# Problem 3047: Find the Largest Area of Square Inside Two Rectangles

## Problem Information

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

## Problem Description

There exist

n

rectangles in a 2D plane with edges parallel to the x and y axis. You are given two 2D integer arrays

bottomLeft

and

topRight

where

bottomLeft[i] = [a_i, b_i]

and

topRight[i] = [c_i, d_i]

represent the

bottom-left

and

top-right

coordinates of the

i

th

rectangle, respectively.

You need to find the

maximum

area of a

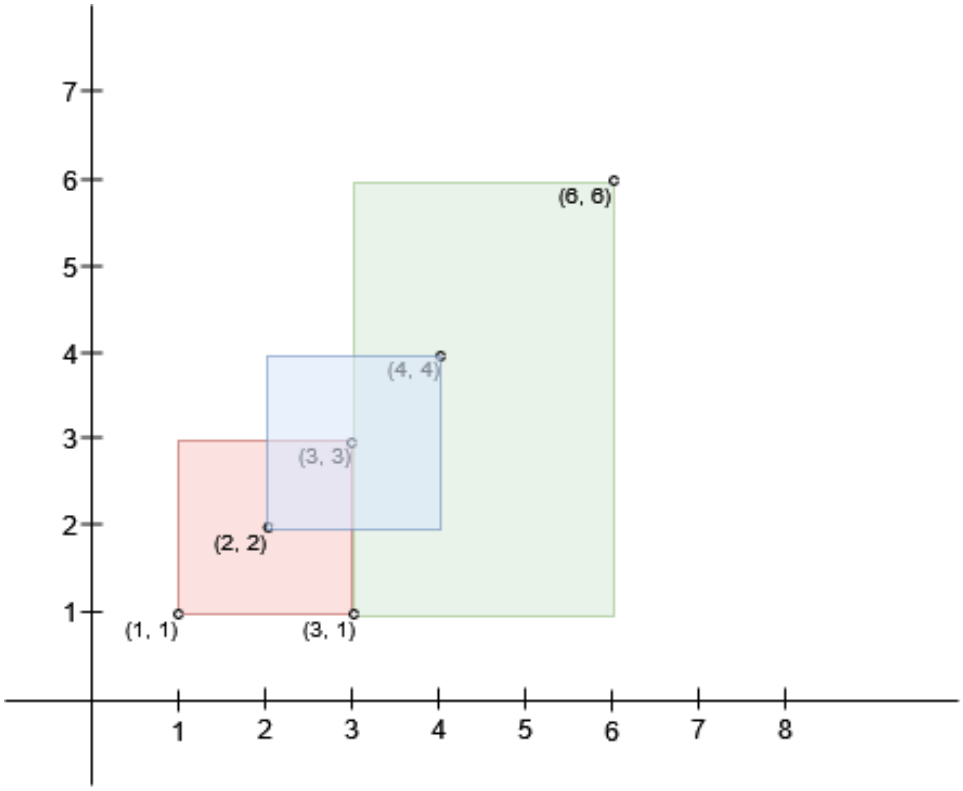
square

that can fit inside the intersecting region of at least two rectangles. Return

0

if such a square does not exist.

Example 1:




Input:

bottomLeft = [[1,1],[2,2],[3,1]], topRight = [[3,3],[4,4],[6,6]]

