**Computing and Algorithms III: Program 1**

**CS-203** **Summer 2013**

---

**Assignment Weight: 1.0**

---

## Functional Requirements

Write a Java program which takes one or two arguments as input.

- If only one argument is present, the argument will be the name of a file in the current directory. Each line of the file will contain two real numbers, separated by whitespace, corresponding to the x- and y-coordinates of a point in the two-dimensional plane. (The number of lines in the file will be unknown.)
- If two arguments are present, the first argument must be the literal string "-debug". The other will be the name of the datafile, as described above.

After reading the file and storing the data in an appropriate data structure, the program will compute the convex hull of the specified points using the brute-force algorithm (see Levitin, Section 3.3).

If the "-debug" flag has been specified at startup, the program will print out various information (both for your debugging purposes and for the instructor's grading purposes). As each pair of points is considered by the algorithm, the program will: print out the pair of points under consideration, the line formed by those points, information on which (or how many) points fall on each side of that line, and the final decision made on that pair of points. If the "-debug" flag is not specified, none of this information will be printed.

After the convex hull has been computed, the program will print out (in either case) the points which appear in the convex hull, and the total time used in finding the hull. Both of these should be printed in a "readable" format. The program should print out this information whether or not the "-debug" flag is specified.

## Design Requirements

As noted above, your program should use the brute-force algorithm (Levitin, section 3.3) to find the convex hull.

In computing the time used in finding the hull, you should use `System.nanoTime()`. This method returns a `long` value representing the current time, relative to some unknown starting value. Successive calls to `System.nanoTime()` allow one to measure the elapsed time between calls, *e.g.*:

```
long startTime = System.nanoTime();
doTheAlgorithm();
long endTime = System.nanoTime();
long elapsedTime = endTime - startTime;
```

Your timing should specifically exclude the time required to initialize the program (*i.e.* reading the input file into your internal data structures) as well as the final output tasks (*i.e.* printing out the final results). That is, your timing should only measure the time taken to find the solution.

You are encouraged to use the built-in Java data structures whenever possible to simplify your code. In particular, note that there are various data structures which implement lists and sets. (These classes may not provide the most efficient implementations of these datatypes for these algorithms; nevertheless, they will suffice for our purposes.) Of course, you may need to implement additional classes in order to use these built-in data structures. (You rarely get something for nothing . . . )

Your program should use "good style". See the separate handout on style requirements for CS-203 programs. In particular, note that you should describe the algorithms you implement in sufficient detail to demonstrate your deep understanding of the algorithms in question.

## Additional Requirements

Additionally, you should create the following ordinary text files:

- README: Information on how to compile and run your program.
- ANALYSIS: An analysis of the running time of your program, with both theoretical and empirical analysis.

  In particular, for your empirical analysis, you should run your algorithm multiple times, using multiple different input sizes. (Of course, those runs should not be performed with the "-debug" flag, since printing will obscure the true time required by the program.) List those results of your empirical runs in your file. Are the empirical results you see consistent with your theoretical analysis? Explain why (or why not).

## Submitting Your Program

Before 11:59:59 p.m., Thursday, 1 August 2013 (3rd Thursday), you must must upload a zip archive to the course Blackboard assignment for Programming Assignment 1. This zip archive must contain all source code files for your program, including a class named `Prog1` with a `main` method.

No code printouts are required.