Problem 1779: Find Nearest Point That Has the Same X or Y Coordinate

Problem Information

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

Problem Description

You are given two integers,

x

and

y

, which represent your current location on a Cartesian grid:

(x, y)

. You are also given an array

points

where each

points[i] = [a

i

, b

i

]

represents that a point exists at

(a

i

, b

i

)

. A point is

valid

if it shares the same x-coordinate or the same y-coordinate as your location.

Return

the index

(0-indexed)

of the

valid

point with the smallest

Manhattan distance

from your current location

. If there are multiple, return

the valid point with the

smallest

index

. If there are no valid points, return

-1

.

The

Manhattan distance

between two points

(x

1

, y

1

)

and

(x

2

, y

2

)

is

abs(x

1

- x

2

) + abs(y

1

- y

2

)

.

Example 1:

Input:

x = 3, y = 4, points = [[1,2],[3,1],[2,4],[2,3],[4,4]]

Output:

2

Explanation:

Of all the points, only [3,1], [2,4] and [4,4] are valid. Of the valid points, [2,4] and [4,4] have the smallest Manhattan distance from your current location, with a distance of 1. [2,4] has the smallest index, so return 2.

Example 2:

Input:

x = 3, y = 4, points = [[3,4]]

Output:

0

Explanation:

The answer is allowed to be on the same location as your current location.

Example 3:

Input:

x = 3, y = 4, points = [[2,3]]

Output:

-1

Explanation:

There are no valid points.

Constraints:

1 <= points.length <= 10

4

points[i].length == 2

1 <= x, y, a

i

, b

i

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
int nearestValidPoint(int x, int y, vector<vector<int>>& points) {

}
};
```

Java:

```
class Solution {
public int nearestValidPoint(int x, int y, int[][] points) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @param {number[][]} points
* @return {number}
*/
var nearestValidPoint = function(x, y, points) {

};
```

TypeScript:

```
function nearestValidPoint(x: number, y: number, points: number[][]): number
{

};
```

C#:

```
public class Solution {
public int NearestValidPoint(int x, int y, int[][] points) {

}
}
```

C:

```
int nearestValidPoint(int x, int y, int** points, int pointsSize, int*
pointsColSize) {

}
```

Go:

```
func nearestValidPoint(x int, y int, points [][]int) int {

}
```

Kotlin:

```
class Solution {
fun nearestValidPoint(x: Int, y: Int, points: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func nearestValidPoint(_ x: Int, _ y: Int, _ points: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn nearest_valid_point(x: i32, y: i32, points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer[][]} points
# @return {Integer}
def nearest_valid_point(x, y, points)

end
```

PHP:

```
class Solution {

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

```
*/
function nearestValidPoint($x, $y, $points) {

}
}
```

Dart:

```
class Solution {
int nearestValidPoint(int x, int y, List<List<int>> points) {

}
}
```

Scala:

```
object Solution {
def nearestValidPoint(x: Int, y: Int, points: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec nearest_valid_point(x :: integer, y :: integer, points :: [[integer]])
:: integer
def nearest_valid_point(x, y, points) do

end
end
```

Erlang:

```
-spec nearest_valid_point(X :: integer(), Y :: integer(), Points ::
[[integer()]]) -> integer().
nearest_valid_point(X, Y, Points) ->
.
```

Racket:

```
(define/contract (nearest-valid-point x y points)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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 nearestValidPoint(int x, int y, vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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 nearestValidPoint(int x, int y, int[][] points) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Find Nearest Point That Has the Same X or Y Coordinate
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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} x
* @param {number} y
* @param {number[][]} points
* @return {number}
*/
var nearestValidPoint = function(x, y, points) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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 nearestValidPoint(x: number, y: number, points: number[][]): number
{

};
```

C# Solution:

```
/*
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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 NearestValidPoint(int x, int y, int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Find Nearest Point That Has the Same X or Y Coordinate
* Difficulty: Easy
* Tags: array
*
* 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 nearestValidPoint(int x, int y, int** points, int pointsSize, int*
pointsColSize) {

}
```

Go Solution:

```
// Problem: Find Nearest Point That Has the Same X or Y Coordinate
// Difficulty: Easy
// Tags: array
//
// 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 nearestValidPoint(x int, y int, points [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun nearestValidPoint(x: Int, y: Int, points: Array<IntArray>): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func nearestValidPoint(_ x: Int, _ y: Int, _ points: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Nearest Point That Has the Same X or Y Coordinate
// Difficulty: Easy
// Tags: array
//
// 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 nearest_valid_point(x: i32, y: i32, points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer[][]} points
# @return {Integer}
def nearest_valid_point(x, y, points)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer[][] $points
* @return Integer
*/
function nearestValidPoint($x, $y, $points) {

}
}
```

Dart Solution:

```
class Solution {
int nearestValidPoint(int x, int y, List<List<int>> points) {

}
}
```

Scala Solution:

```
object Solution {
def nearestValidPoint(x: Int, y: Int, points: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec nearest_valid_point(x :: integer, y :: integer, points :: [[integer]])
:: integer
def nearest_valid_point(x, y, points) do

end
end
```

Erlang Solution:

```
-spec nearest_valid_point(X :: integer(), Y :: integer(), Points ::
[[integer()]]) -> integer().
nearest_valid_point(X, Y, Points) ->
.
```

Racket Solution:

```
(define/contract (nearest-valid-point x y points)
(-> exact-integer? exact-integer? (listof (listof exact-integer?))
exact-integer?)
)
```