Problem 1037: Valid Boomerang

Problem Information

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

Problem Description

Given an array

points

where

points[i] = [x

i

, y

i

]

represents a point on the

X-Y

plane, return

true

if these points are a

boomerang

.

A

boomerang

is a set of three points that are

all distinct

and

not in a straight line

.

Example 1:

Input:

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

Output:

true

Example 2:

Input:

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

Output:

false

Constraints:

points.length == 3

points[i].length == 2

0 <= x

i

, y

i

<= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean isBoomerang(int[][] points) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function isBoomerang(points: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool IsBoomerang(int[][] points) {

}
}
```

C:

```
bool isBoomerang(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func isBoomerang(points [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun isBoomerang(points: Array<IntArray>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Dart:

```
class Solution {
bool isBoomerang(List<List<int>> points) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-boomerang points)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Boomerang
* Difficulty: Easy
* 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:
bool isBoomerang(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Boomerang
* Difficulty: Easy
* 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 boolean isBoomerang(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Boomerang
Difficulty: Easy
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 isBoomerang(self, points: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Valid Boomerang
* Difficulty: Easy
* 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[][]} points
* @return {boolean}
*/
var isBoomerang = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Boomerang
* Difficulty: Easy
* 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 isBoomerang(points: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Boomerang
* Difficulty: Easy
* 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 bool IsBoomerang(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Valid Boomerang
* Difficulty: Easy
* 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
```

```
*/

bool isBoomerang(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Valid Boomerang
// Difficulty: Easy
// 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 isBoomerang(points [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isBoomerang(points: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Valid Boomerang
// Difficulty: Easy
// 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 is_boomerang(points: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool isBoomerang(List<List<int>> points) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-boomerang points)
(-> (listof (listof exact-integer?)) boolean?)
)
```