CPSC 3620 Course Project
Fall 2019

The Closest-Pair Problem

Due: TBA

The total number of marks is 80.

The project can be done by a group of at most *three* (3) members. It is your responsibility to distribute the workload among your group buddies. All the members in the same group will be awarded the same score.

In our lectures, we have discussed the *brute-force solution* and the *divide-and-conquer solution* to the *closest-pair problem*. This project asks you to implement those two solutions and conduct some experiments to show that indeed the latter solution is better than the former one. The details of the project are as follows.

(A) Programming (35 marks)

The language that can be used is C/C++. The programming work in this project contains two components.

The first component is to implement the brute-force solution to the closest-pair problem, while the second component is to implement the divide-and-conquer solution. Both of them take *n* points (see below) and output the pair of points which have the shortest distance between them. For each component, create a separate program for it. The program containing the implementation of the brute-force solution is called *bruteforce.c/ bruteforce.cpp*, while the one for the divide-and-conquer solution is called *divideconquer.c*/*divideconquer.cpp*.

Each of the *n* points has the format (*x*, *y*), which *x* and *y* are integers within the range [-10000, +10000]. You need to create a small program, called *create.c* or *create.cpp,* to generate such *n* random points and output them into a file called *output**n**.txt*, where *n* is the number of points. For instance, if *n* = 800, then the output file is output800.txt. Your main program will read this file for input points.

You may find that it is useful to know functions, such as *srand*(…), *rand*(…), etc. in C/C++.

You should have a good programming style in your implementations.

(B) Experiments (15 marks)

Change *n* from 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 2500, 3000, 4000, 5000, 6000, 7000, 8000, 9000, and 100,000. Run your implementations of the two solutions on each choice of *n*. For instance, if *n* = 800, then you will create 800 points and run your implementations of the two solutions on them.

For each choice of *n*, report how much time your implementation for each of the two solutions is spending to find the closest pair.

Some notes are in order.

(1) As a safeguard, always compare the results from the two solutions as whether they are equal to each other or not.
(2) Generally, the time from the divide-and-conquer solution is shorter than the one from the brute-force solution.
(3) You can use *printf* or *cout* for your debugging purpose. But once your implementations become stable, suppress them (except for the ones to report time and final output), since calling to them frequently makes your time measurement inaccurate.
(4) You may find it is useful to know functions, *time*(…), *clock*(…), *clock_gettime*(…), etc. You can choose second, millisecond, microsecond, or nanosecond as your time unit.
(5) In order to have a more accurate measurement on the time your implementations are spending, keep optimizing your code and removing any unnecessary parts.

(C) Submission and demonstration (30 marks)

Your hardcopy report should include, but not limited to, the following points.

(1) Create a cover page containing the title and group members' names.
(2) Create a chart for each solution to the closest-pair problem regarding the running times over different choices of *n* (as discussed above). You can use *Microsoft Excel* or *OpenOffice* in Linux for this purpose. The *x*-axis represents the number of points while the *y*-axis is the corresponding running time.
(3) Based on the two charts, argue and discuss whether your implementations match the theoretical analysis on the time complexity of the two solutions, as discussed in class.
(4) Discuss the main variables, data structures and algorithms, including the brute-force solution and divide-and-conquer solution, in your implementations.
(5) Discuss what you have learned from doing this project and what more you can do to further enhance your implementations.
(6) Discuss any other specific issues that will help appreciate your efforts in this project.

Also create a *readme.txt* file, which contains the instructions for compiling your source files, running executables, a name list of auxiliary files, such as header files (if you have them in your implementations), and a list of known bugs (if your implementations have any).

Some notes on your submissions.

(1) Submit a hardcopy of your report.
(2) Submit a hardcopy of your *readme.txt* file.
(3) Submit a USB stick containing: (a) Makefile that can be used to compile your

source programs, (b) *bruteforce.c* (*cpp*), *divideconquer.c* (*cpp*), *create.*c (*cpp*), and any auxiliary header or implementation files, if there are any; (b) all the point files for your experiments, e.g., *output100.txt*, *output200.txt*, etc.; and (b) the two Excel or OpenOffice files containing the resultant charts.

(4) Prepare a 15-minute demo for your project. The time and location will be announced shortly.

Final notes:

There is no strict requirement on the report's format and page number. But your presentation should be clear and concise.
All your programs should work on the lab linux machines.
The USB will be returned to you once your project is marked.