i

, y

i

]

. You are also given an integer

w

. Your task is to

cover

all

the given points with rectangles.

Each rectangle has its lower end at some point

(x

1

, 0)

and its upper end at some point

(x

2

, y

2

)

, where

x

1

<= x

2

,

y

2

>= 0

, and the condition

x

2

- x

1

<= w

must

be satisfied for each rectangle.

A point is considered covered by a rectangle if it lies within or on the boundary of the rectangle.

Return an integer denoting the

minimum

number of rectangles needed so that each point is covered by

at least one

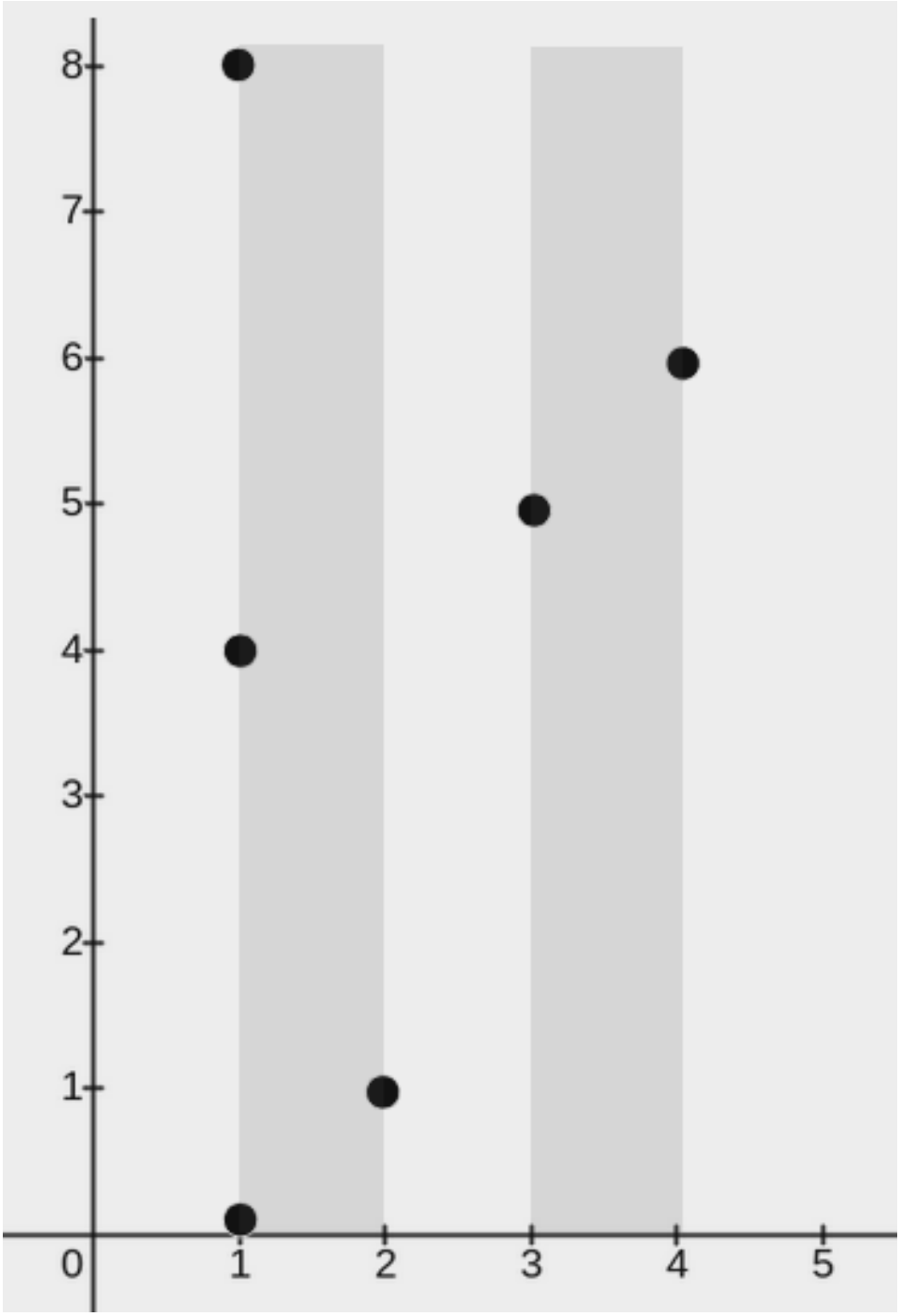
rectangle

.

Note:

A point may be covered by more than one rectangle.

