CSI 403: Programming Assignment

No late submissions will be accepted for this assignment (check Deadline in Brightspace).

Very important note: You MUST use the 3 java files provided and solve the assignment using the predefined Driver and Point classes. Submissions using other data structures and/or additional classes or libraries will be scored as 0 points. You must electronically turn in the three Java classes with implementations (follow the instructions in the comments.) along with a PDF (or MS Word) file that shows your analysis in part (b).

The list of files (INCLUDED IN THIS ASSIGNMENT) you should use are:

1) **Point.java**: defines the Point object and some comparators to support sorting. You should not change this class.

2) **ClosestPair.java**: contains method stubs for the two algorithms to be implemented

3) **ClosestPairDriver.java**: Contains tests to verify the correctness of your algorithms as well as a main() method. You need to implement the timing experiments required for part (b) of the assignment (see below)

Your three java classes should be compilable without errors using the standard Java 1.7 or higher compilers, using the following from a unix command line:

javac *.java

After successful compilation, one should be able to execute your driver class using the following command:

java -cp ../ closestpair.ClosestPairDriver

Solutions that do not compile or cannot be run will be scored as 0.

The objective of this programming assignment is to implement two different algorithms for the **closest pair** problem and experimentally compare their computation times.

In the closest pair problem, we are given a set of n points in the plane. Each point is specified by its x and y coordinates, which are real numbers. Recall that given two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, the (Euclidean) distance $d(p, q)$ between them is given by the formula

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

The goal of the closest pair problem is to find the pair of closest points (a pair that has the shortest distance).

A simple (brute-force) method of computing the shortest distance is to compute the $n(n-1)/2$ distances between all pairs of points and take the smallest of these distance values. Clearly, this

algorithm runs in $\Theta(n^2)$ time. For this problem, there is a divide-and-conquer algorithm that runs in $O(n \log n)$ time.

A description of this algorithm was discussed in class (see slides in assignment, also Chapter 33.4 of the book scanned in the assignments). You are required to implement both the brute-force algorithm and the divide-and-conquer algorithm and determine the actual times used by these two algorithms on test inputs. There are two parts to this assignment.

Part (a)[40 pts]: Implement the Brute Force and Divide-And-Conquer algorithms whose method stubs are provided in ClosestPair.java. **Do not change other methods or classes or the signatures of these functions.** Consult the lecture slides for the pseudocode and analysis justifying the $\Theta(n^2)$ and $O(n \log n)$ growth rates respectively. Beyond compilable, all tests defined in the driver class should pass with success.

Part (b)[60 pts]: In this part, you are required to experiment with inputs of sizes 10,100,1000 and 10000 by employing the random point generator implemented in ClosestPairDriver.java. To do this analysis, you should implement the runnningTimeComparison() method stub the ClosestPairDriver.java (follow the instruction in the code comments). Report the computation times of the two algorithms as discussed below.

Using the outputs produced by your algorithms, you should prepare a table with four rows (one corresponding to each size) and three columns labeled No. of points, Time used by Brute Force and Time used by Divide-and-Conquer. Plot this table (using MS Excel or any other software for plotting) in a line chart with x axis the number of points and y axis: time taken by an algorithm. You will have two curves: one for each of the algorithm. Plot also in the same figure the functions $f(n) = c_1 n^2$ and $g(n) = c_2 n \log n$ functions for the same range of number of points $n = (10, 100, 1000, 10000)$, by choosing appropriate constants c_1 and c_2 to fit visually your observed running times. Discuss what you observe about those constants. Do they matter for small n? How about, large n?

The table, your explanation and plots should be submitted as a separate PDF or MS word file (Do not use other formats. If you have to, export to PDF before submitting).

All files should be submitted to Brightspace no later than the deadline.