Name: HAVI BOHRA
Roll No.: 210429

Solution (1):

Algorithm:

- Convert coordinates of every point (x,y) to (x, theta) where theta is angle b/w [0,2pi) formed by line joining centre of circle and x-axis
- Now for each pair of points (p_i , q_i) , if $p_i > q_i$ swap(p_i , q_i);
- S be the set of line segments, S = {(p_1,q_1),…..,(p_n,q_n)}.
- Divide S into equal subsets(say s1 and s2), and calculate intersections in corresponding subsets using recursion. (Say *ans1* and *ans2*).
- Now, in the combine-step, we need to calculate intersections between these two subsets.
- First, store all points individually from s1 and s2 in an array P and sort it
- Now maintain two balalanced BSTs T1 and T2 which will store points coming from P which are from s1 and s2 respectively
- Now iterate through, array P:
    - If we get p_i from s1, **insert** it into T1; similarly If we get p_i from s2, insert it into T2
    - If we get q_i from s1, then **delete** corresponding p_i from T1 and find no. of points in T2 which lies in range (p_i, q_i) using **rank**(predecessor(q_i)) - **rank**(successor(p_i))+1 , add this to *ans3* (counting intersections between s1 and s2); similarly things can be done if we get q_i from s2

Time Complexity Analysis:

1. Insert in BST = O(log(n))
2. Delete in BST = O(log(n))
3. Calculate rank in BST = O(log(n))
4. Calculate successor and predecessor in BST = O(log(n))
5. T(n) = 2*T(n/2) + O(n*log(n)) (for sorting) + 2n*log(n) = T(n) = 2*T(n/2) + O(n*log(n)).
6. Solving recurrence relation **$T(n) = O(n*\log^2(n))$.**

Pseudo-code:

```
Find_Intersection(S, i, j)
{
    if(i==j)return 0;
    ans1 = Find_Intersection(S, i, mid);
    ans2 = Find_Intersection(S, mid, j);
    ans3=0;


    For (pi, qi) in S:
    {
        If (i <= mid){
            // i is for index , negative represent s1
            P.push_back({pi,-i});
            P.push_back({qi,-i});
        }
        Else{
            P.push_back({pi,i});
            P.push_back({qi,i});
        }
    }

    Sort(P);

    Unordered_set st; // to store if for some line segment pi has already visited

    For (point, index) in P
    {
        if(index<=0){ //point from s1
            if( index is not present in st){
                T1.insert(point);
                st.insert(index);
            }
            else{
                T1.delete(S[abs(index)].first);
                ans3+= no. of points in T2 in range( S[abs(index)].first, S[abs(index)].second);
            }
        }
        else{
            if( index is not present in st){
                T2.insert(point);
                st.insert(index);
            }
            else{
                T2.delete(S[abs(index)].first);
                ans3+= no. of points in T1 in range( S[abs(index)].first, S[abs(index)].second);
            }
        }
    }

    return ans1+ans2+ans3;
}
```

Name: HAVI BOHRA
Roll No.: 210429

Solution (2)(a):
Algorithm:

- We will store the non-dominated points in height balanced BST according to x-coordinate.
- For i-th incoming point p:
    - find the successor of the p in the BST, say ***succ = Successor(p)***:
    - If p(y) < succ(y) : p is dominated by succ, do nothing
    - If p(y) > succ(y) : p is non-dominated point (x-coordinate of p is greater than the x coordinates of points before it and y – coordinates of p is greater than y-coordinates of points after it in BST)
        - Insert p in BST
        - P may dominate some points in BST- to find such points, find the predecessor of p in BST , say ***pred = Predecessor(p)***
        - While (pred != NULL && pred(y)<p(y)){
            Delete(pred);
            pred= ***Predecessor(p)***
            }

Time Complexity Analysis:

1. Calculate successor and predecessor in BST = O(log(i))
2. Time complexity of delete operation in tree of size i is O(log(i)). Maximum no. of deletion in BST from start to i-th incoming point < i , so in worst case O(i*log(i))
3. Hence, time complexity of online algorithm of non-dominated points till i-th incoming point is **O(i*log(i))**

Solution (2)(b):
Algorithm:

- Sort all the points(say set S) in non-increasing order according to z-coordinate.
- Maintain a height balanced BST (T) which will store the non-dominated points.
- Iterate through point p in S:
    - Check whether the point is non-dominated with respect to points in BST using algorithm written in (a). (i.e check if it is dominated by successor)
        - If non-dominated - insert(p,T).
        - Else - do nothing
    - As points are being iterated in non-increasing order according to z-coordinate, by addition of a new non-dominated point, **no** already existing point will be deleted
- After iterating through all points, points present in T will be non-dominated points

Time Complexity Analysis:

1. Time complexity for sorting = O(n*log(n))
2. For maintaining non-dominated points using part (a) = O(n*log(n)).
3. Total complexity = **O(n*log(n))+ O(n*log(n)) = O(n*log(n)).**