Example 1:




Input:

points = [[2,1],[1,0],[1,4],[1,8],[3,5],[4,6]], w = 1

Output:

2

Explanation:

The image above shows one possible placement of rectangles to cover the points:

A rectangle with a lower end at

(1, 0)

and its upper end at

(2, 8)

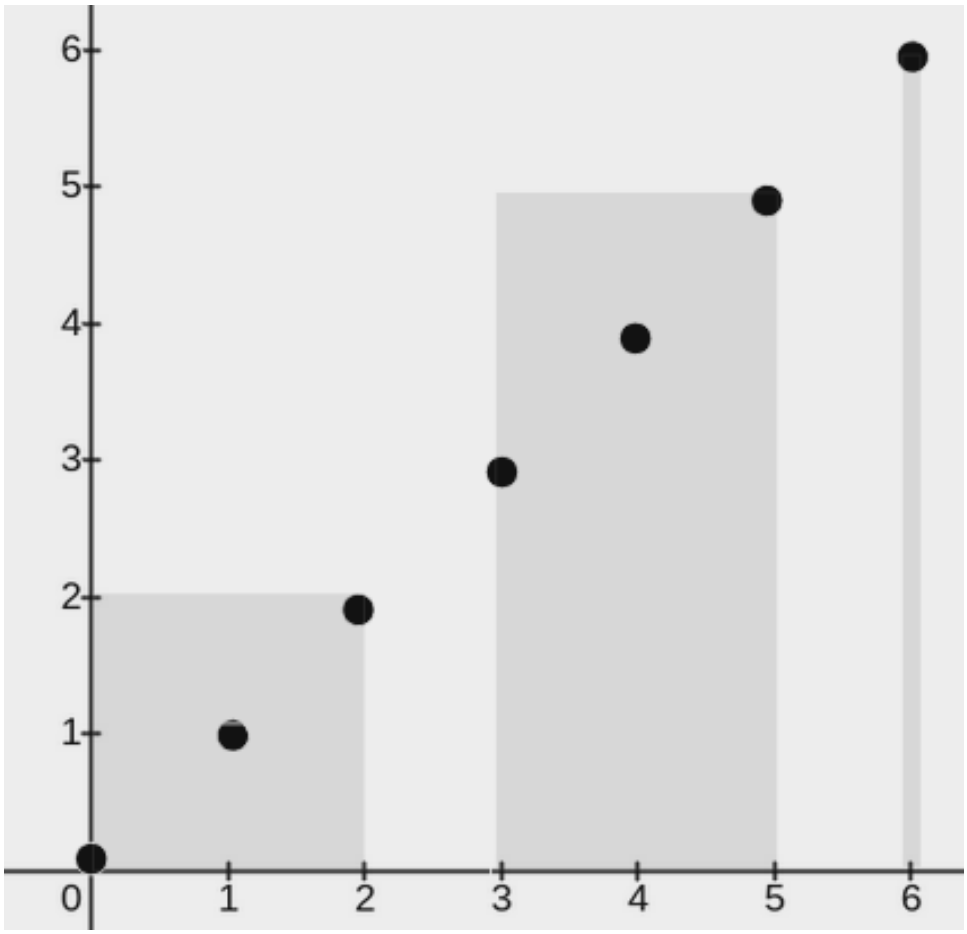
A rectangle with a lower end at

(3, 0)

and its upper end at

(4, 8)

Example 2:




Input:

points = [[0,0],[1,1],[2,2],[3,3],[4,4],[5,5],[6,6]], w = 2

Output:

3

Explanation:

The image above shows one possible placement of rectangles to cover the points:

A rectangle with a lower end at

(0, 0)

and its upper end at

(2, 2)

A rectangle with a lower end at

(3, 0)

and its upper end at

(5, 5)

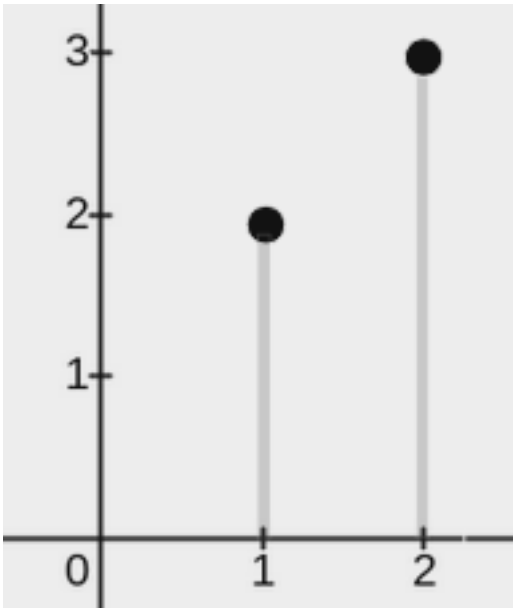
A rectangle with a lower end at

(6, 0)

and its upper end at

(6, 6)

Example 3:




Input:

points = [[2,3],[1,2]], w = 0

Output:

2

Explanation:

The image above shows one possible placement of rectangles to cover the points:

A rectangle with a lower end at

(1, 0)

and its upper end at

(1, 2)

A rectangle with a lower end at

(2, 0)

and its upper end at

(2, 3)

Constraints:

1 <= points.length <= 10

5

points[i].length == 2

0 <= x

i

== points[i][0] <= 10

9

0 <= y

i

== points[i][1] <= 10

9

0 <= w <= 10

9

All pairs

(x

i

, y

i

)

are distinct.