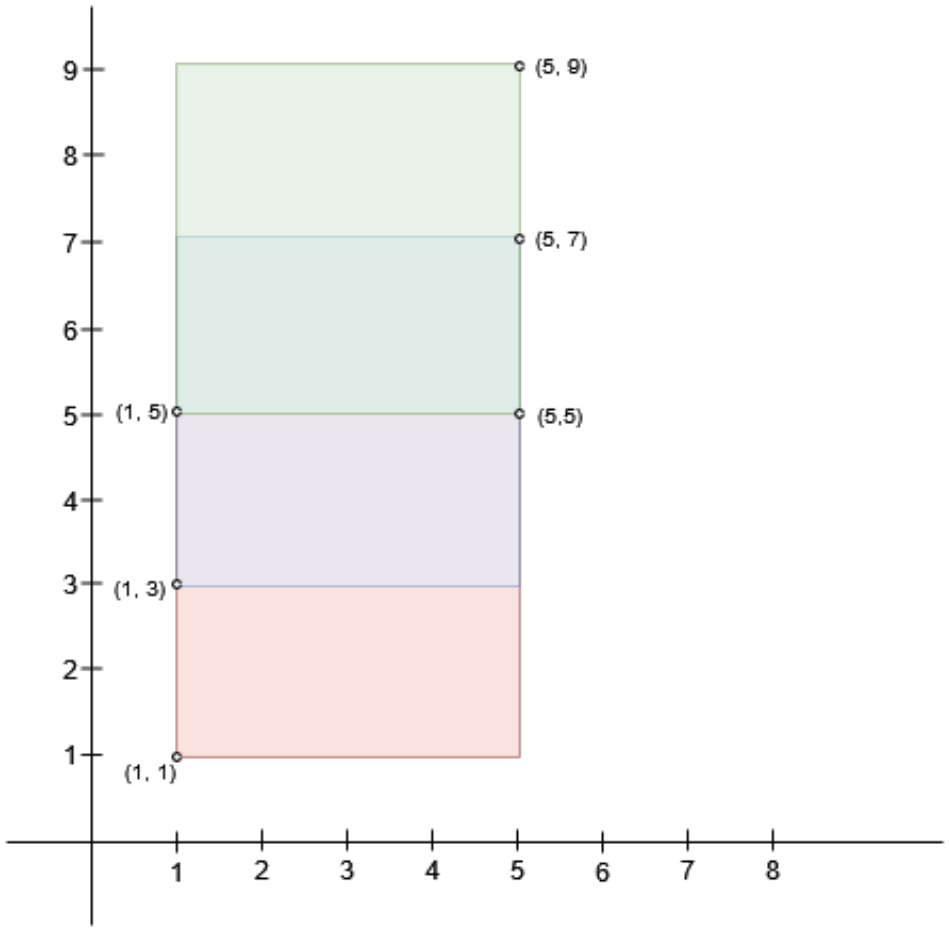
Output:

1

Explanation:

A square with side length 1 can fit inside either the intersecting region of rectangles 0 and 1 or the intersecting region of rectangles 1 and 2. Hence the maximum area is 1. It can be shown that a square with a greater side length can not fit inside any intersecting region of two rectangles.

Example 2:




Input:

bottomLeft = [[1,1],[1,3],[1,5]], topRight = [[5,5],[5,7],[5,9]]

Output:

4

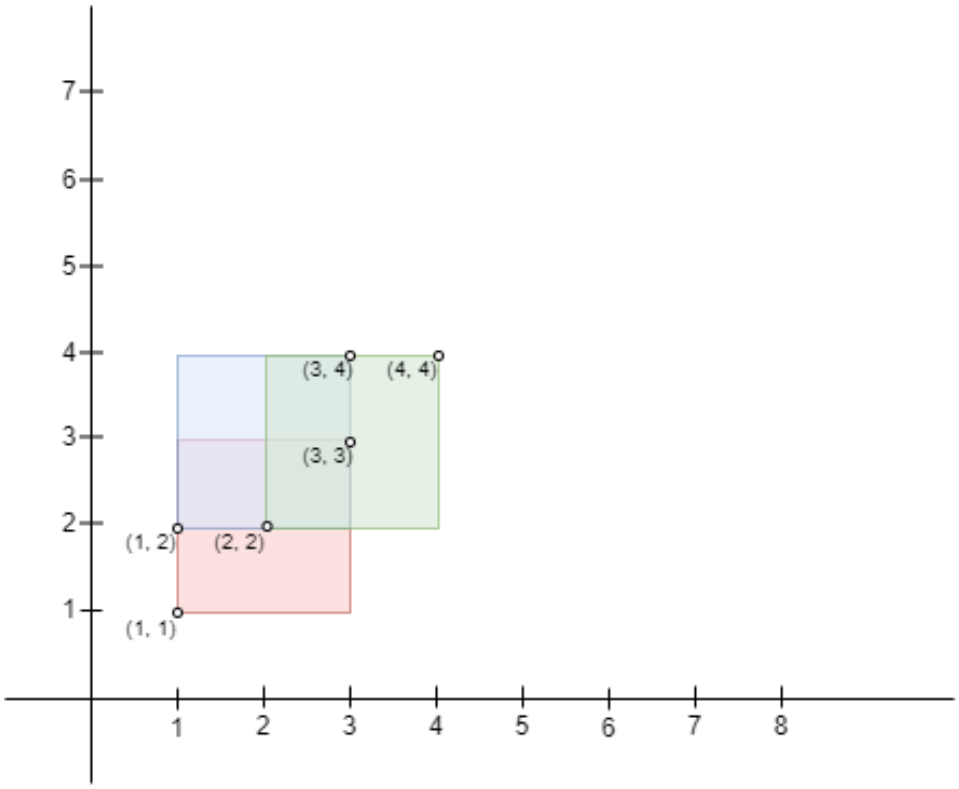
Explanation:

A square with side length 2 can fit inside either the intersecting region of rectangles 0 and 1 or the intersecting region of rectangles 1 and 2. Hence the maximum area is

2 * 2 = 4

. It can be shown that a square with a greater side length can not fit inside any intersecting region of two rectangles.

Example 3:




Input:

bottomLeft = [[1,1],[2,2],[1,2]], topRight = [[3,3],[4,4],[3,4]]

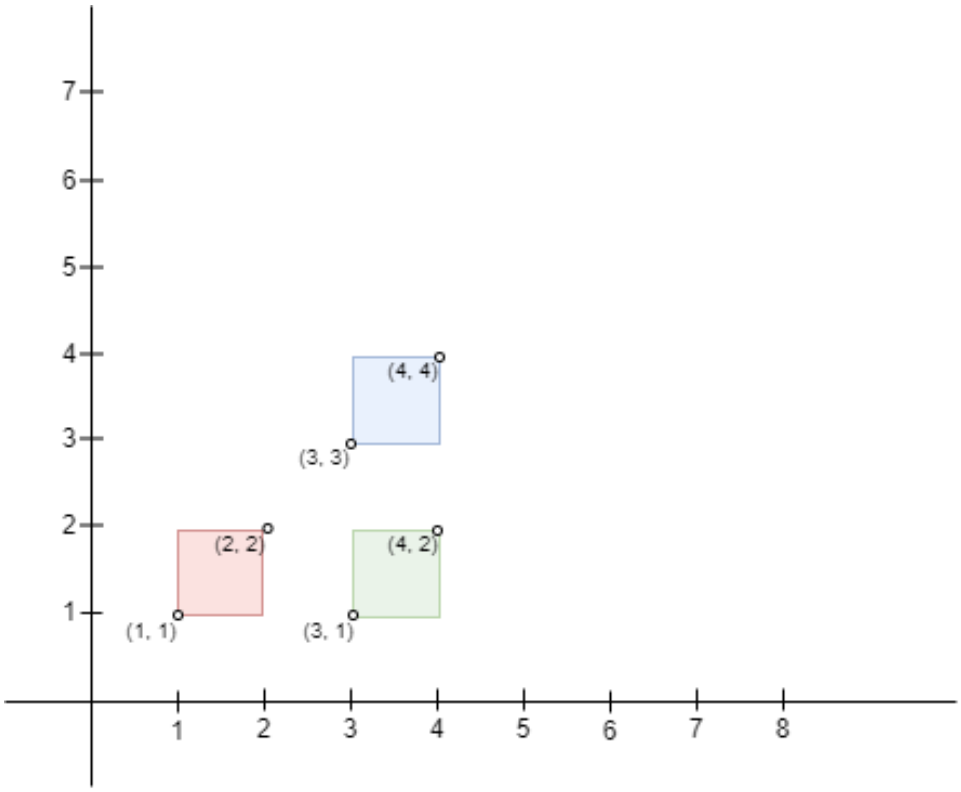
Output:

