CS325: Analysis of Algorithms, Fall 2016

Group Assignment 1 Solution

Step 1

Idea: Let L_i be the line $p_i q_i$. We calculate the number of intersections, say x, between L_n with each line in $\{L_1, \ldots, L_{n-1}\}$ and recursively calculate the number of intersections, say y, between every pair of lines in $\{L_1, \ldots, L_{n-1}\}$. The final result is $x + y$.

Pseudocode:

```
IntersectionCount(P[1,...,n], Q[1,...,n])
    if n = 1
        return 0
    x ← 0
    for i ← 1 to n − 1
        x ← x + Intersection(P[n], Q[n], P[i], Q[i])
    y ← IntersectionCount(P[1,...,n − 1], Q[1,...,n − 1])
    return x + y
```

The procedure $\textsc{Intersection}(P[i], Q[i], P[j], Q[j])$ returns 1 if L_i and L_j intersect and return 0 otherwise.

```
Intersection(P[i], Q[i], P[j], Q[j])
    if P[i] < P[j] and Q[i] > Q[j]
        return 1
    if P[i] > P[j] and Q[i] < Q[j]
        return 1
    return 0
```

Proof. We give a proof by induction.

Base Case: If $n = 1$, there is no intersection.

Induction Hypothesis: For any $N < n$, $\textsc{IntersectionCount}(P[1, \ldots, N], Q[1, \ldots, N])$ correctly computes the number of intersections between line segments in $\{L_1, L_2, \ldots, L_N\}$.

Induction Step: We write the total number of intersections as the sum of:

(1) y, the number intersections between L_n and line segments in $\{L_1, \ldots, L_{n-1}\}$, and

(2) x, the number of intersections between pairs of line segments in $\{L_1, L_2, \ldots, L_{n-1}\}$.

From the pseudocode, the algorithm correctly computes x as it simply goes through every $L_i \in \{L_1, \ldots, L_{n-1}\}$ and checks if L_i intersects L_n. The induction Hypothesis implies that the algorithm correctly computes y. Thus, the algorithm correctly computes the total number of intersections. □

Running time: Let $T(n)$ be the running time of $\textsc{IntersectionCount}(P[1,\ldots,n], Q[1,\ldots,n])$. In the procedure we have one recursive call to $\textsc{IntersectionCount}(P[1,\ldots,n-1], Q[1,\ldots,n-1])$. Also, we spend $O(n)$ time to compute the number of intersection points on L_n. Therefore, we have:

$$T(n) = T(n-1) + O(n).$$

You can use recursion tree method (your recursion tree is actually a path now) to compute $T(n)$. Also, you can just expand the recursion as follows:

$$\begin{aligned} T(n) &= T(n-2) + O(n-1) + O(n) = T(n-3) + O(n-2) + O(n-1) + O(n) \\ &= \ldots \\ &= T(1) + O(2) + O(3) + \ldots + O(n) = O(1) + O(2) + \ldots + O(n) = O(n^2). \end{aligned}$$

Algorithm 2.

Idea: The main idea for speeding up the algorithm is to divide and conquer. First we divide the line segments into two sets $B = \{L_1, \ldots, L_m\}$ and $R = \{L_{m+1}, \ldots, L_n\}$, where $m = \lfloor n/2 \rfloor$. We refer to segments of B as blue segments and the segments of R as red segments. Note that on the p side every blue endpoint is on the left side of every red endpoint (see the figure for an example). Intersection points can be categorized as follows.

(A) Blue-Blue intersection point, both intersecting segments are blue.

(B) Red-Red Intersection point, both intersecting segments are red.

(B) Blue-Red Intersection point, exactly one intersecting segment is blue and exactly one intersecting segment is red.

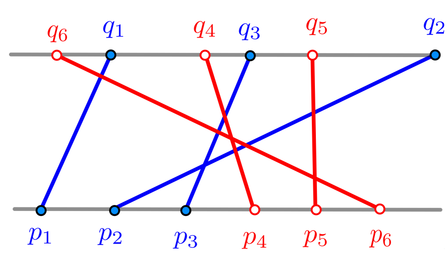

Figure 1: The algorithm recursively computes the number of Blue-Blue and Red-Red intersections, one and two, respectively. Then, it counts the number of Blue-Red intersections, six, in $O(n)$ time.

