Problem 3380: Maximum Area Rectangle With Point Constraints I

Problem Information

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

Problem Description

You are given an array

points

where

points[i] = [x

i

, y

i

]

represents the coordinates of a point on an infinite plane.

Your task is to find the

maximum

area of a rectangle that:

Can be formed using

four

of these points as its corners.

Does

not

contain any other point inside or on its border.

Has its edges

parallel

to the axes.

Return the

maximum area

that you can obtain or -1 if no such rectangle is possible.

Example 1:

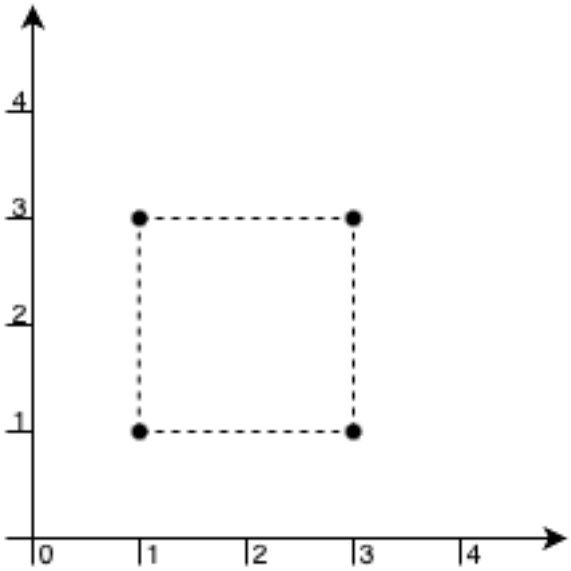
Input:

points = [[1,1],[1,3],[3,1],[3,3]]

Output:

4

Explanation:



We can make a rectangle with these 4 points as corners and there is no other point that lies inside or on the border

. Hence, the maximum possible area would be 4.

Example 2:

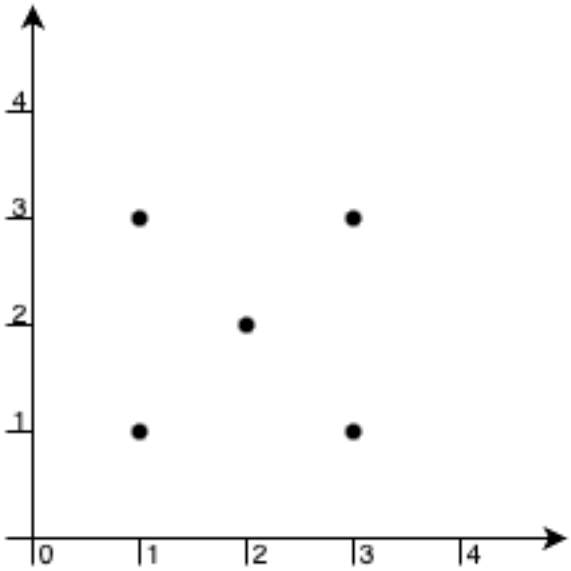
Input:

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

Output:

-1

Explanation:



There is only one rectangle possible is with points

[1,1], [1,3], [3,1]

and

[3,3]

but

[2,2]

will always lie inside it. Hence, returning -1.

Example 3:

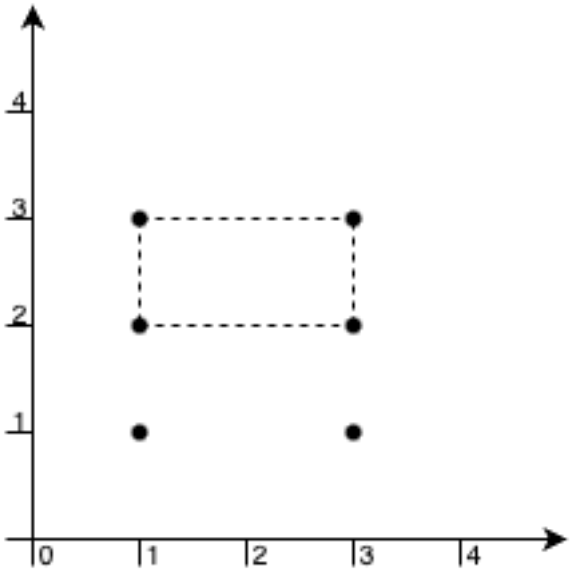
Input:

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

Output:

2

Explanation:



The maximum area rectangle is formed by the points

[1,3], [1,2], [3,2], [3,3]

, which has an area of 2. Additionally, the points

[1,1], [1,2], [3,1], [3,2]

also form a valid rectangle with the same area.

Constraints:

1 <= points.length <= 10

points[i].length == 2

0 <= x

i

, y

i

<= 100

All the given points are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-rectangle-area points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int maxRectangleArea(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Area Rectangle With Point Constraints I
Difficulty: Medium
Tags: array, tree, math, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def maxRectangleArea(self, points: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

function maxRectangleArea(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MaxRectangleArea(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Area Rectangle With Point Constraints I
* Difficulty: Medium
* Tags: array, tree, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int maxRectangleArea(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Area Rectangle With Point Constraints I
// Difficulty: Medium
// Tags: array, tree, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxRectangleArea(points [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Area Rectangle With Point Constraints I
// Difficulty: Medium
// Tags: array, tree, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-rectangle-area points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```