

# Problem 2013: Detect Squares

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 51.95%

**Paid Only:** No

**Tags:** Array, Hash Table, Design, Counting

## Problem Description

You are given a stream of points on the X-Y plane. Design an algorithm that:

\* **Adds** new points from the stream into a data structure. **Duplicate** points are allowed and should be treated as different points. \* Given a query point, **counts** the number of ways to choose three points from the data structure such that the three points and the query point form an **axis-aligned square** with **positive area**.

An **axis-aligned square** is a square whose edges are all the same length and are either parallel or perpendicular to the x-axis and y-axis.

Implement the `DetectSquares` class:

\* `DetectSquares()` Initializes the object with an empty data structure. \* `void add(int[] point)` Adds a new point `point = [x, y]` to the data structure. \* `int count(int[] point)` Counts the number of ways to form **axis-aligned squares** with point `point = [x, y]` as described above.

**Example 1:**

 (<https://assets.leetcode.com/uploads/2021/09/01/image.png>)

**Input** ["DetectSquares", "add", "add", "add", "count", "count", "add", "count"]  
[[], [[3, 10]], [[11, 2]], [[3, 2]], [[11, 10]], [[14, 8]], [[11, 2]], [[11, 10]]  
**Output** [null, null, null, null, 1, 0, null, 2]  
**Explanation** DetectSquares detectSquares = new DetectSquares();  
detectSquares.add([3, 10]); detectSquares.add([11, 2]); detectSquares.add([3, 2]);  
detectSquares.count([11, 10]); // return 1. You can choose: // - The first, second, and third  
points detectSquares.count([14, 8]); // return 0. The query point cannot form a square with any

points in the data structure. `detectSquares.add([11, 2]);` // Adding duplicate points is allowed.  
`detectSquares.count([11, 10]);` // return 2. You can choose: // - The first, second, and third points // - The first, third, and fourth points

**\*\*Constraints:\*\***

\* `point.length == 2` \* `0 <= x, y <= 1000` \* At most `3000` calls **\*\*in total\*\*** will be made to `add` and `count`.

## Code Snippets

### C++:

```
class DetectSquares {
public:
    DetectSquares() {

    }

    void add(vector<int> point) {

    }

    int count(vector<int> point) {

    }
};

/**
 * Your DetectSquares object will be instantiated and called as such:
 * DetectSquares* obj = new DetectSquares();
 * obj->add(point);
 * int param_2 = obj->count(point);
 */
```

### Java:

```
class DetectSquares {

    public DetectSquares() {
```

```

}

public void add(int[] point) {

}

public int count(int[] point) {

}

}

/**
 * Your DetectSquares object will be instantiated and called as such:
 * DetectSquares obj = new DetectSquares();
 * obj.add(point);
 * int param_2 = obj.count(point);
 */

```

### Python3:

```

class DetectSquares:

    def __init__(self):

    def add(self, point: List[int]) -> None:

    def count(self, point: List[int]) -> int:

    # Your DetectSquares object will be instantiated and called as such:
    # obj = DetectSquares()
    # obj.add(point)
    # param_2 = obj.count(point)

```