Problem 2152: Minimum Number of Lines to Cover Points

Problem Information

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

Problem Description

You are given an array

points

where

points[i] = [x

i

, y

i

]

represents a point on an

X-Y

plane.

Straight lines

are going to be added to the

X-Y

plane, such that every point is covered by at

least

one line.

Return

the

minimum

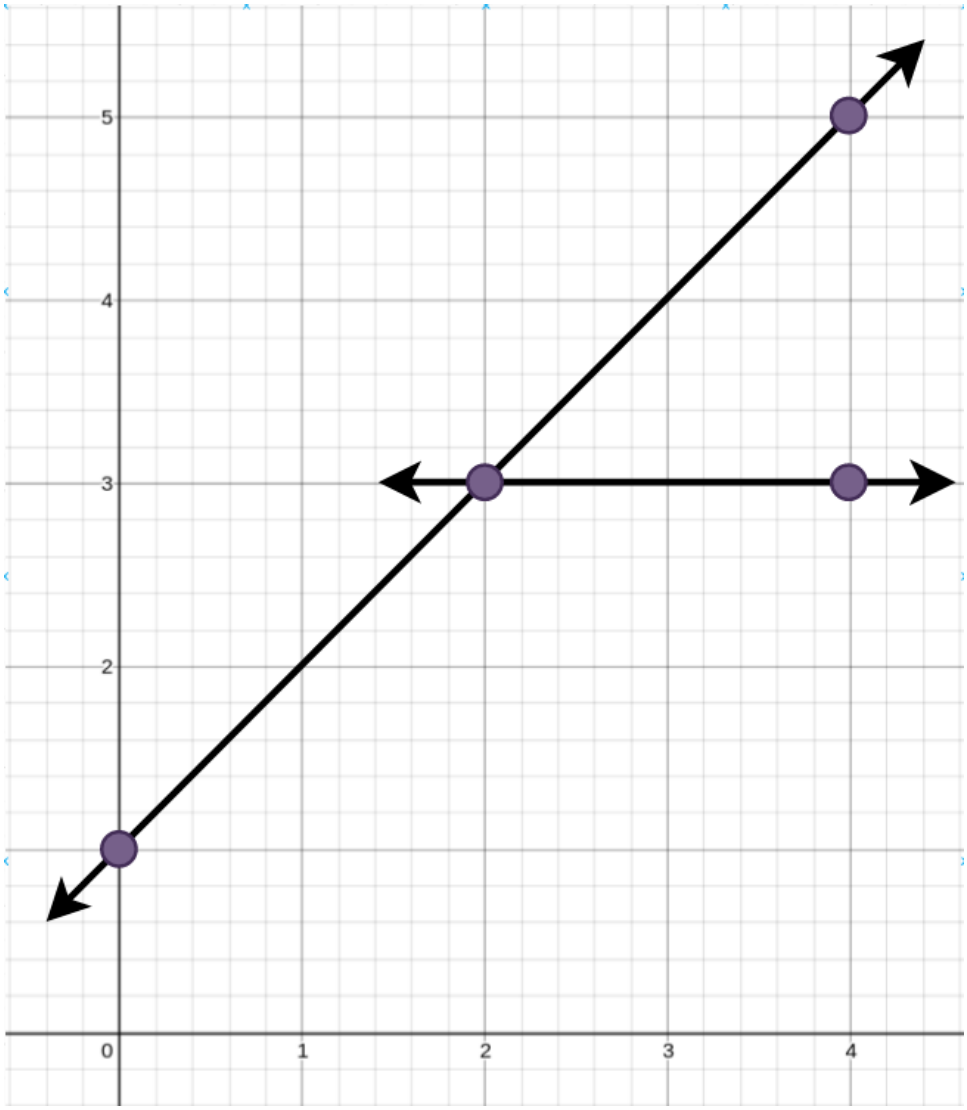
number of

straight lines

needed to cover all the points

.

