**Problem Name:** Queries on number of points inside a circle

**Topics:** Array, Math, Geometry

**Companies:** Google

**Level:** Medium

**Language:** C++

**Problem Statement:** You are given an array points where points[i] = [xi, yi] is the coordinates of the ith point on a 2D plane. Multiple points can have the **same** coordinates.

You are also given an array queries where queries[j] = [xj, yj, rj] describes a circle centered at (xj, yj) with a radius of rj.

For each query queries[j], compute the number of points **inside** the jth circle. Points **on the border** of the circle are considered **inside**.

Return *an array*answer*, where*answer[j]*is the answer to the*jth*query*.

**Input Format:**

First line of the input contains integer n (length of first 2D Vector) no of coordinate points.

Second line contain 2n (x,y) space separated integer values of first vector.

Third line contain integer value m (length of second 2D Vector) no of queries

Fourth line contain 3m (x,y,r) space separated integer values of second vector.

Ex:

4

1 3 3 3 5 3 2 2

3

2 3 1 4 3 1 1 1 2

**Output Format:** Print an array answer, where answer[j] is the answer to the jth query.

Ex for above input output would bee

3 2 2

**Constraints:**

* 1 <= points.length <= 500
* points[i].length == 2
* 0 <= x​​​​​​i, y​​​​​​i <= 500
* 1 <= queries.length <= 500
* queries[j].length == 3
* 0 <= xj, yj <= 500
* 1 <= rj <= 500
* All coordinates are integers.

**Examples:**

**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.

**Brute force Solution:**

**Explanation:** Basic Mathematics will tell you that the distance between two points (x1, y1) and (x2, y2) is given by - sqrt((x1 - x2)^2 + (y1 - y2)^2).

We need the distance between the center of the circle and a point less than or equal to radius of the circle for it to be considered as a point inside the circle.

Thus, we need - (circle\_center\_x - x1) ^ 2 - (circle\_center\_y - y1) ^ 2 <= r \* r for a point to be inside the circle.

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<int> countPoints(vector<vector<int>>& points, vector<vector<int>>& queries) {

    vector<int> ans(queries.size());

    for(int i = 0; i < queries.size(); i++){

        int x = queries[i][0], y = queries[i][1], r = queries[i][2];

        for(int j = 0; j < points.size(); j++){

            if((x - points[j][0]) \* (x - points[j][0]) + (y - points[j][1]) \* (y - points[j][1]) <= r \* r)

                ans[i]++;

        }

    }

    return ans;

}

int main() {

    int m, n;

    cin>>n;

    vector<vector<int>> points;

    for(int i=0; i<n; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        points.push\_back(temp);

    }

    cin>>m;

    vector<vector<int>> queries;

    for(int i=0; i<m; i++){

        vector<int> temp(3);

        for(int j=0; j<3; j++){

            cin>>temp[j];

        }

        queries.push\_back(temp);

    }

    vector<int> result;

    result = countPoints(points, queries);

    for(int i=0; i< m; i++){

        cout<<result[i]<<" ";

    }

    return 0;

}

**Time Complexity**: O(M\*N)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:** To reduce comparisons we can think of that in each query we are serching points by comparing each and every point with central point and by this argument first intution comes in mind to fasten every searching we use binary search.

1. Sort points array on basis of one axis either x or y (here we are using x axis)
2. For each query:  
   a. Now if we find the first point fp which is coming inside or on boundry of circle on the x axis then we won't need to iterate other points (it's saving our extra comparison)  
   b. Because points array is sorted on the basis of x-axis so we just need to iterate from fp point to untill we get out from the circle (we can imagine like we are travelling on parallel line of x-axis that is intersecting centre point of circle)  
   c. now we can just compare point with centre point to know weather it's inside or outside.

**Code:**

#include <bits/stdc++.h>

using namespace std;

int sqr(int a) {

    return a \* a;

}

int find\_left\_boundry\_index(vector<vector<int> > & points , int x\_center , int y\_center , int r) {

    int lo = 0, hi = points.size();

    while (lo < hi) {

        int mi = lo + (hi - lo ) / 2;

        if (x\_center - r <= points[mi][0])

            hi = mi;

        else

            lo = mi + 1;

    }

    return hi == points.size() or (points[hi][0] > x\_center + r or points[hi][0] < x\_center - r) ? points.size() : hi;

}

vector<int> countPoints(vector<vector<int>>& points, vector<vector<int>>& queries) {

    sort(points.begin(), points.end());

    vector<int> ans;

    for (int i = 0; i < queries.size(); i++) {

        int x\_center = queries[i][0], y\_center = queries[i][1], r = queries[i][2];

        int index = find\_left\_boundry\_index(points, x\_center , y\_center, r);

        int count = 0;

        for (int j = index; j < points.size() and points[j][0] <= x\_center + r; j++) {

            int x = points[j][0];

            int y = points[j][1];

            count += sqr(x\_center - x) + sqr(y\_center - y) <= r \* r;

        }

        ans.push\_back(count);

    }

    return ans;

}

int main() {

    int m, n;

    cin>>n;

    vector<vector<int>> points;

    for(int i=0; i<n; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        points.push\_back(temp);

    }

    cin>>m;

    vector<vector<int>> queries;

    for(int i=0; i<m; i++){

        vector<int> temp(3);

        for(int j=0; j<3; j++){

            cin>>temp[j];

        }

        queries.push\_back(temp);

    }

    vector<int> result;

    result = countPoints(points, queries);

    for(int i=0; i< m; i++){

        cout<<result[i]<<" ";

    }

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)