Problem 939: Minimum Area Rectangle

Problem Information

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

Problem Description

You are given an array of points in the

X-Y

plane

points

where

points[i] = [x

i

, y

i

]

.

Return

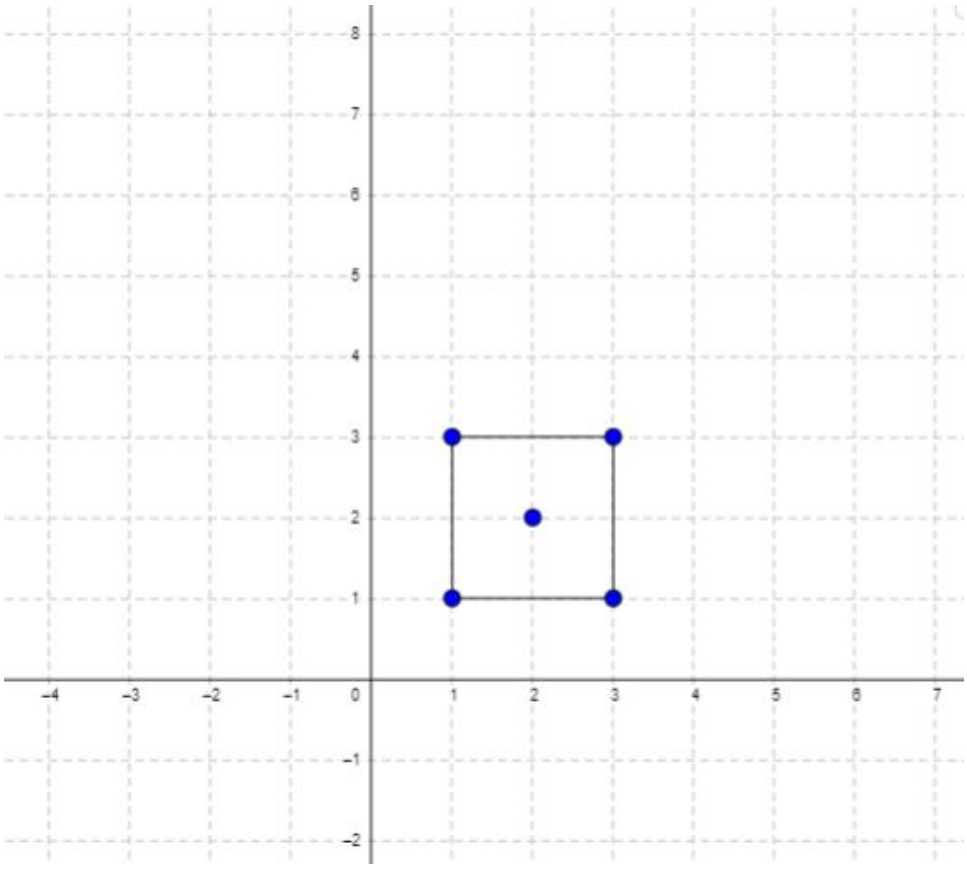
the minimum area of a rectangle formed from these points, with sides parallel to the X and Y axes

. If there is not any such rectangle, return

0

.

Example 1:



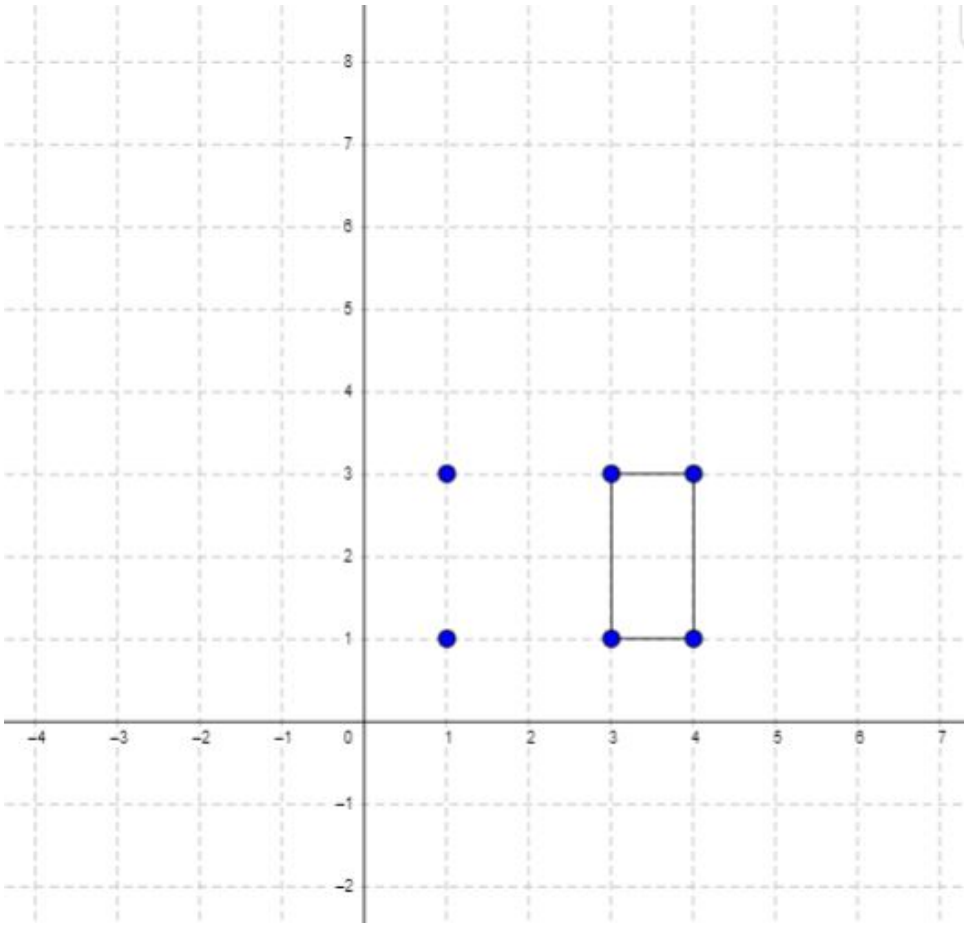
Input:

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

Output:

4

Example 2:



Input:

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

Output:

2

Constraints:

1 <= points.length <= 500

points[i].length == 2

0 <= x

i

, y

i

<= 4 * 10

4

All the given points are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-area-rect points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Area Rectangle
Difficulty: Medium
Tags: array, math, hash, sort
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Area Rectangle
* Difficulty: Medium
* Tags: array, math, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

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

}
```

Go Solution:

```
// Problem: Minimum Area Rectangle
// Difficulty: Medium
// Tags: array, math, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Area Rectangle
// Difficulty: Medium
// Tags: array, math, hash, sort
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-area-rect points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```