Problem 3275: K-th Nearest Obstacle Queries

Problem Information

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

Problem Description

There is an infinite 2D plane.

You are given a positive integer

k

. You are also given a 2D array

queries

, which contains the following queries:

queries[i] = [x, y]

: Build an obstacle at coordinate

(x, y)

in the plane. It is guaranteed that there is

no

obstacle at this coordinate when this query is made.

After each query, you need to find the

distance

of the

k

th

nearest

obstacle from the origin.

Return an integer array

results

where

results[i]

denotes the

k

th

nearest obstacle after query

i

, or

results[i] == -1

if there are less than

k

obstacles.

Note

that initially there are

no

obstacles anywhere.

The

distance

of an obstacle at coordinate

(x, y)

from the origin is given by

|x| + |y|

.

Example 1:

Input:

queries = [[1,2],[3,4],[2,3],[-3,0]], k = 2

Output:

[-1,7,5,3]

Explanation:

Initially, there are 0 obstacles.

After

queries[0]

, there are less than 2 obstacles.

After

queries[1]

, there are obstacles at distances 3 and 7.

After

queries[2]

, there are obstacles at distances 3, 5, and 7.

After

queries[3]

, there are obstacles at distances 3, 3, 5, and 7.

Example 2:

Input:

queries = [[5,5],[4,4],[3,3]], k = 1

Output:

[10,8,6]

Explanation:

After

queries[0]

, there is an obstacle at distance 10.

After

queries[1]

, there are obstacles at distances 8 and 10.

After

queries[2]

, there are obstacles at distances 6, 8, and 10.

Constraints:

1 <= queries.length <= 2 * 10

5

All

queries[i]

are unique.

-10

9

<= queries[i][0], queries[i][1] <= 10

9

1 <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> resultsArray(vector<vector<int>>& queries, int k) {

}
};
```

Java:

```
class Solution {
public int[] resultsArray(int[][] queries, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function resultsArray(queries: number[][], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ResultsArray(int[][] queries, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* resultsArray(int** queries, int queriesSize, int* queriesColSize, int k,
int* returnSize) {

}
```

Go:

```
func resultsArray(queries [][]int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun resultsArray(queries: Array<IntArray>, k: Int): IntArray {

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} queries
# @param {Integer} k
# @return {Integer[]}
def results_array(queries, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $queries
* @param Integer $k
* @return Integer[]
*/
function resultsArray($queries, $k) {

}
}
```

Dart:

```
class Solution {
List<int> resultsArray(List<List<int>> queries, int k) {

}
}
```

Scala:

```
object Solution {
def resultsArray(queries: Array[Array[Int]], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec results_array(queries :: [[integer]], k :: integer) :: [integer]
def results_array(queries, k) do

end
end
```

Erlang:

```
-spec results_array(Queries :: [[integer()]], K :: integer()) -> [integer()].
results_array(Queries, K) ->
.
```

Racket:

```
(define/contract (results-array queries k)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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:
vector<int> resultsArray(vector<vector<int>>& queries, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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[] resultsArray(int[][] queries, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K-th Nearest Obstacle Queries
Difficulty: Medium
Tags: array, tree, queue, heap

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 resultsArray(self, queries: List[List[int]], k: int) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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[][]} queries
* @param {number} k
* @return {number[]}
*/
var resultsArray = function(queries, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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 resultsArray(queries: number[][], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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[] ResultsArray(int[][] queries, int k) {

}
}
```

C Solution:

```
/*
* Problem: K-th Nearest Obstacle Queries
* Difficulty: Medium
* Tags: array, tree, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* resultsArray(int** queries, int queriesSize, int* queriesColSize, int k,
int* returnSize) {

}
```

Go Solution:

```
// Problem: K-th Nearest Obstacle Queries
// Difficulty: Medium
// Tags: array, tree, queue, heap
//
// 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 resultsArray(queries [][]int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun resultsArray(queries: Array<IntArray>, k: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: K-th Nearest Obstacle Queries
// Difficulty: Medium
// Tags: array, tree, queue, heap
//
```

```
// 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 results_array(queries: Vec<Vec<i32>>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} queries
# @param {Integer} k
# @return {Integer[]}
def results_array(queries, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $queries
* @param Integer $k
* @return Integer[]
*/
function resultsArray($queries, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> resultsArray(List<List<int>> queries, int k) {

}
}
```

Scala Solution:

```
object Solution {
def resultsArray(queries: Array[Array[Int]], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec results_array(queries :: [[integer]], k :: integer) :: [integer]
def results_array(queries, k) do

end
end
```

Erlang Solution:

```
-spec results_array(Queries :: [[integer()]], K :: integer()) -> [integer()].
results_array(Queries, K) ->
.
```

Racket Solution:

```
(define/contract (results-array queries k)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```