COMP 251 Assignment 4

Prepared by Prof. Michael Langer
(with assistance from Balaji Purushotham)

Posted: Fri. March 14, 2014
Due: Sunday, March 30, 2014 at 23:59.

General Instructions

- The T.A.s handing the assignment are Mohammed Smaoui and Mingzhou Yang. Their office hours will be posted on the public web page. Office hours will be in Trottier 3110
- The other instructions are the same as in Assignments 1 to 3, in particular, the late penalty policy.
- Note that this assignment is out of 120 points rather than 100 points. It will be worth slightly more than Assignments 1, 2, 3.

Question 1 (90 points)

In lecture 14, you learned about the segmented least squares problem and how to solve it using dynamic programming. Here you are given a partial implementation, namely a class **SegmentedLeastSquares**. You need to complete the methods in this class. This will give you some hands on experience with dynamic programming.

You are required to complete three methods. There will be 30 points for each:

- **computeOptIterative()** - This method should compute the opt[ ] array using a for loop.
- **computeOptRecursive()** - This method should compute the opt[ ] array using recursion. Note that you need to use memoization to avoid recomputing the solutions many times.
- **computeSegmentation()** – This method does the backtracking. Given the opt[ ] values, it finds a segmentation that produced these opt[ ] values.

Submit the file **SegmentedLeastSquares.java.**

A few notes:

- You are also given a **GUITester** class for testing your code. It prints out information about the segments in the console. It also provides you with a GUI for visually examining your computed solution. There are radio buttons for switching between your iterative solution and your recursive solution. There is also a slider for adjusting the cost of each segment that you compute. Since the range of interesting costs will depend on the data set, we have also provided a field for you to adjust the range of costs covered by the slider.

  The interface is frankly a bit clunky. e.g. You need to hit return after entering a new integer into the field and you will also need to move the slider to get it to recompute a new solution for the new value of the cost of each segment, and you have entered the new range for the slider. But its good enough to get the job done.

- The method **computeEijAB()** efficiently computes the errors in segment (i,j). As mentioned in class, if you do this naively then it take O(n^3) namely for each (i,j) you need to sum quantities from i to j. However, re-doing all these summations for each (i,j) pair is overkill. I've given implementation that runs in O(n^2). The idea is that if you know the sums from 1 to j for all j, then you can compute the sum from i to j in O(1) time, simply by taking the sum to j and subtracting the sum to i-1. That should be enough of a description for you to have a look at the code.

Question 2 (30 points)

In the closest pair of points algorithm from lecture 16, the points are sorted by their x values and by their y values as an initial step, i.e. before the findClosestPoints recursion is called. Suppose we only sorted the x values but not the y values at this initial step and we only passed the sorted array X. To take advantage of the "next 7 y values" constraint, the findClosestPoints algorithm would need to sort the points by the y coordinate, giving array Y.

Write and solve the recurrence for the time taken by findClosestPoints if has to sort by y in each recursive call, as just described. Hint: this recurrence is not covered directly by the Master theorem seen in class, and so you will need to solve it without relying directly on that theorem (although the derivation is very similar to that used in proving the theorem).

Submit your solution as a PDF **Q2.pdf**. Do *not* waste time typesetting it to make it pretty. Instead, just write it out neatly by hand (one page only!) and scan it.
(http://www.mcgill.ca/library/services/scan-print-copy)
Please verify that the scan is easily legible, before you submit it.

Have fun, get started early, and good luck!