CSC 440 Assignment 2: Convex Hull

Out: Thursday, Feb 13th

Due: Tuesday, Feb 25th, by 11:59PM

Introduction

In this assignment, you will be implementing a convex hull algorithm. **Specifically, you must implement the divide-and-conquer algorithm for computing a convex hull.**

You will be provided with a zip file on EdStem, convexhull.zip, which contains a GUI called `draw_hull.py`. This GUI allows you to click in a window to add points, click a button to compute and draw the convex hull around those points using the algorithm that you have implemented. note that the "points" are not points in the mathematical sense; They have some size to them due to implementation details of the GUI, so the convex hull actually goes through the center of each one. You can run this GUI by executing from the terminal `$ python draw_hull.py` (or `$ python3 draw_hull.py`).

In the same zip file, there is some starter code in `convex_hull.py`, which is the file you will ultimately submit to GradeScope. It contains several functions that you will find useful (and should **not** modify). You will have to implement the `compute_hull` function which takes as input a list of points and returns only those points that are on the hull, in clockwise order. Currently, `compute_hull` is incorrect; it simply returns the input instead of the actual convex hull.

In this implementation, we represent a point as a 2-tuple of integers `(x, y)`. As is usually the case with computer graphics, the upper-left of the drawing canvas is the origin `(0, 0)`, the positive x-axis is directed rightwards, and the positive y-axis is directed downwards. As a result of this, all coordinates are positive in this implementation.

You must implement `compute_hull` and then benchmark its running time.

You may write any helper functions and/or classes you deem useful; they should all live in `convex_hull.py`.

You should also write a separate function for your *base case*, which should be the naive algorithm. Call it whenever you need to compute the hull on fewer than something like 5 or 6 points.

You also need to benchmark your code. For this purpose, we have provided some starter code in `benchmarks.py`. Report the time taken by your implementation on inputs of various sizes. Draw a plot and discuss the shapes of any curves that you see.

I have given you some trickier functions for the trickier geometric tasks, including a function for sorting a set of points counter-clockwise. Even without those, what you still have to implement took me roughly 6 hours. **Start now.**

For this assignment, your solution must be in Python 3.

Pair Programming

You should work with a partner for this entire assignment. Please refer to the pair programming policy in the syllabus.

Lateness

Submissions will not be accepted after the due date, except with the intervention of the Dean's Office in the case of serious medical, family, or other personal crises.

Grading Rubric

Your grade will be based on three components:

- Functional Correctness (50%)
- Design and Representation (15%)
- Invariant Documentation (10%)
- Benchmarking and Analysis (25%)

Functional Correctness means passing tests on GradeScope. Remember that your function `compute_hull` must return the points in the convex hull *in clockwise order*. We have provided you with a minimal test-suite that uses the `hypothesis` library for some automated testing. We do not provide any specific test cases, so you would be wise to write some tests for yourself. You should use the GUI for small tests, and for building your intuition.

Note: you may modify `draw_hull.py` in any way you see fit to assist with debugging. That file need not be submitted to GradeScope.

The file `convex_hull.py` is what must be submitted to Gradescope, along with your benchmarking and analysis PDF.

Design and Representation is our evaluation of the structure of your program, the choice of representations (de-emphasized here because some choices have already been made for you), and the use of appropriate (not excessive) comments.

Invariant Documentation is to get you to reason about the correctness and running-time of the algorithm.

- Invariant: A statement that can be checked at any point in time. It should relate to how the algorithm makes progress.
- Initialization: how is the problem set up
- Maintenance: how do I know I'm making progress?
- Termination: how do I know I'm done?
- Usually, these should be closely related.
- Properly documenting invariants can save you a great deal of time thinking about your algorithm.
- ***The wise lumberjack takes time to sharpen the axe.***

Benchmarking and Analysis is similar to assignment 1. You should include a PDF in your upload to GradeScope, along with your code.

- You should **not** use the GUI for benchmarking. The GUI is there for you to play with, but proper benchmarking should be non-interactive.
- You must *also* benchmark your base-case (which should actually work for any reasonably small number of points) and plot its running time.
- Benchmark your implementation on a wide enough set of inputs that you can plot a meaningful curve.
- Write a brief summary of your benchmarking results. Do they support your expectation of the asymptotic complexity of this algorithm? Why or why not? What about the naive (base case) algorithm?

Tips and Pitfalls

- Depending on your system, you may need to install a version of python3 that includes Tk
  - Linux (debian derivatives such as Ubuntu): `sudo apt-get install python3-tk`
  - macOS (with Homebrew): `brew install python-tk`
  - more info here: https://stackoverflow.com/questions/25905540/importerror-no-module-named-tkinter/25905642#25905642
- You may find it useful to modify `draw_hull.py` so that it puts something other than the Rams logo on the canvas. What data might be more helpful to you?
- One common pitfall is to end up with two points having the same x-coordinates in two different hulls. The merge algorithm will fail in this case! Make sure that this never happens when you divide an input for the recursive calls.
- Remember that the intercept between two parallel lines does not exist. If you're seeing divide-by-zero errors, check this first.

- Your base case should work for any small-ish number of points. You may find it's easier to stop the divide-and-conquer when you reach a set of 5 or 6 points, rather than going all the way down to 3, as this can help avoid some difficult edge cases.
- If you have a collection of collinear points that are all on the hull, *include them all* on the hull, rather than choosing only the extremal points.

Property-based testing

This assignment can ***drive you insane*** because it can be hard to reproduce a failing test, and the auto-grader gives you little feedback.

Should you wish to preserve your sanity, use unit tests. I recommend property-based testing using the `hypothesis` framework. Why is `hypothesis` so great? Because not only will it try to find an input that breaks your solution, but it will then try to shrink that input to the *smallest* input it can find that breaks your solution!

We have provided some basic tests in `tests.py`. You should first install the `hypothesis` library using the command `$ pip install hypothesis`

Run the tests by executing the command `$ python -m unittest discover`