Problem 963: Minimum Area Rectangle II

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 any rectangle formed from these points, with sides

not necessarily parallel

to the X and Y axes

. If there is not any such rectangle, return

0

.

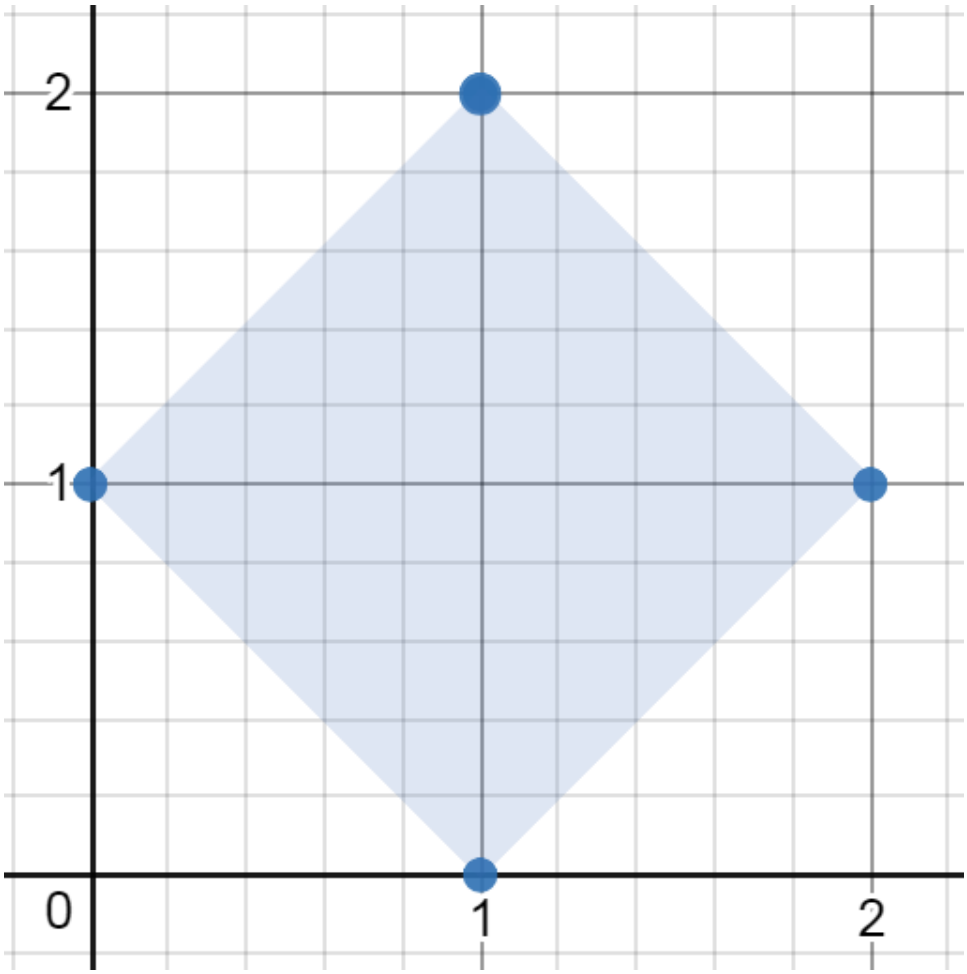
Answers within

10

-5

of the actual answer will be accepted.

Example 1:



Input:

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

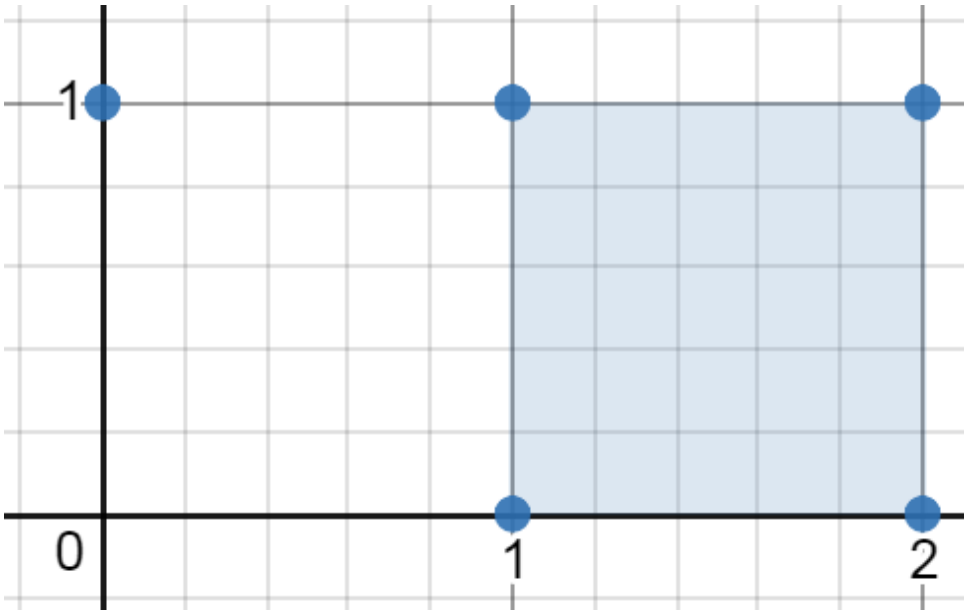
Output:

2.00000

Explanation:

The minimum area rectangle occurs at [1,2],[2,1],[1,0],[0,1], with an area of 2.

