MA/CSSE 473 – Design and Analysis of Algorithms

Homework 5 and ConvexHull problem

HW 5 39 points total

(Summer: in drop box) These are to be turned in as hard copy. You can write solutions out by hand, or write them on your computer and print them. If there are multiple pages, please staple them together.

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

3.2.2 (best/worst case for sequential search)
3.2.9 (Substrings that begin/end with specific characters) Brute force is $\Theta(N^2)$
3.3.2 (closest straight-line post office)
3.4.2 (Hamiltonian Circuit)
3.4.9 (Magic Squares)

Problems to write up and turn in:

1. (10) 3.2.3 (Gadget drop) Let N be the total number of floors, and F the number of the lowest floor on which a gadget fails when dropped from there.

   Assume that due to weight, volatility, or some other factor, there is a high cost (C) for each floor that a gadget must be carried up. Also a cost (T) for each test to determine whether the drop caused the gadget to fail after each drop.

   First, give big-Theta worst-case running times in terms of N, C, and T for the obvious algorithm that tries every floor in succession (that algorithm only requires one gadget). Then design and analyze a the most efficient algorithm you can).

   Can you think of any flaws in this general approach to testing?
2. (2) 3.2.4 (String matching count)
3. (5) 3.2.6 (String matching worst case). Searching for a pattern of length n in a string of length m.
4. (6) 3.3.3 Points(:3 , 1, 2) (Manhattan distance). Clarifications: (b) You can sketch or simply list them. (c) By "a solution", they mean "an algorithm to solve the problem".
5. (3) 3.3.5 (brute-force k-dimensional closest-point algorithm)
6. (2) 3.3.8 (dealing with collinear points in brute-force complex hull algorithm)
7. (4) 3.4.1 (traveling salesman analysis)
8. (2) 3.4.5 (simple cost matrix for an assignment problem whose optimal solution does not include smallest element)
9. (5) 3.4.6 (partition problem)

(Continued on next page)

Implementation problem

(50) 3.3.9 (Convex Hull implementation)

Implement the brute force Convex Hull algorithm found on pages 110-112 of the Levitin textbook. Submit it to the ConvexHull drop box on ANGEL

Your implementations should be in a language (or combinations of languages) that I can easily compile and run. These languages definitely include Java 6, C, C++, Python 3, and Scheme. If you want to use another language, ask me in advance about it.

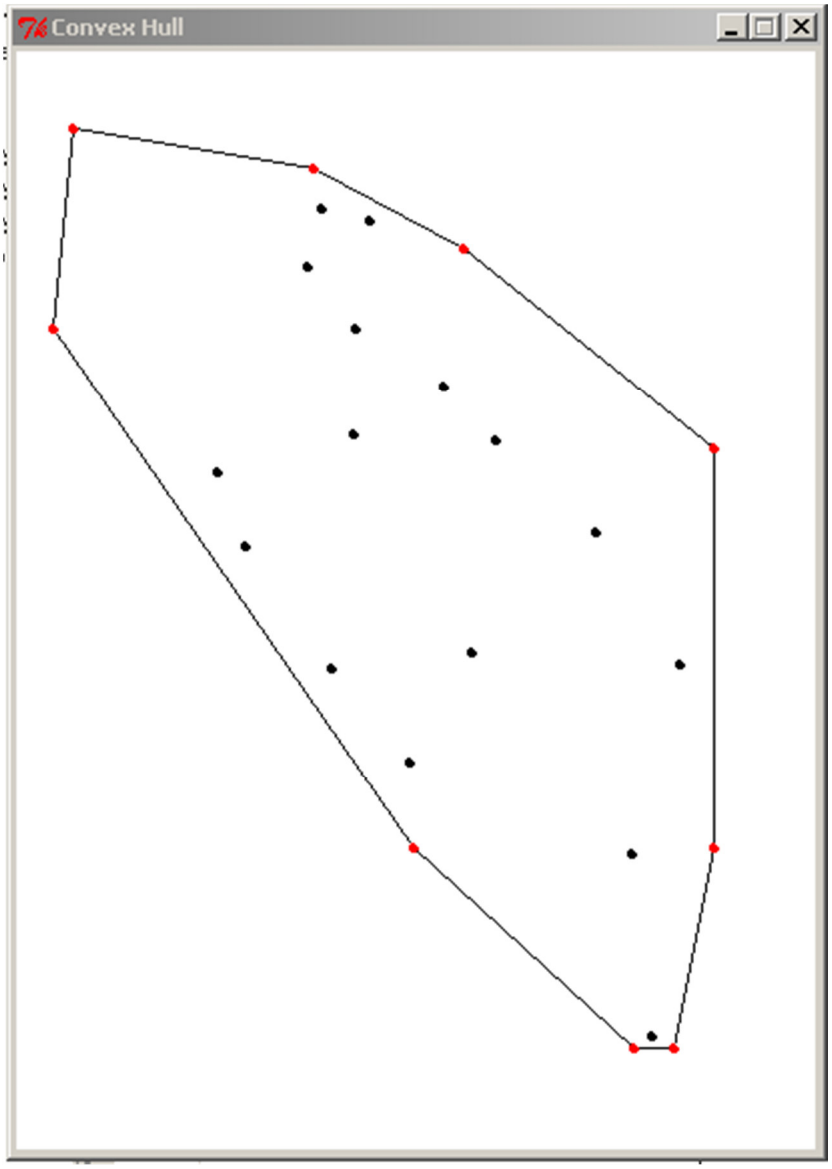
For input, your program should read a textfile containing integer coordinates of a collection of points (no duplicates), at most 10 points per line. The program should calculate the convex hull of this collection of points, and print it to standard output. Your program should also draw the points and the convex hull, similar to the example here. It is okay if your program writes a file that is then read by another program (perhaps even Excel) that can do the drawing. I found that our ZelleGraphics module from CSSE 120 was quite sufficient to do the drawing.

There are some sample input files in http://www.rose-hulman.edu/class/csse/csse473/201040/Homework/ConvexHullFiles

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)
```

and an output window showing the points and their convex hull (in the file **25.png**)



You should submit a ZIP file containing your code and the results of sample runs. Package your code in a way that makes

it as easy as you can for me to actually run the code once it is unzipped. For example, if you use Eclipse, include the entire Eclipse project.

Also include in your ZIP file a document (or documents) that include(s):

- Your code (8 point font is fine)
- Instructions for running the code (if it is not obvious from context)
- Results of runs on some of my input files
- Approximate execution times of your program for various size inputs.
- An indication of which is the largest of my files that your program can run in under 2 minutes.

Note that my display uses standard graphics coordinates, where the origin is at the upper left, and positive y direction is downward. .

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.