Problem 3009: Maximum Number of Intersections on the Chart

Problem Information

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

Problem Description

There is a line chart consisting of

n

points connected by line segments. You are given a

1-indexed

integer array

y

. The

k

th

point has coordinates

(k, y[k])

. There are no horizontal lines; that is, no two consecutive points have the same y-coordinate.

We can draw an infinitely long horizontal line. Return

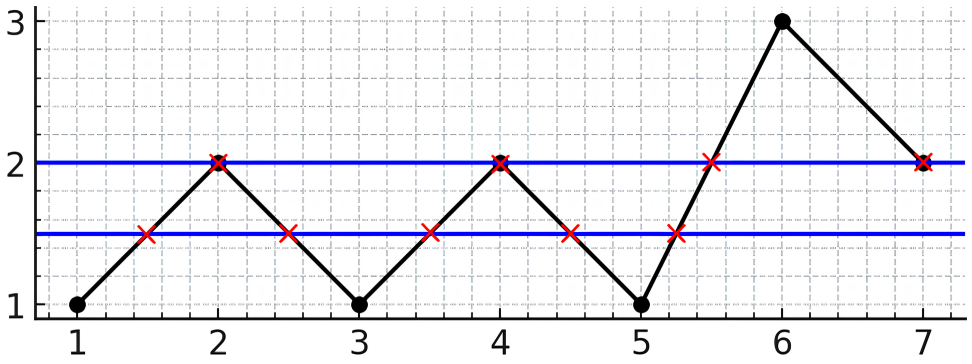
the

maximum

number of points of intersection of the line with the chart

.

Example 1:



Input:

y = [1,2,1,2,1,3,2]

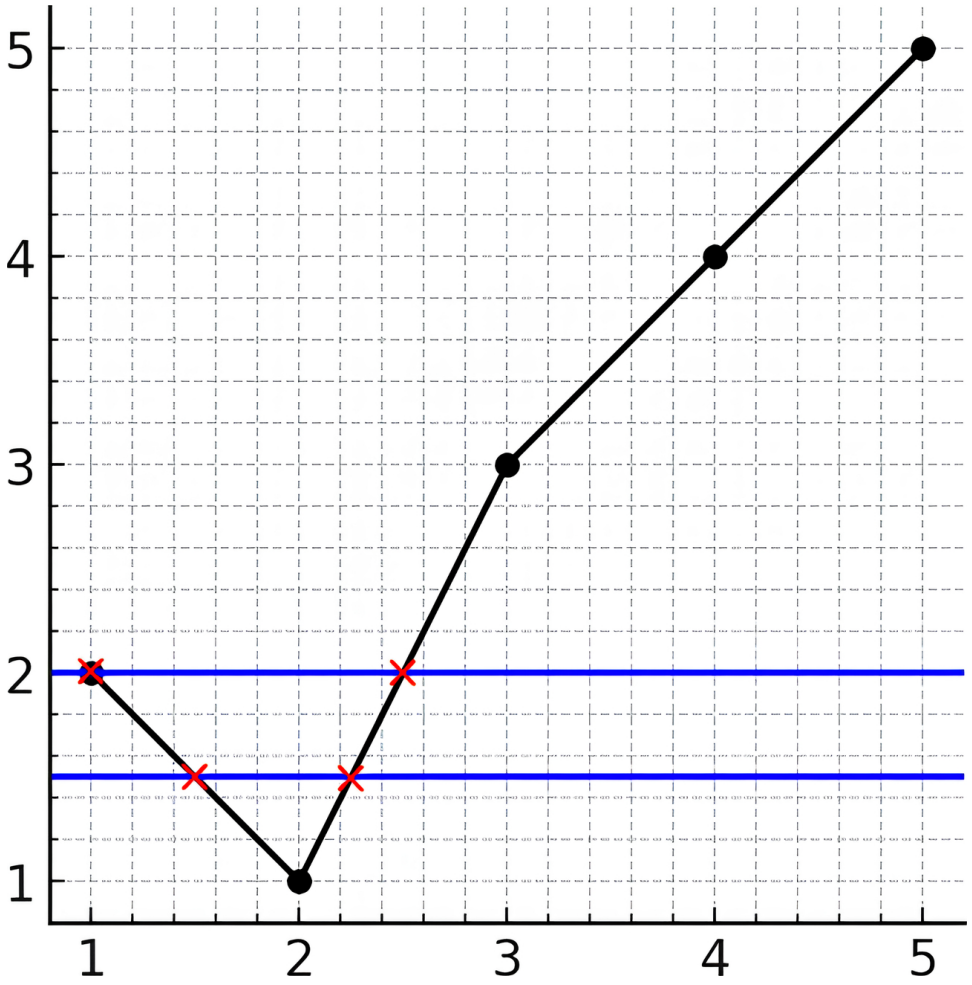
Output:

5

Explanation:

As you can see in the image above, the line y = 1.5 has 5 intersections with the chart (in red crosses). You can also see the line y = 2 which intersects the chart in 4 points (in red crosses). It can be shown that there is no horizontal line intersecting the chart at more than 5 points. So the answer would be 5.

Example 2:



Input:

y = [2,1,3,4,5]

Output:

2

Explanation:

As you can see in the image above, the line y = 1.5 has 2 intersections with the chart (in red crosses). You can also see the line y = 2 which intersects the chart in 2 points (in red crosses). It can be shown that there is no horizontal line intersecting the chart at more than 2 points. So the answer would be 2.

Constraints:

2 <= y.length <= 10

5

1 <= y[i] <= 10

9

y[i] != y[i + 1]

for

i

in range

[1, n - 1]

Code Snippets

C++:

```
class Solution {
public:
int maxIntersectionCount(vector<int>& y) {

}
};
```

Java:

```
class Solution {
public int maxIntersectionCount(int[] y) {

}
}
```

Python3:

```
class Solution:
def maxIntersectionCount(self, y: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxIntersectionCount(y: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxIntersectionCount(int[] y) {

}
}
```

C:

```
int maxIntersectionCount(int* y, int ySize) {

}
```

Go:

```
func maxIntersectionCount(y []int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun maxIntersectionCount(y: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxIntersectionCount(_ y: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_intersection_count(y: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} y
# @return {Integer}
def max_intersection_count(y)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $y
* @return Integer
*/
```

```
function maxIntersectionCount($y) {

}
}
```

Dart:

```
class Solution {
int maxIntersectionCount(List<int> y) {

}
}
```

Scala:

```
object Solution {
def maxIntersectionCount(y: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_intersection_count(y :: [integer]) :: integer
def max_intersection_count(y) do

end
end
```

Erlang:

```
-spec max_intersection_count(Y :: [integer()]) -> integer().
max_intersection_count(Y) ->
.
```

Racket:

```
(define/contract (max-intersection-count y)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
* Tags: array, tree, math, hash, sort
*
* 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:
int maxIntersectionCount(vector<int>& y) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
* Tags: array, tree, math, hash, sort
*
* 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 int maxIntersectionCount(int[] y) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Intersections on the Chart
Difficulty: Hard
Tags: array, tree, math, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
* Tags: array, tree, math, hash, sort
*
* 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[]} y
* @return {number}
*/
var maxIntersectionCount = function(y) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
* Tags: array, tree, math, hash, sort
*
* 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 maxIntersectionCount(y: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
* Tags: array, tree, math, hash, sort
*
* 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 int MaxIntersectionCount(int[] y) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Intersections on the Chart
* Difficulty: Hard
```

```
* Tags: array, tree, math, hash, sort
*
* 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
*/

int maxIntersectionCount(int* y, int ySize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Intersections on the Chart
// Difficulty: Hard
// Tags: array, tree, math, hash, sort
//
// 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 maxIntersectionCount(y []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxIntersectionCount(y: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxIntersectionCount(_ y: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Intersections on the Chart
// Difficulty: Hard
// Tags: array, tree, math, hash, sort
//
// 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 max_intersection_count(y: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} y
# @return {Integer}
def max_intersection_count(y)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $y
* @return Integer
*/
function maxIntersectionCount($y) {

}
}
```

Dart Solution:

```
class Solution {
int maxIntersectionCount(List<int> y) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maxIntersectionCount(y: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_intersection_count(y :: [integer]) :: integer
def max_intersection_count(y) do

end
end
```

Erlang Solution:

```
-spec max_intersection_count(Y :: [integer()]) -> integer().
max_intersection_count(Y) ->
.
```

Racket Solution:

```
(define/contract (max-intersection-count y)
(-> (listof exact-integer?) exact-integer?)
)
```