Example 1:



Input:

points = [[0,1],[2,3],[4,5],[4,3]]

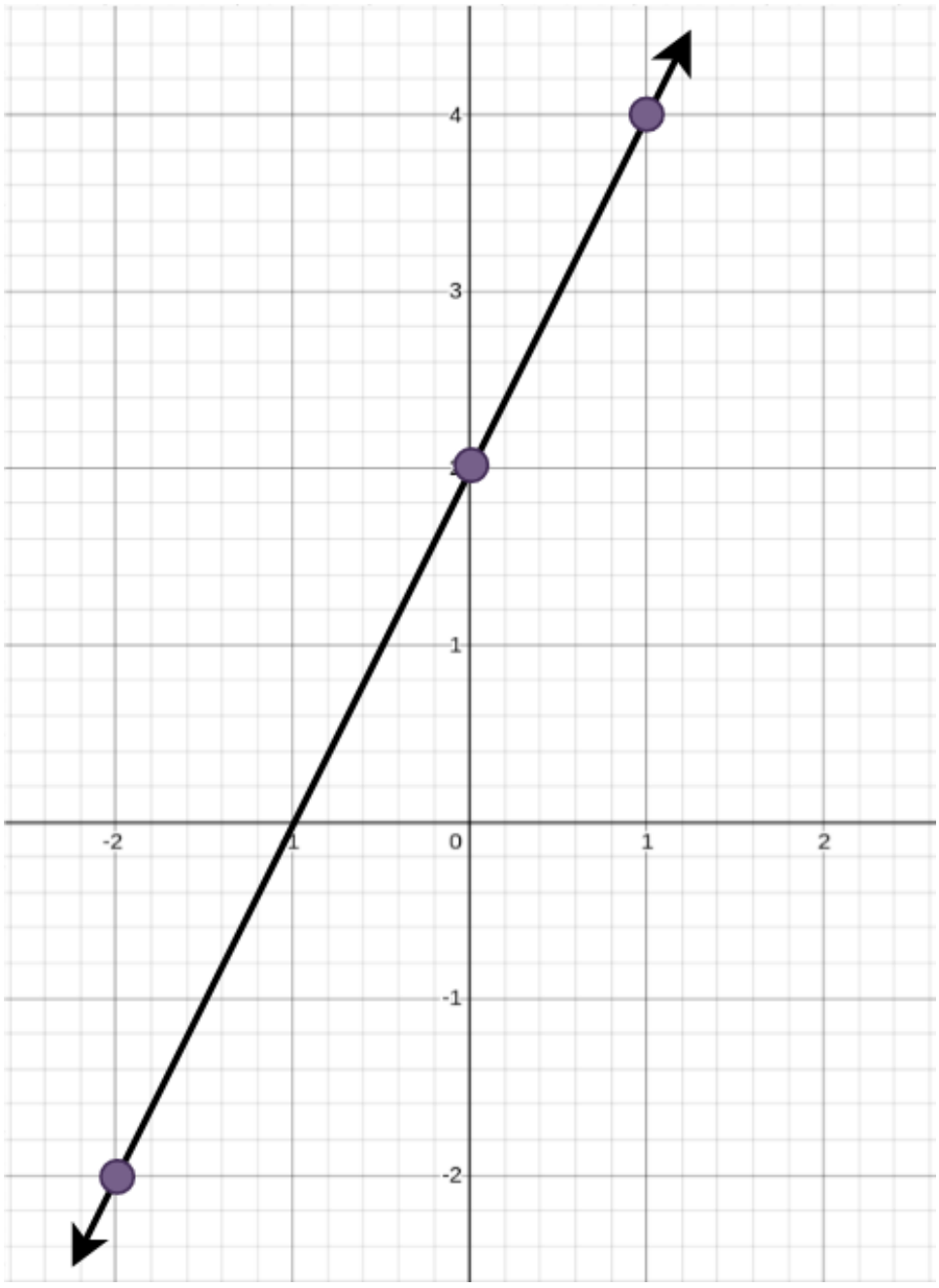
Output:

2

Explanation:

The minimum number of straight lines needed is two. One possible solution is to add: - One line connecting the point at (0, 1) to the point at (4, 5). - Another line connecting the point at (2, 3) to the point at (4, 3).

Example 2:



Input:

points = [[0,2],[-2,-2],[1,4]]

Output:

1

Explanation:

The minimum number of straight lines needed is one. The only solution is to add: - One line connecting the point at (-2, -2) to the point at (1, 4).

Constraints:

1 <= points.length <= 10

points[i].length == 2

-100 <= x

i

, y

i

<= 100

All the

points

are

unique

.

Code Snippets

C++:

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

}
```

```
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-lines points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
* Tags: array, dp, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
* Tags: array, dp, math, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Lines to Cover Points
Difficulty: Medium
Tags: array, dp, math, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
```

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

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
* Tags: array, dp, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
* Tags: array, dp, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Lines to Cover Points
* Difficulty: Medium
* Tags: array, dp, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Minimum Number of Lines to Cover Points
// Difficulty: Medium
// Tags: array, dp, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Lines to Cover Points
// Difficulty: Medium
// Tags: array, dp, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-lines points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```