Problem 1637: Widest Vertical Area Between Two Points Containing No Points

Problem Information

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

Problem Description

Given

n

points

on a 2D plane where

points[i] = [x

i

, y

i

]

, Return

the

widest vertical area

between two points such that no points are inside the area.

A

vertical area

is an area of fixed-width extending infinitely along the y-axis (i.e., infinite height). The

widest vertical area

is the one with the maximum width.

Note that points

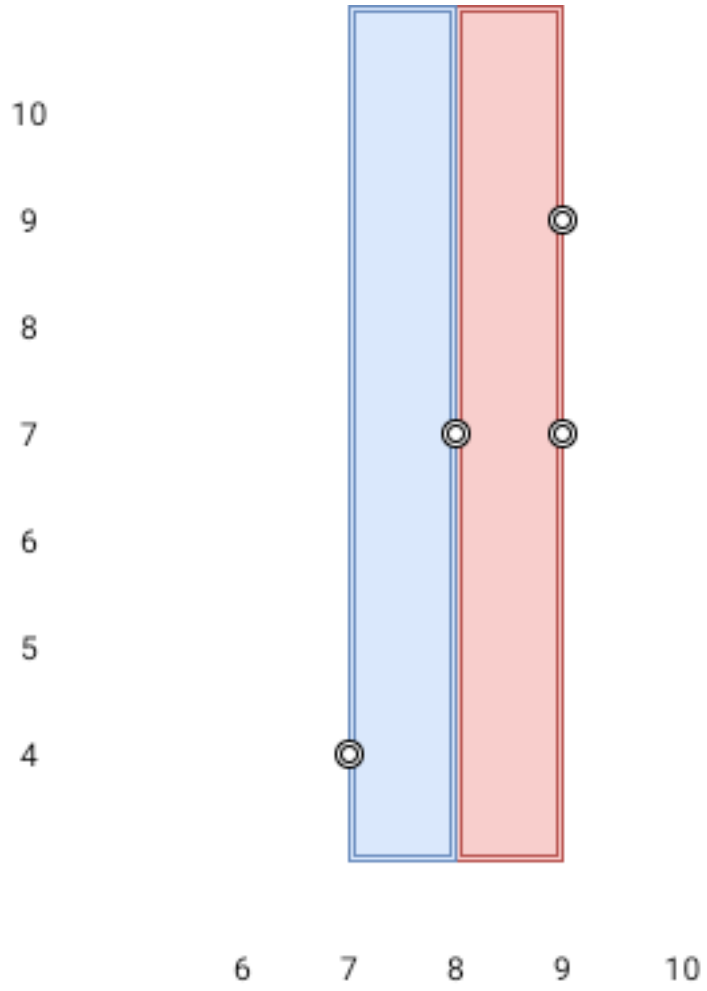
on the edge

of a vertical area

are not

considered included in the area.

Example 1:



Input:

points = [[8,7],[9,9],[7,4],[9,7]]

Output:

1

Explanation:

Both the red and the blue area are optimal.

Example 2:

Input:

points = [[3,1],[9,0],[1,0],[1,4],[5,3],[8,8]]

Output:

3

Constraints:

n == points.length

2 <= n <= 10

5

points[i].length == 2

0 <= x

i

, y

i

<= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-width-of-vertical-area points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
* Tags: array, sort
*
* 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:
int maxWidthOfVerticalArea(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
* Tags: array, sort
*
* 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 int maxWidthOfVerticalArea(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Widest Vertical Area Between Two Points Containing No Points
Difficulty: Easy
Tags: array, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
```

```
* Tags: array, sort
*
* 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[][]} points
* @return {number}
*/
var maxWidthOfVerticalArea = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
* Tags: array, sort
*
* 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 maxWidthOfVerticalArea(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
* Tags: array, sort
*
* 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 int MaxWidthOfVerticalArea(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Widest Vertical Area Between Two Points Containing No Points
* Difficulty: Easy
* Tags: array, sort
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Widest Vertical Area Between Two Points Containing No Points
// Difficulty: Easy
// Tags: array, sort
//
// 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 maxWidthOfVerticalArea(points [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Widest Vertical Area Between Two Points Containing No Points
// Difficulty: Easy
// Tags: array, sort
//
// 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 max_width_of_vertical_area(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-width-of-vertical-area points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```