Example 2:



Input:

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

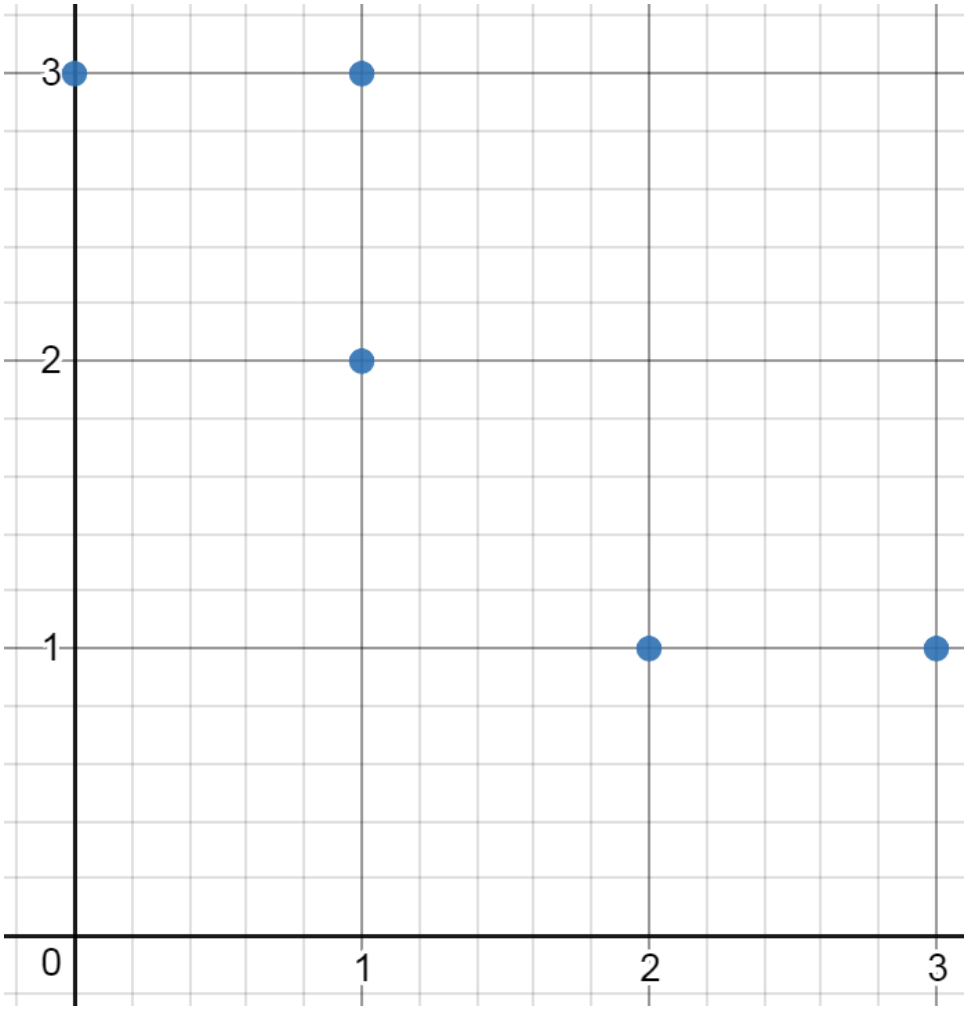
Output:

1.00000

Explanation:

The minimum area rectangle occurs at [1,0],[1,1],[2,1],[2,0], with an area of 1.

Example 3:



Input:

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

Output:

0

Explanation:

There is no possible rectangle to form from these points.

Constraints:

1 <= points.length <= 50

points[i].length == 2

0 <= x

i

, y

i

<= 4 * 10

4

All the given points are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public double minAreaFreeRect(int[][] points) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public double MinAreaFreeRect(int[][] points) {

}
}
```

C:

```
double minAreaFreeRect(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func minAreaFreeRect(points [][]int) float64 {

}
```

Kotlin:

```
class Solution {
fun minAreaFreeRect(points: Array<IntArray>): Double {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_area_free_rect(points: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
*/
function minAreaFreeRect($points) {

}
}
```

Dart:

```
class Solution {
double minAreaFreeRect(List<List<int>> points) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
* Tags: array, math, hash
*
* 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:
double minAreaFreeRect(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 double minAreaFreeRect(int[][] points) {

}
}
```

Python3 Solution:

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 minAreaFreeRect = function(points) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 minAreaFreeRect(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 double MinAreaFreeRect(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Area Rectangle II
* Difficulty: Medium
```

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

double minAreaFreeRect(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Area Rectangle II
// Difficulty: Medium
// Tags: array, math, hash
//
// 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 minAreaFreeRect(points [][]int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun minAreaFreeRect(points: Array<IntArray>): Double {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Area Rectangle II
// Difficulty: Medium
// Tags: array, math, hash
//
// 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_free_rect(points: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
double minAreaFreeRect(List<List<int>> points) {
```

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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