Problem 812: Largest Triangle Area

Problem Information

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

Problem Description

Given an array of points on the

X-Y

plane

points

where

points[i] = [x

i

, y

i

]

, return

the area of the largest triangle that can be formed by any three different points

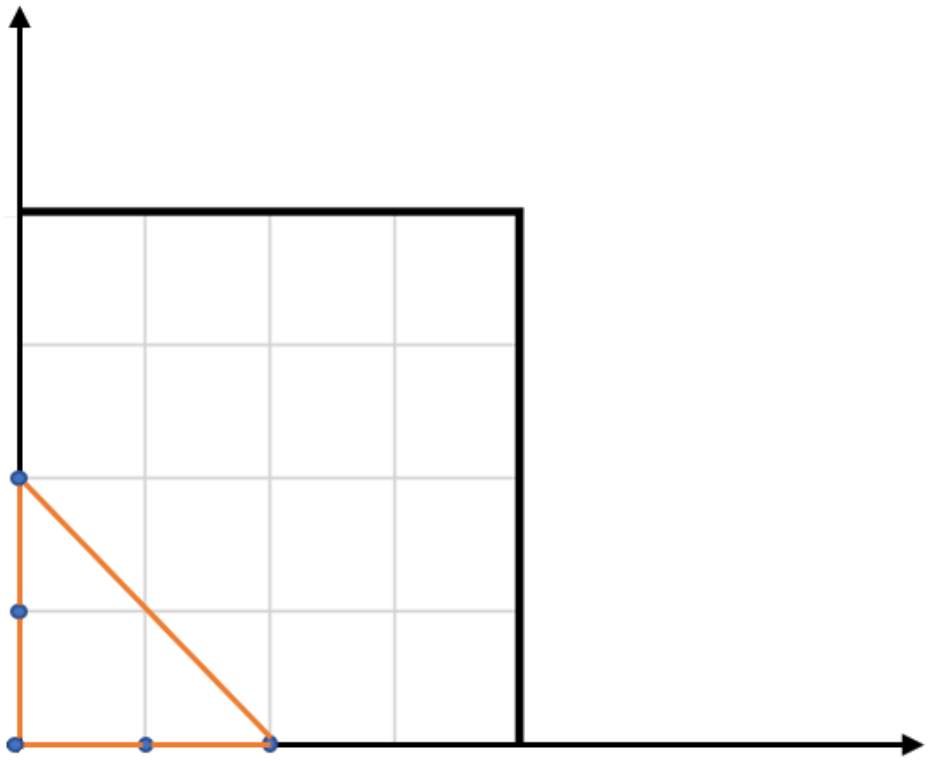
. Answers within

10

-5

of the actual answer will be accepted.

Example 1:



Input:

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

Output:

2.00000

Explanation:

The five points are shown in the above figure. The red triangle is the largest.

Example 2:

Input:

points = [[1,0],[0,0],[0,1]]

Output:

0.50000

Constraints:

3 <= points.length <= 50

-50 <= x

i

, y

i

<= 50

All the given points are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public double largestTriangleArea(int[][] points) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public double LargestTriangleArea(int[][] points) {
```

```
}
}
```

C:

```
double largestTriangleArea(int** points, int pointsSize, int* pointsColSize)
{

}
```

Go:

```
func largestTriangleArea(points [][]int) float64 {

}
```

Kotlin:

```
class Solution {
fun largestTriangleArea(points: Array<IntArray>): Double {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn largest_triangle_area(points: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
double largestTriangleArea(List<List<int>> points) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (largest-triangle-area points)
(-> (listof (listof exact-integer?)) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Triangle Area
* 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:
double largestTriangleArea(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Triangle Area
* 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 double largestTriangleArea(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Triangle Area
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 largestTriangleArea(self, points: List[List[int]]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Triangle Area
* 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 {number}
*/
var largestTriangleArea = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Triangle Area
* 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 largestTriangleArea(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Largest Triangle Area
* 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 double LargestTriangleArea(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Largest Triangle Area
* 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
*/

double largestTriangleArea(int** points, int pointsSize, int* pointsColSize)
{

}
```

Go Solution:

```
// Problem: Largest Triangle Area
// 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 largestTriangleArea(points [][]int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun largestTriangleArea(points: Array<IntArray>): Double {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Triangle Area
// 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 largest_triangle_area(points: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

```
class Solution {
double largestTriangleArea(List<List<int>> points) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (largest-triangle-area points)
(-> (listof (listof exact-integer?)) flonum?)
)
```