

Problem 1828: Queries on Number of Points Inside a Circle

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an array

points

where

$\text{points}[i] = [x$

i

$, y$

i

$]$

is the coordinates of the

i

th

point on a 2D plane. Multiple points can have the

same

coordinates.

You are also given an array

queries

where

`queries[j] = [x`

`j`

`, y`

`j`

`, r`

`j`

`]`

describes a circle centered at

`(x`

`j`

`, y`

`j`

`)`

with a radius of

`r`

j

.

For each query

queries[j]

, compute the number of points

inside

the

j

th

circle. Points

on the border

of the circle are considered

inside

.

Return

an array

answer

, where

answer[j]

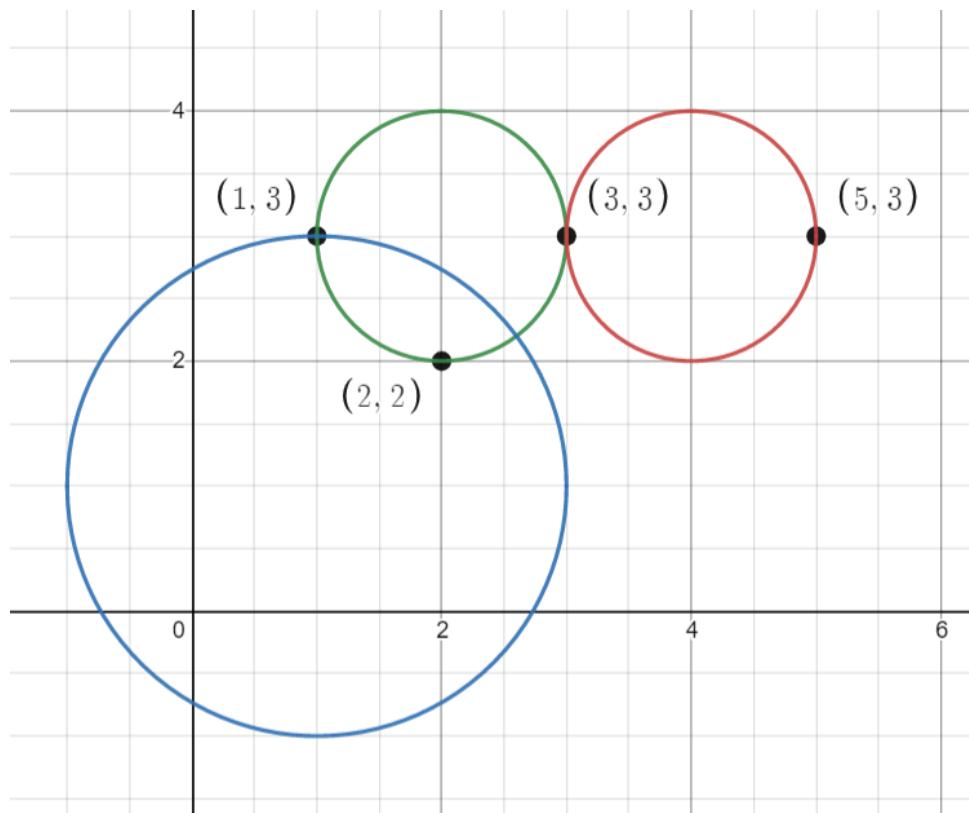
is the answer to the

j

th

query

Example 1:



Input:

```
points = [[1,3],[3,3],[5,3],[2,2]], queries = [[2,3,1],[4,3,1],[1,1,2]]
```

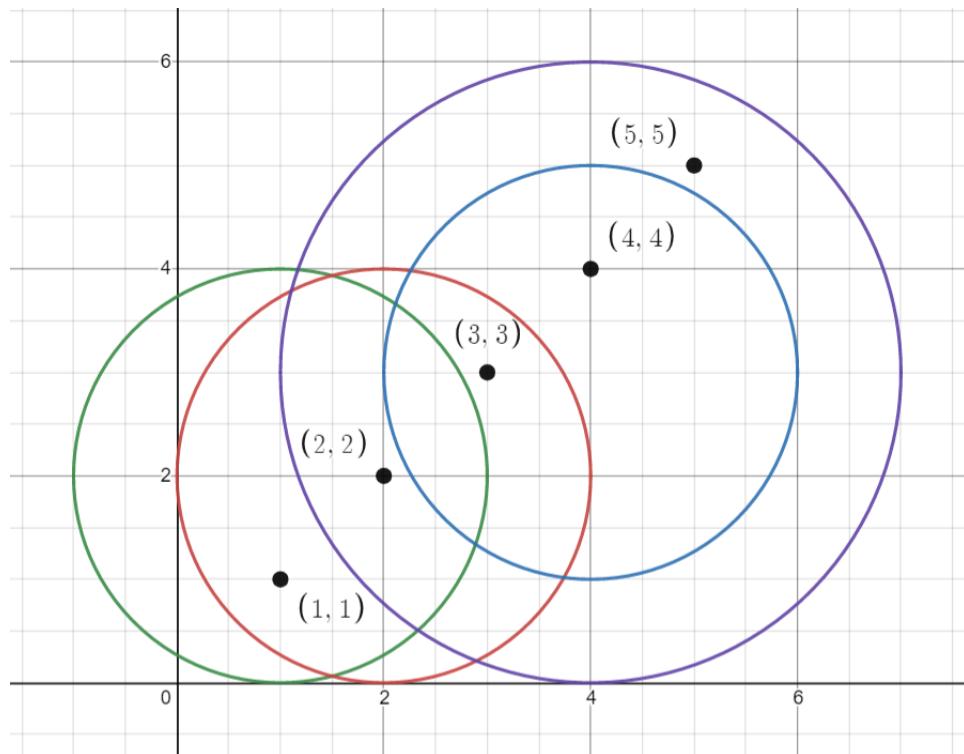
Output:

```
[3,2,2]
```

Explanation:

The points and circles are shown above. queries[0] is the green circle, queries[1] is the red circle, and queries[2] is the blue circle.

