Problem 1499: Max Value of Equation

Problem Information

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

Problem Description

You are given an array

points

containing the coordinates of points on a 2D plane, sorted by the x-values, where

points[i] = [x

i

, y

i

]

such that

x

i

< x

j

for all

1 <= i < j <= points.length

. You are also given an integer

k

.

Return

the maximum value of the equation

y

i

+ y

j

+ |x

i

- x

j

|

where

|x

i

- x

j

| <= k

and

1 <= i < j <= points.length

.

It is guaranteed that there exists at least one pair of points that satisfy the constraint

|x

i

- x

j

| <= k

.

Example 1:

Input:

points = [[1,3],[2,0],[5,10],[6,-10]], k = 1

Output:

4

Explanation:

The first two points satisfy the condition |x

i

- x

j

| <= 1 and if we calculate the equation we get 3 + 0 + |1 - 2| = 4. Third and fourth points also satisfy the condition and give a value of 10 + -10 + |5 - 6| = 1. No other pairs satisfy the condition, so we return the max of 4 and 1.

Example 2:

Input:

points = [[0,0],[3,0],[9,2]], k = 3

Output:

3

Explanation:

Only the first two points have an absolute difference of 3 or less in the x-values, and give the value of 0 + 0 + |0 - 3| = 3.

Constraints:

2 <= points.length <= 10

5

points[i].length == 2

-10

8

<= x

i

, y

i

<= 10

8

0 <= k <= 2 * 10

8

x

i

< x

j

for all

1 <= i < j <= points.length

x

i

form a strictly increasing sequence.

Code Snippets

C++:

```
class Solution {
public:
```

```
int findMaxValueOfEquation(vector<vector<int>>& points, int k) {

}
};
```

Java:

```
class Solution {
public int findMaxValueOfEquation(int[][] points, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findMaxValueOfEquation(points: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int FindMaxValueOfEquation(int[][] points, int k) {

}
}
```

C:

```
int findMaxValueOfEquation(int** points, int pointsSize, int* pointsColSize,
int k) {

}
```

Go:

```
func findMaxValueOfEquation(points [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findMaxValueOfEquation(points: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func findMaxValueOfEquation(_ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_max_value_of_equation(points: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def find_max_value_of_equation(points, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function findMaxValueOfEquation($points, $k) {

}
}
```

Dart:

```
class Solution {
int findMaxValueOfEquation(List<List<int>> points, int k) {

}
}
```

Scala:

```
object Solution {
def findMaxValueOfEquation(points: Array[Array[Int]], k: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec find_max_value_of_equation(points :: [[integer]], k :: integer) ::
integer
def find_max_value_of_equation(points, k) do

end
end
```

Erlang:

```
-spec find_max_value_of_equation(Points :: [[integer()]], K :: integer()) ->
integer().
find_max_value_of_equation(Points, K) ->
.
```

Racket:

```
(define/contract (find-max-value-of-equation points k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 findMaxValueOfEquation(vector<vector<int>>& points, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 findMaxValueOfEquation(int[][] points, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Max Value of Equation
Difficulty: Hard
Tags: array, sort, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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
* @param {number} k
* @return {number}
*/
var findMaxValueOfEquation = function(points, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 findMaxValueOfEquation(points: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 FindMaxValueOfEquation(int[][] points, int k) {

}
}
```

C Solution:

```
/*
* Problem: Max Value of Equation
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 findMaxValueOfEquation(int** points, int pointsSize, int* pointsColSize,
int k) {

}
```

Go Solution:

```
// Problem: Max Value of Equation
// Difficulty: Hard
// Tags: array, sort, queue, heap
//
// 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 findMaxValueOfEquation(points [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMaxValueOfEquation(points: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMaxValueOfEquation(_ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Max Value of Equation
// Difficulty: Hard
// Tags: array, sort, queue, heap
//
// 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 find_max_value_of_equation(points: Vec<Vec<i32>>, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def find_max_value_of_equation(points, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function findMaxValueOfEquation($points, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findMaxValueOfEquation(List<List<int>> points, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findMaxValueOfEquation(points: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_max_value_of_equation(points :: [[integer]], k :: integer) ::
integer
def find_max_value_of_equation(points, k) do

end
end
```

Erlang Solution:

```
-spec find_max_value_of_equation(Points :: [[integer()]], K :: integer()) ->
integer().
find_max_value_of_equation(Points, K) ->
.
```

Racket Solution:

```
(define/contract (find-max-value-of-equation points k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```