HomeWork2 Goldy

A)

Solved using Range Tree and Fractional Cascading

1. Building the Range Tree:
    a. Input conversion from rectilinear coordinates to polar

        (Radius)R = Math.sqrt(x*x + y*y);
        (Angle)Q = 180/Math.PI*Math.acos(x/R);
        Cos Inverse lies in 0 t0 180 degrees,
        →if the point is in 3rd quadrant: Q = Q+90
        →if in 4th : Q = Q + 270

    b. An arraylist of points AST
    c. Sort arraylist on both the variables using MergeSort- ASTr, ASTq
        i. If two numbers have equal R or Equal Q thet are sorted on the basis of the other dimension
        ii. Cases where both R and Q are equal are not handled as this is the case of asteroids (physical objects)
    d. 2D RangeSearch Using Range Tree and Fractional Cascading
        i. Build the binary tree using ASTr
        ii. At everynode an Array of all the points in the subtree sorted by angle is stored
            1. Every element of that array also stores the index of the smallest element equal or greater than itself in the left and right child if present
                a. The arrays are built bottom to top starting from leaves to the root. Arrays are merged using the **Merge Method** of Merge Sort
                b. Indices are copied (as per the Fractional Cascading Rule) if such an element is present or "-1" if no element equal or greater is found or the node is child and there is no subtree.

2. Query Preprocessing:
    a. Input (Q1, Q2, R1, R2) PreProcessing:
        i. Range = Q2 – Q1;
            1. If Range < 0→ No asteroid in range
            2. If Range > 360→ all the nodes on the annulus R1-R2
            3. else
                a. Q1 = Remainder of Q1 when divided by 360
                b. Q2 = Q1 + range
                c. if (Q2 > 360) : two queries with ranges (Q1, 360) and (0, Q2 %360)
                d. else Query with range (Q1, Q2)

3. Query:
    a. Input (Q1, Q2, R1, R2)
    b. Query on main binary tree for (R1, R2)
        i. Find the split node(first node in the range). Also in the associated array find the smallest element larger or equal to Q1 by binary sort.
        ii. Then recursively find in left and right tree the nodes in the range(detail not explained). The index of the smallest element larger or equal to Q1 is followed in the array from node to the left and right child.
        iii. Points are reported.

B) Complexity:

Time:

1. Conversion from X,Y to R, Q – O(N)
2. Sorting the points in R and Q using merge sort – O(N(LogN)
3. Building the tree :
    a. Main tree sorted by R – O(NLogN)
    b. At every node an associated subtree is stored – O(N(LogN)^2)
        i. At every tree level all the points are stored in a sorted manner. Sorting is done bottom to top by merging (better than sorting at every node) but in worst case complexity is - O(NLogN)
        ii. Number of levels – O(LogN)

Net: O(N(LogN)^2)

4. Query:
    a. Finding the split node in a binary tree - O(LogN)
    b. Finding the smallest q greater equal to q1 using binary search – O(LogN)
    c. Following the indices in constant time
    d. Reporting found points- O(k) (mk =K)

Net: For m number of queries: m(O(LogN) + k)

5. Sorting the K output: O(KLog(K))

Total : O(N(LogN)^2) + O(mLogN) + O(KLog(K))

C) Finding the asteroid closest to a point(Xo, Yo)

Algorithm on the present data structure: (assuming the data structure is built on x-y coordinates of the point).By finding the bounding box with the least width:

1. At the root Find the points with largest Y less than Yo and smallest Y larger than Yo
2. At the root Find the points with largest X less than Xo and smallest X larger than Xo
3. Check the distance with the root point if less: update the bounding box
4. Go on checking with right and left child.
5. If cannot find a better distance. That node is the closest point

In worst case it can go O(N)

An efficient data structure can be X-tree