1

Explanation:

A square with side length 1 can fit inside the intersecting region of any two rectangles. Also, no larger square can, so the maximum area is 1. Note that the region can be formed by the intersection of more than 2 rectangles.

Example 4:




Input:

bottomLeft = [[1,1],[3,3],[3,1]], topRight = [[2,2],[4,4],[4,2]]

Output:

0

Explanation:

No pair of rectangles intersect, hence, the answer is 0.

Constraints:

n == bottomLeft.length == topRight.length

2 <= n <= 10

3

bottomLeft[i].length == topRight[i].length == 2

1 <= bottomLeft[i][0], bottomLeft[i][1] <= 10

7

1 <= topRight[i][0], topRight[i][1] <= 10

7

bottomLeft[i][0] < topRight[i][0]

bottomLeft[i][1] < topRight[i][1]

## Code Snippets

**C++:**

```
class Solution {
public:
long long largestSquareArea(vector<vector<int>>& bottomLeft,
vector<vector<int>>& topRight) {

}
};
```

**Java:**

```
class Solution {
public long largestSquareArea(int[][] bottomLeft, int[][] topRight) {

}
}
```

**Python3:**

```
class Solution:
def largestSquareArea(self, bottomLeft: List[List[int]], topRight:
List[List[int]]) -> int:
```

**Python:**

```
class Solution(object):
def largestSquareArea(self, bottomLeft, topRight):
"""
:type bottomLeft: List[List[int]]
:type topRight: List[List[int]]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[][]} bottomLeft
* @param {number[][]} topRight
* @return {number}
*/
var largestSquareArea = function(bottomLeft, topRight) {

};
```

**TypeScript:**

```
function largestSquareArea(bottomLeft: number[][], topRight: number[][]):
number {

};
```

**C#:**

```
public class Solution {
public long LargestSquareArea(int[][] bottomLeft, int[][] topRight) {

}
}
```

**C:**

```
long long largestSquareArea(int** bottomLeft, int bottomLeftSize, int*
bottomLeftColSize, int** topRight, int topRightSize, int* topRightColSize) {

}
```

**Go:**

```
func largestSquareArea(bottomLeft [][]int, topRight [][]int) int64 {

}
```

**Kotlin:**

```
class Solution {
fun largestSquareArea(bottomLeft: Array<IntArray>, topRight:
Array<IntArray>): Long {

}
}
```

**Swift:**

```
class Solution {
func largestSquareArea(_ bottomLeft: [[Int]], _ topRight: [[Int]]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn largest_square_area(bottom_left: Vec<Vec<i32>>, top_right:
Vec<Vec<i32>>) -> i64 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} bottom_left
# @param {Integer[][]} top_right
# @return {Integer}
def largest_square_area(bottom_left, top_right)

end
```

**PHP:**

```
class Solution {
```

```
/**
* @param Integer[][] $bottomLeft
* @param Integer[][] $topRight
* @return Integer
*/
function largestSquareArea($bottomLeft, $topRight) {

}
}
```

**Dart:**

```
class Solution {
int largestSquareArea(List<List<int>> bottomLeft, List<List<int>> topRight) {

}
}
```

**Scala:**

```
object Solution {
def largestSquareArea(bottomLeft: Array[Array[Int]], topRight:
Array[Array[Int]]): Long = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec largest_square_area(bottom_left :: [[integer]], top_right ::
[[integer]]) :: integer
def largest_square_area(bottom_left, top_right) do

end
end
```

