CS 560: Design and Analysis of Algorithms, Spring 20
Progr. Assignment: 2-D Maxima, Due: Thurs, April 16

Let x and y denote the two coordinate axes in the two dimensional space R^2. A point $p \in R^2$ is specified by its coordinates along the two axes: $p = (x(p), y(p))$. For $p, q \in R^2$, we say that p is *dominated* by q, if $x(p) \leq x(q)$ and $y(p) \leq y(q)$.

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in R^2. $p_i \in S$ is called a *maximal element* of S, if p_i is not dominated by any other element of S. The set of all maximal elements of S is denoted by $maxima(S)$. The *maxima problem* is: Given S, find $maxima(S)$.

A brute-force approach to solve this problem is as follows: Compare each point $p_i \in S$ against all the other points in S to determine if p_i is dominated by any of those points; if p_i is not dominated by any of them, add it to the output set $maxima(S)$. This algorithm takes $\Theta(n)$ time for each point p_i, for a total of $\Theta(n^2)$ time.

You will implement an efficient algorithm (**AlgorithmA**) that runs in $\Theta(n \log n)$ time. Your implementation must be in C++. AlgorithmA consists of the following steps:

I. Input: The point set S is in an input file. The first line contains the value of n (the number of points). Following that, there will be n lines, each line containing the x and y coordinates of one point. The points must be read and stored in an array $Points[1..n]$ of records (struct). The record $Points[i]$ corresponds to point p_i, and has four fields: the x and y coordinates (float), *maximal* (boolean), and *where* (integer). *Maximal* indicates whether $p_i \in maxima(S)$. The use of the *where* field will be explained latter; for now, initialize $Points[i].where = i$. Do not use $Points[0]$.

II. Sorting: Sort the points in *Points* according to their x-coordinates, and reindex them such that $x(p_1) \leq x(p_2) \leq \ldots \leq x(p_n)$. For each point, the *where* field should contain the index of the point in the original input. So, if the 7th point in the input (in Step I above) moved to the 3rd position in the array (after Step II), then you should have $Points[3].where = 7$. As you move the points during sorting, carry the *where* field with the points.

The sorting must be done using the MergeSort algorithm. It should be implemented as efficiently as possible, and as described in the class. Keep variables *SortCount* and *SortTime*.

SortCount counts the number of **key** comparisons performed by the MergeSort algorithm.

SortTime is the time taken by the algorithm; it should be incremented by 1
for each pass through any subroutine and any loop.

Print out *SortCount* and *SortTime* (**do not** print the sorted array).

III. Finding the Maxima: Process the points, one-by-one, in decreasing order of x-coordinates.
Note that $p_n \in maxima(S)$ since it has the largest x coordinate;

$p_{n-1} \in maxima(S)$ iff $y(p_{n-1}) > y(p_n)$,

$p_{n-2} \in maxima(S)$ iff $y(p_{n-2}) > max(y(p_n), y(p_{n-1}))$, and so on.

Suppose that, at some instant, you have processed the points $p_n, p_{n-1}, \ldots, p_{i+1}$.

Let $maxima[i + 1..n]$ denote the set of maximal elements among them.

All these points are in $maxima(S)$, because none of them can be dominated by any of the points $p_1, p_2, \ldots, p_i$ (because the latter have smaller x-coordinates).

Now we want to process p_i. p_i has a smaller x-coordinate than the points processed so far.

So, $p_i \in maxima(S)$ iff $y(p_i)$ is larger than the y-coordinate of any point in $maxima[i + 1..n]$;

i.e., $y(p_i)$ is greater than the y-coordinate of the last point q in $maxima[i + 1..n]$.

If $y(p_i) > y(q)$, set $Points[i].maximal$ to *true*; else ignore p_i.

Keep variables *MaxNumA*, *MaxCountA* and *MaxTimeA*.

MaxNumA is the number of elements in $Maxima(S)$.

MaxCountA counts the number of **key** comparisons performed during Step III.

$MaxTimeA$ is the time taken by step III; it should be incremented by 1 for each pass through any subroutine and any loop in Step III.
Print out $MaxNumA$, $MaxCountA$, $MaxTimeA$ and $Maxima(S)$. For each point in $Maxima(S)$, in **increasing** order of x-coordinate, print out its **original** index (i.e., the *where* field).

Your program should be modular, and contain appropriate procedures/functions. No comments or other documentation is needed. Use meaningful names for all variables.

You will run your program on 10 different sets of points; your program should have a loop for this. At the very end, print a table (one row for each point set) containing the following: $SortCount$, $SortTime$, $MaxCountA$, $MaxTimeA$, $SortCount + MaxCountA$.

All the 10 point sets are in the input file Points1; sample output is in the file maxima.out. I will provide you these two files. The name of your program file must be maxima.cpp.

IT IS YOUR RESPONSIBILITY TO PROTECT YOUR FILES, DIRECTORIES AND PRINT-OUTS. ANY ONE INVOLVED IN CHEATING/COPYING **(INCLUDING THE ONE WHOSE WORK WAS COPIED)** WILL BE SEVERELY PUNISHED.