Austin Row and Jeremy Fischer
Dr. Nayerri
CS 325
October 5, 2016

Group Assignment #1

1

Θ(n^2) Algorithm

Description:

```
Intersections(lineSegments[1…n])
    If there is 1 or fewer elements in lineSegments:
        Return 0

    intersections = the number of intersections among previous n-1 line
                    segments (Intersections(lineSegments[1…n-1]))

    intersections = intersections + PrecedingIntersections(lineSegments[1…n], n)

    Return intersections


PrecedingIntersections(lineSegments[1…n],n)
    precedingIntersections = 0

    For each line segment in lineSegments[1…n-1]:
        If q_i ≥ q_n:
            Increment precedingIntersections

    Return precedingIntersections
```

Proof by Induction:

Claim 1: *PrecedingIntersections* finds and returns the correct number of intersections between the n^{th} line segment in lineSegments[1…n] and the n-1 line segments that precede it.

Base Case: If $n \leq 1$, *PrecedingIntersections* correctly returns 0.

Inductive Hypothesis: *PrecedingIntersections* increments precedingIntersections for all instances where the N^{th} line segment such that $N < n-1$ intersects the n^{th} line segment.

Inductive Step: The variable precedingIntersections stores the correct number of intersections between the n^{th} line segment and all of the line segments preceding the N^{th} such that $N < n-1$ (by the Induction Hypothesis). If $q_{n-1} \geq q_n$ then the n^{th} and $(n-1)^{th}$ line must intersect because $p_n > p_{n-1}$ (by the problem definition) and thus precedingIntersections is incremented and correctly represents the number of intersections between the n^{th} line and all lines before it.

Claim 2: *Intersections* correctly finds and returns the number of intersections between n line segments.

Base Case: For the sets $Q = \{q_1, q_2, \ldots, q_n\}$ and $P = \{p_1, p_2, \ldots, p_n\}$, if $n \leq 1$ then *Intersections* correctly returns 0.

Inductive Hypothesis: For $N < n$, suppose *Intersections* will return the correct number of intersections between the line segments composed of the points Q[1…N] and P[1…N].

Inductive Step: After the recursive call to *Intersections* on line 4, the number of intersections between the previous n-1 line segments has been found (by the Inductive Hypothesis). *PrecedingIntersections* then correctly finds the number of intersections between the n^{th} line and the preceding n-1 lines (by Claim 1) and thus after line 6 ‘intersections’ stores the number of intersections between all n line segments. This number is then returned.

Runtime Analysis:

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

2

$\Theta(n \log_2 n)$ Algorithm

Description:

```
Intersections(lineSegments[1…n], start, end)
    If there is 1 or fewer elements in lineSegments:
        Return 0

    midpoint = ⌈n/2⌉

    leftIntersections = Number of intersections between lines in
                        lineSegments[1…midpoint-1]
                        (Intersections(lineSegments[1…midpoint-1]))
    rightIntersections = Number of intersections between lines in
                         lineSegments[midpoint…n]
                         (Intersections(lineSegments[midpoint…n])
    totalIntersections = leftIntersections + rightIntersections +
                         BetweenIntersections(lineSegments[1…n], midpoint)
    Return totalIntersections
```

```
10  BetweenIntersections(lineSegments[1…n], midpoint)
11      Intersections = 0
12      Left = lineSegments[1, midpoint-1]
13      Right = lineSegments[midpoint, n]
14      SortingIterator = 1
15
16      While Left is not empty:
17          If Right is not empty and Right[1] q value <= Left[1] q value
18              Add the length of Left to Intersections
19              lineSegments[SortingIterator] = Right[1]
20              Remove Right[1]
21          Else
22              lineSegments[SortingIterator] = Left[1]
23              Remove Left[1]
24          Increment SortingIterator
25
26      Return intersections
```

Proof by Induction:

Claim 1: *BetweenIntersections* finds and returns the correct number of intersections between line segments in the sub-lists [1…midpoint-1] and [midpoint…n] of lineSegments and sorts lineSegments[1…n] by q values of its line segments.

Base Case: If Left is empty, the algorithm correctly sorts lineSegments[1…n] by doing nothing and returns the correct number of intersections between line segments in Left and Right.

Inductive Hypothesis: After the first iteration of the loop on line 19, *BetweenIntersections* correctly merges Left and Right back into lineSegments[1…n] so that the subarray is sorted by the q values of the line segments and correctly counts the number of intersections between elements in Left and Right.

Inductive Step: If Left is empty, *BetweenIntersections* works by the base case.

Otherwise if Right is empty, Left[1] is the line segment with the next smallest q value and so it is merged back into lineSegments at the correct index SortingIterator. Then it is removed from Left, SortingIterator is incremented, and the rest is solved by the Inductive Hypothesis.

Otherwise if Right[1] q value <= Left[1] value then Right[1] intersects all of the elements currently in Left[1] because its p value is greater than all of those in Left (by the problem definition) so K is added to Intersections where K is the number of elements currently in Left. Then Right[1], as the line segment with the next smallest q value, is merged back into lineSegments at index SortingIterator and SortingIterator is incremented. The rest is solved by the induction hypothesis.

Otherwise Left[1] must contain the line segment with the next smallest q value and so it is merged back in to lineSegments at the correct index SortingIterator. Then it is removed from Left, SortingIterator is incremented, and the rest is solved by the Inductive Hypothesis.

Claim 2: *Intersections* correctly finds and returns the number of intersections between n line segments.

Base Case: If there is 1 or fewer elements in lineSegments, then there cannot be any intersections and thus 0 is correctly returned.

Inductive Hypothesis: For N =$\lceil n/2 \rceil$, suppose *Intersections* will return the correct number of intersections between the elements in lineSegments[1….N-1] and lineSegments[N…n]

Inductive Step: After the recursive calls to *Intersections* on line 6 and line 7, the number of intersections between the line segments in lineSegments[1...N-1] and lineSegments[N…n] have both been calculated and stored in leftIntersections and rightIntersections respectively (by the Induction Hypothesis). The number of intersections between elements in one subarray and elements in the other is then calculated by *BetweenIntersections* and lineSegments is sorted by the q values of its elements (by Claim 1). The sum of leftIntersections, rightIntersections, and the result of *BetweenIntersections* is then returned as the correct total of intersections for the n line segments in lineSegments.

Runtime Analysis:

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$

$T(n) = 2T(n/2) + n$

$= 4T(n/4) + 2n$

$= 8T(n/8) + 3n$

$= 2^k T(n/2^k) + kn$ for $k > 0$ where k represents the depth of the call tree.

The base case is $T(1) = 1$ (when lineSegments has one element, it only requires a single return statement to solve the problem). To use this base case to solve for the runtime, $n/2^k = 1$ or $k = \log_2 n$. Making this substitution gives all everything in terms of n [1]:

$2^{\log n} T(1) + n\log_2 n$

$= n + n\log_2 n$

$= \Theta(n \log_2 n)$

References

[1] J. Erickson, "Jeff Erickson’s Algorithms, etc," 1999. [Online]. Available: http://web.engr.illinois.edu/~jeffe/teaching/algorithms/. Accessed: Oct. 1, 2016.