MA/CSSE 473 – Design and Analysis of Algorithms

Convex Hull Implementation Comparison (100 points)

This is an individual assignment. You may of course get help with design and debugging from other students, but all of the code should be yours. I am giving you several weeks to do this, so you can work on it whenever you have some time. I will not pause the regular written assignments at the time this is due, so you should try to put a bit of time in on this each week. Submit to the Convex Hull drop box on ANGEL.

Your implementations should be in a language (or combinations of languages) that I can easily compile and run. These languages include Java 6, C, C#, Python 3 (with zellegraphics, pygame, or tkinter), Javascript embedded in HTML, and *Chez* Scheme plus SWL. If you want to use another language, ask me in advance about it. If you use a graphics library that is not a standard part of the language you are using, please give very clear instructions for (getting and) using the library.

Implement and compare the running times of two Convex Hull algorithms from the textbook:

1. the brute force Convex Hull algorithm found on pages 111-112 of the Levitin textbook.
2. the quickhull algorithm found on pages 150-154 of the same book.

Your program should ask the user for the name of an input file, and a choice of which calculation method to use. It should read the points from the input file, and time how long it takes to calculate the convex hull. It should then output the running time and the list of the points that make up the convex hull, and finally draw all of the points, the convex hull points (in a different color) and the convex enclosing polygon that connects the convex hull points. Note that the input and output operations are not part of the time that you should measure.

Input file format: A text file containing integer coordinates of a collection of points, at most 10 points per input line. The coordinates of any points in files that I give you will be non-negative integers that are small enough to be displayed in a 1280×1024 window.

Here is sample input containing 25 randomly-generated points (in the file **25.txt**):

```
(102,212), (241,196), (154,80), (350,200), (333,308), (215,169), (116,249), (200,400), (147,109), (150,60)
(225,100), (30,40), (171,140), (198,357), (319,494), (330,500), (350,400), (309,403), (229,302), (291,242)
(170,193), (20,140), (310,500), (178,86), (159,310)
```

On the next page I demonstrate an output window showing the points and their convex hull (in the file **25.png**). Note that this display uses standard graphics coordinates, where the origin is at the upper left, and positive y direction is downward. http://www.rose-hulman.edu/class/csse/csse473/201110/Homework/ConvexHullFiles/ contains some larger input files, and another corresponding output window. All of these collections happen to have the same convex hull. I may run your code with some different input files (same format) for testing purposes. You may want to create some files of your own of different sizes in order to facilitate your analysis.

You should do multiple runs of each algorithm with various sized input files, and use the timing data that you collect to determine whether your implementation of algorithms seems to conform to the big-theta running times described in the Levitin book.

Submit a ZIP or RAR archive file (whose name includes your username) that includes at least the following:

1. Your well-documented source code. If you use Eclipse or Visual Studio to create the code, please include the entire project (the project's name should include your username).
2. A README.txt file that contains any instructions needed to compile and run your file (including where the input files need to be). You should not include any of my large input files, since they will use a lot of ANGEL's disk space. You may include smaller files (1250 points or smaller) to make it easier for me to run your code.
3. Screen shots of the drawings from two or three runs of your program (not the "25 points" run) .
4. A document that summarizes and analyzes your timing results. You should include a graph of file-size *vs.* calculation time, including attempts to fit the points to appropriate curves. How large a file can you process in a reasonable amount of time with each algorithm? NOTE: Depending on the graphics environment that

you use, you may find (for quickhull) that calculation is fast, but drawing takes too long or runs out of heap space. If that is the case, try running for large files just to get the timing information; don't call your drawing code at all.

5. A brief document stating that you did no receive code from anyone else or share your code for this project with anyone else.