# Problem 3454: Separate Squares II

## Problem Information

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

## Problem Description

You are given a 2D integer array

squares

. Each

squares[i] = [x

i

, y

i

, l

i

]

represents the coordinates of the bottom-left point and the side length of a square parallel to the x-axis.

Find the

minimum

y-coordinate value of a horizontal line such that the total area covered by squares above the line

equals

the total area covered by squares below the line.

Answers within

10

-5

of the actual answer will be accepted.

Note

: Squares

may

overlap. Overlapping areas should be counted

only once

in this version.

Example 1:

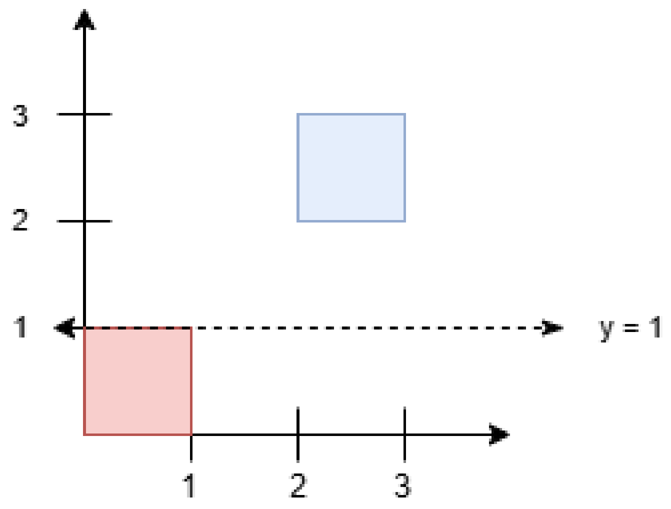
Input:

squares = [[0,0,1],[2,2,1]]

Output:

1.00000

Explanation:



Any horizontal line between

y = 1

and

y = 2

results in an equal split, with 1 square unit above and 1 square unit below. The minimum y-value is 1.

Example 2:

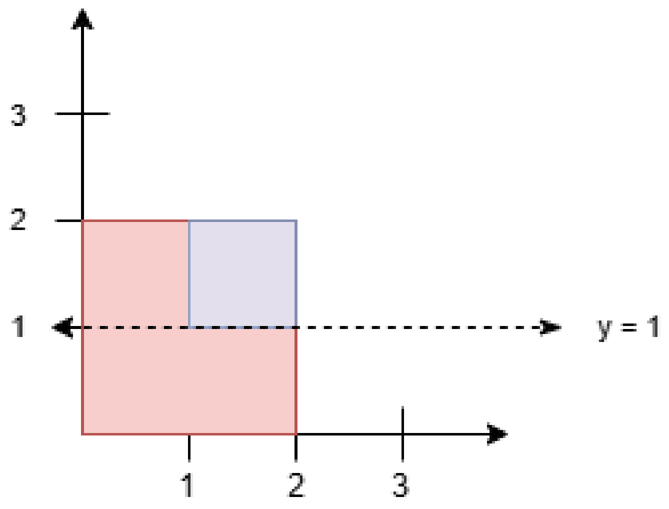
Input:

squares = [[0,0,2],[1,1,1]]

Output:

1.00000

Explanation:



Since the blue square overlaps with the red square, it will not be counted again. Thus, the line

y = 1

splits the squares into two equal parts.

Constraints:

1 <= squares.length <= 5 * 10

4

squares[i] = [x

i

, y

i

, l

i

]

squares[i].length == 3

0 <= x

i

, y

i

<= 10

9

1 <= l

i

<= 10

9

The total area of all the squares will not exceed

10

15

.

## Code Snippets

**C++:**

```
class Solution {
public:
double separateSquares(vector<vector<int>>& squares) {
}
};
```

**Java:**

```
class Solution {
public double separateSquares(int[][] squares) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function separateSquares(squares: number[][]): number {

};
```

**C#:**

}

```
public class Solution {
public double SeparateSquares(int[][] squares) {

}
}
```

**C:**

```
double separateSquares(int** squares, int squaresSize, int* squaresColSize) {

}
```

**Go:**

```
func separateSquares(squares [][]int) float64 {

}
```

**Kotlin:**

```
class Solution {
fun separateSquares(squares: Array<IntArray>): Double {

}
}
```

**Swift:**

```
class Solution {
func separateSquares(_ squares: [[Int]]) -> Double {

}
}
```

**Rust:**

```
impl Solution {
pub fn separate_squares(squares: Vec<Vec<i32>>) -> f64 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} squares
# @return {Float}
def separate_squares(squares)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $squares
* @return Float
*/
function separateSquares($squares) {

}
}
```

**Dart:**

```
class Solution {
double separateSquares(List<List<int>> squares) {

}
}
```

**Scala:**

```
object Solution {
def separateSquares(squares: Array[Array[Int]]): Double = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec separate_squares(squares :: [[integer]]) :: float
def separate_squares(squares) do

end
end
```

### Erlang:

```
-spec separate_squares(Squares :: [[integer()]]) -> float().
separate_squares(Squares) ->
.
```

### Racket:

```
(define/contract (separate-squares squares)
(-> (listof (listof exact-integer?)) flonum?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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:
double separateSquares(vector<vector<int>>& squares) {

}
};
```

### Java Solution:

```
/**
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 double separateSquares(int[][] squares) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Separate Squares II
Difficulty: Hard
Tags: array, tree, search

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

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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[][]} squares
* @return {number}
*/
var separateSquares = function(squares) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 separateSquares(squares: number[][]): number {

};
```

**C# Solution:**

```
/*
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 double SeparateSquares(int[][] squares) {

}
}
```

**C Solution:**

```
/*
* Problem: Separate Squares II
* Difficulty: Hard
* Tags: array, tree, search
*
* 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
*/

double separateSquares(int** squares, int squaresSize, int* squaresColSize) {

}
```

**Go Solution:**

```
// Problem: Separate Squares II
// Difficulty: Hard
// Tags: array, tree, search
//
// 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 separateSquares(squares [][]int) float64 {

}
```

**Kotlin Solution:**

```
class Solution {
fun separateSquares(squares: Array<IntArray>): Double {

}
}
```

**Swift Solution:**

```
class Solution {
func separateSquares(_ squares: [[Int]]) -> Double {

}
}
```

**Rust Solution:**

```
// Problem: Separate Squares II
// Difficulty: Hard
// Tags: array, tree, search
//
// 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 separate_squares(squares: Vec<Vec<i32>>) -> f64 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[][]} squares
# @return {Float}
def separate_squares(squares)

end
```

**PHP Solution:**

```
class Solution {
```

```
/**
* @param Integer[][] $squares
* @return Float
*/
function separateSquares($squares) {

}
}
```

**Dart Solution:**

```
class Solution {
double separateSquares(List<List<int>> squares) {

}
}
```

**Scala Solution:**

```
object Solution {
def separateSquares(squares: Array[Array[Int]]): Double = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec separate_squares(squares :: [[integer]]) :: float
def separate_squares(squares) do

end
end
```

**Erlang Solution:**

```
-spec separate_squares(Squares :: [[integer()]]) -> float().
separate_squares(Squares) ->
.
```

**Racket Solution:**

```
(define/contract (separate-squares squares)
(-> (listof (listof exact-integer?)) flonum?)
)
```