Example 2:



Input:

```
points = [[1,1],[2,2],[3,3],[4,4],[5,5]], queries = [[1,2,2],[2,2,2],[4,3,2],[4,3,3]]
```

Output:

```
[2,3,2,4]
```

Explanation:

The points and circles are shown above. queries[0] is green, queries[1] is red, queries[2] is blue, and queries[3] is purple.

Constraints:

```
1 <= points.length <= 500
```

`points[i].length == 2`

`0 <= x`

`i`

`, y`

`i`

`<= 500`

`1 <= queries.length <= 500`

`queries[j].length == 3`

`0 <= x`

`j`

`, y`

`j`

`<= 500`

`1 <= r`

`j`

`<= 500`

All coordinates are integers.

Follow up:

Could you find the answer for each query in better complexity than

$O(n)$

?

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> countPoints(vector<vector<int>>& points, vector<vector<int>>&  
queries) {  
  
}  
};
```

Java:

```
class Solution {  
public int[] countPoints(int[][] points, int[][] queries) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
 * @param {number[][]} points
 * @param {number[][]} queries
 * @return {number[]}
 */
var countPoints = function(points, queries) {
};
```

TypeScript:

```
function countPoints(points: number[][], queries: number[][][]): number[] {
};
```

C#:

```
public class Solution {
    public int[] CountPoints(int[][] points, int[][] queries) {
        return null;
    }
}
```

C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* countPoints(int** points, int pointsSize, int* pointsColSize, int** queries, int queriesSize, int* queriesColSize, int* returnSize) {
    return NULL;
}
```

Go:

```
func countPoints(points [][]int, queries [][]int) []int {
}
```

Kotlin:

```
class Solution {
    fun countPoints(points: Array<IntArray>, queries: Array<IntArray>): IntArray
```

```
{  
}  
}  
}
```

Swift:

```
class Solution {  
    func countPoints(_ points: [[Int]], _ queries: [[Int]]) -> [Int] {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn count_points(points: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) ->  
    Vec<i32> {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer[][]} points  
# @param {Integer[][]} queries  
# @return {Integer[]}  
def count_points(points, queries)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $points  
     * @param Integer[][] $queries  
     * @return Integer[]  
     */  
    function countPoints($points, $queries) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
List<int> countPoints(List<List<int>> points, List<List<int>> queries) {  
}  
}  
}
```

Scala:

```
object Solution {  
def countPoints(points: Array[Array[Int]], queries: Array[Array[Int]]):  
  Array[Int] = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec count_points(points :: [[integer]], queries :: [[integer]]) ::  
  [integer]  
def count_points(points, queries) do  
  
end  
end
```

Erlang:

```
-spec count_points(Points :: [[integer()]], Queries :: [[integer()]]) ->  
  [integer()].  
count_points(Points, Queries) ->  
.
```

Racket:

```
(define/contract (count-points points queries)  
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof  
    exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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:
vector<int> countPoints(vector<vector<int>>& points, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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 int[] countPoints(int[][] points, int[][] queries) {

}
```

Python3 Solution:

```
"""
Problem: Queries on Number of Points Inside a Circle
Difficulty: Medium
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 countPoints(self, points: List[List[int]], queries: List[List[int]]) ->
        List[int]:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):

    def countPoints(self, points, queries):
        """
:type points: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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
 * @param {number[][]} queries
```

```
* @return {number[]}
*/
var countPoints = function(points, queries) {
};
```

TypeScript Solution:

```
/** 
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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 countPoints(points: number[][][], queries: number[][][]): number[] {
};
```

C# Solution:

```
/*
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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 int[] CountPoints(int[][][] points, int[][][] queries) {
        }
}
```

C Solution:

```

/*
 * Problem: Queries on Number of Points Inside a Circle
 * Difficulty: Medium
 * 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
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* countPoints(int** points, int pointsSize, int* pointsColSize, int** queries,
                 int queriesSize, int* queriesColSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Queries on Number of Points Inside a Circle
// Difficulty: Medium
// 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 countPoints(points [][]int, queries [][]int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun countPoints(points: Array<IntArray>, queries: Array<IntArray>): IntArray {
        }

    }
}

```

Swift Solution:

```

class Solution {
    func countPoints(_ points: [[Int]], _ queries: [[Int]]) -> [Int] {
        }
    }
}

```

Rust Solution:

```

// Problem: Queries on Number of Points Inside a Circle
// Difficulty: Medium
// 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 count_points(points: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) ->
    Vec<i32> {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer[][]} points
# @param {Integer[][]} queries
# @return {Integer[]}
def count_points(points, queries)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[][] $points
     * @param Integer[][] $queries
     * @return Integer[]
     */
    function countPoints($points, $queries) {

```

```
}
```

```
}
```

Dart Solution:

```
class Solution {  
List<int> countPoints(List<List<int>> points, List<List<int>> queries) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def countPoints(points: Array[Array[Int]], queries: Array[Array[Int]]):  
  Array[Int] = {  
  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec count_points(points :: [[integer]], queries :: [[integer]]) ::  
  [integer]  
def count_points(points, queries) do  
  
end  
end
```

Erlang Solution:

```
-spec count_points(Points :: [[integer()]], Queries :: [[integer()]]) ->  
  [integer()].  
count_points(Points, Queries) ->  
.
```

Racket Solution:

```
(define/contract (count-points points queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?)))
)
```