Our algorithm, $\textsc{FastCount}(P[1,\ldots,n], Q[1,\ldots,n])$, computes the number of Blue-Blue, and Red-Red intersection points via two recursive calls, $\textsc{FastCount}(P[1,\ldots,m], Q[1,\ldots,m])$ and $\textsc{FastCount}(P[m+1,\ldots,n], Q[m+1,\ldots,n])$, respectively.

It remains to compute the number of Blue-Red intersections. To that end, let $q_{\pi_1}, q_{\pi_2}, \ldots, q_{\pi_n}$ be the sorted list of $q_1, q_2, \ldots, q_n$ with respect to X-coordinates. (Recall that $q_1, q_2, \ldots, q_n$ is not necessarily sorted in the input). Note that $\pi_1, \pi_2, \ldots, \pi_n$ is a permutation of $1, 2, \ldots, n$, which can be computed as a side product of any sorting algorithm (How?). Specifically, we need (i) $q_1, q_2, \ldots, q_n$, X-coordinates of q's as they appear in the input, and (ii) $\pi_1, \ldots, \pi_n$, a permutation of $1, \ldots, n$ such that $q_{\pi_1}, q_{\pi_2}, \ldots, q_{\pi_n}$ is sorted.

To count the number of Blue-Red intersections our algorithm iterate through $q_{\pi_1}, q_{\pi_2}, \ldots, q_{\pi_n}$, and it keeps track of two counters: (i) b the number of blue segments with their q-endpoint not reached, and (ii) *brCount* the number of Blue-Red intersections counted so far. Initially, $b = m$, and $brCount = 0$. At step i, we consider q_{π_i} if it is blue, we decrement b by one, and if it is red we increment *brCount* by b. (Why?)

Pseudocode:

```
FastCount(P[1,...,n], Q[1,...,n])
    if n = 1
        return 0
    m ← ⌊n/2⌋
    a ← FastCount(P[1,...,m], Q[1,...,m])
    b ← FastCount(P[m+1,...,n], Q[m+1,...,n])
    c ← BlueRedCount(P[1,...,n], Q[1,...,n], m)
    return a + b + c
```

```
BlueRedCount(P[1,...,n], Q[1,...,n], m)
    π ← the permutation of sorted Q
\\Q[π[1]], Q[π[2]], ..., Q[π[n]] is sorted,
\\π can be computed as a side product of any sorting algorithm (Why?)
    b ← m
    brCount ← 0
    for i ← 1 to n
        if π[i] ≤ m \\Q[π[i]] is blue
            b ← b − 1
        else \\Q[π[i]] is red
            brCount ← brCount + b
    return brCount
```

Proof. We give a proof by induction.

Base case: If there is only one line in the set. This algorithm correctly reports zero number of intersections.

Induction Hypothesis: For any $N < n$, FastCount$(P[1, \dots, N], Q[1, \dots, N])$ correctly computes the number of intersections.

Induction Step: We show that FastCount$(P[1, \dots, n], Q[1, \dots, n])$ correctly computes the number of intersections among L_i's. As explained above, there are three types of intersections. Induction hypothesis implies that the number of Blue-Blue and Red-Red intersections are computed correctly (the values of a and b in the code.) It remains to show that BlueRedCount computes the number of Blue-Red intersections correctly. To see that let $L_i = (p_i, q_i)$ be a red segment. Since all blue segments are on the left side of all red segments on p-side, the number of blue segments that cross L_i is equal to the number of them that are on the right side of q_i on the q-side. This procedure basically, counts this number. □

Running time: Let $T(n)$ be the running time of FastCount$(P[1, \dots, n], Q[1, \dots, n])$. The running time of BlueRedCount$(P[1, \dots, n], Q[1, \dots, n], m)$ is $O(n)$ as there is a for loop with constant time operations on each iteration. Also, FastCount recurse on two subproblems each of half size. Thus, we have:

$$T(n) = 2T(n/2) + O(n) = O(n \log n),$$

similar to Merge Sort.