**Erlang:**

```
-spec largest_square_area(BottomLeft :: [[integer()]], TopRight ::
[[integer()]]) -> integer().
largest_square_area(BottomLeft, TopRight) ->
```

```
.
```

**Racket:**

```
(define/contract (largest-square-area bottomLeft topRight)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Find the Largest Area of Square Inside Two Rectangles
* Difficulty: Medium
* Tags: array, math
*
* 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:
long long largestSquareArea(vector<vector<int>>& bottomLeft,
vector<vector<int>>& topRight) {

}
};
```

### Java Solution:

```
/**
* Problem: Find the Largest Area of Square Inside Two Rectangles
* Difficulty: Medium
* Tags: array, math
*
* 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 long largestSquareArea(int[][] bottomLeft, int[][] topRight) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find the Largest Area of Square Inside Two Rectangles
Difficulty: Medium
Tags: array, math

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 largestSquareArea(self, bottomLeft: List[List[int]], topRight:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def largestSquareArea(self, bottomLeft, topRight):
"""
:type bottomLeft: List[List[int]]
:type topRight: List[List[int]]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Find the Largest Area of Square Inside Two Rectangles
```

```
* Difficulty: Medium
* Tags: array, math
*
* 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[][]} bottomLeft
* @param {number[][]} topRight
* @return {number}
*/
var largestSquareArea = function(bottomLeft, topRight) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Find the Largest Area of Square Inside Two Rectangles
* Difficulty: Medium
* Tags: array, math
*
* 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 largestSquareArea(bottomLeft: number[][], topRight: number[][]):
number {

};
```

**C# Solution:**

```
/*
* Problem: Find the Largest Area of Square Inside Two Rectangles
* Difficulty: Medium
* Tags: array, math
*
```

```
* 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 long LargestSquareArea(int[][] bottomLeft, int[][] topRight) {

}
}
```

### C Solution:

```
/*
* Problem: Find the Largest Area of Square Inside Two Rectangles
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

long long largestSquareArea(int** bottomLeft, int bottomLeftSize, int*
bottomLeftColSize, int** topRight, int topRightSize, int* topRightColSize) {

}
```

### Go Solution:

```
// Problem: Find the Largest Area of Square Inside Two Rectangles
// Difficulty: Medium
// Tags: array, math
//
// 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 largestSquareArea(bottomLeft [][]int, topRight [][]int) int64 {

}
```

### Kotlin Solution:

```
class Solution {
fun largestSquareArea(bottomLeft: Array<IntArray>, topRight:
Array<IntArray>): Long {

}
}
```

### Swift Solution:

```
class Solution {
func largestSquareArea(_ bottomLeft: [[Int]], _ topRight: [[Int]]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Find the Largest Area of Square Inside Two Rectangles
// Difficulty: Medium
// Tags: array, math
//
// 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 largest_square_area(bottom_left: Vec<Vec<i32>>, top_right:
Vec<Vec<i32>>) -> i64 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} bottom_left
# @param {Integer[][]} top_right
# @return {Integer}
def largest_square_area(bottom_left, top_right)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[][] $bottomLeft
* @param Integer[][] $topRight
* @return Integer
*/
function largestSquareArea($bottomLeft, $topRight) {

}
}
```

**Dart Solution:**

```
class Solution {
int largestSquareArea(List<List<int>> bottomLeft, List<List<int>> topRight) {

}
}
```

**Scala Solution:**

```
object Solution {
def largestSquareArea(bottomLeft: Array[Array[Int]], topRight:
Array[Array[Int]]): Long = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec largest_square_area(bottom_left :: [[integer]], top_right ::
[[integer]]) :: integer
def largest_square_area(bottom_left, top_right) do

end
end
```

### Erlang Solution:

```
-spec largest_square_area(BottomLeft :: [[integer()]], TopRight ::
[[integer()]]) -> integer().
largest_square_area(BottomLeft, TopRight) ->
.
```

### Racket Solution:

```
(define/contract (largest-square-area bottomLeft topRight)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```