## Code Snippets

**C++:**

```
class Solution {
public:
int minRectanglesToCoverPoints(vector<vector<int>>& points, int w) {

}
};
```

**Java:**

```
class Solution {
public int minRectanglesToCoverPoints(int[][] points, int w) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function minRectanglesToCoverPoints(points: number[][], w: number): number {

};
```

**C#:**

```
public class Solution {
public int MinRectanglesToCoverPoints(int[][] points, int w) {
```

```
}
}
```

**C:**

```
int minRectanglesToCoverPoints(int** points, int pointsSize, int*
pointsColSize, int w) {

}
```

**Go:**

```
func minRectanglesToCoverPoints(points [][]int, w int) int {

}
```

**Kotlin:**

```
class Solution {
fun minRectanglesToCoverPoints(points: Array<IntArray>, w: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func minRectanglesToCoverPoints(_ points: [[Int]], _ w: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn min_rectangles_to_cover_points(points: Vec<Vec<i32>>, w: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} points
# @param {Integer} w
# @return {Integer}
def min_rectangles_to_cover_points(points, w)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $w
* @return Integer
*/
function minRectanglesToCoverPoints($points, $w) {

}
}
```

**Dart:**

```
class Solution {
int minRectanglesToCoverPoints(List<List<int>> points, int w) {

}
}
```

**Scala:**

```
object Solution {
def minRectanglesToCoverPoints(points: Array[Array[Int]], w: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec min_rectangles_to_cover_points(points :: [[integer]], w :: integer) ::
integer
def min_rectangles_to_cover_points(points, w) do
```

```
end
end
```

**Erlang:**

```
-spec min_rectangles_to_cover_points(Points :: [[integer()]], W :: integer())
-> integer().
min_rectangles_to_cover_points(Points, W) ->
.
```

**Racket:**

```
(define/contract (min-rectangles-to-cover-points points w)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Minimum Rectangles to Cover Points
* 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 minRectanglesToCoverPoints(vector<vector<int>>& points, int w) {

}
};
```

**Java Solution:**

```
/**
* Problem: Minimum Rectangles to Cover Points
* 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 minRectanglesToCoverPoints(int[][] points, int w) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Rectangles to Cover Points
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 minRectanglesToCoverPoints(self, points: List[List[int]], w: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Minimum Rectangles to Cover Points
* 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
* @param {number} w
* @return {number}
*/
var minRectanglesToCoverPoints = function(points, w) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Rectangles to Cover Points
* 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 minRectanglesToCoverPoints(points: number[][], w: number): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Rectangles to Cover Points
* 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 MinRectanglesToCoverPoints(int[][] points, int w) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Rectangles to Cover Points
* 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 minRectanglesToCoverPoints(int** points, int pointsSize, int*
pointsColSize, int w) {

}
```

### Go Solution:

```
// Problem: Minimum Rectangles to Cover Points
// 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 minRectanglesToCoverPoints(points [][]int, w int) int {
```

```
}
```

### Kotlin Solution:

```
class Solution {
fun minRectanglesToCoverPoints(points: Array<IntArray>, w: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func minRectanglesToCoverPoints(_ points: [[Int]], _ w: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Minimum Rectangles to Cover Points
// 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 min_rectangles_to_cover_points(points: Vec<Vec<i32>>, w: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} points
# @param {Integer} w
# @return {Integer}
def min_rectangles_to_cover_points(points, w)
```

```
end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $w
* @return Integer
*/
function minRectanglesToCoverPoints($points, $w) {

}
}
```

**Dart Solution:**

```
class Solution {
int minRectanglesToCoverPoints(List<List<int>> points, int w) {

}
}
```

**Scala Solution:**

```
object Solution {
def minRectanglesToCoverPoints(points: Array[Array[Int]], w: Int): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec min_rectangles_to_cover_points(points :: [[integer]], w :: integer) ::
integer
def min_rectangles_to_cover_points(points, w) do

end
```

```
end
```

### Erlang Solution:

```
-spec min_rectangles_to_cover_points(Points :: [[integer()]], W :: integer())
-> integer().
min_rectangles_to_cover_points(Points, W) ->
.
```

### Racket Solution:

```
(define/contract (min-rectangles-to-cover-points points w)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```