CSC 440 Assignment 2: Convex Hull

Out: Thursday, Feb 6th

Due: Monday, Feb 17th, 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 Sakai, convexhull.zip, which contains a tkinter GUI called `hullGUI.py`. This GUI allows you to click in a window to add points, and then click a button to compute and draw the convex hull around those points (note that the "points" are not points, and thus have some size to them, so the convex hull actually goes through the middle of each one). Also in this zip archive is `convexhull.py` which is the file you will ultimately submit to Gradescope. It contains several functions that you will find useful (and should not modify), as well as a function, `computeHull(points)`, which takes a list of points and returns a list of points in the convex hull (in clockwise order). Currently, `computeHull(points)` is incorrect; it returns all the points, not just those in the convex hull.

The representation of a point in this implementation is as a 2-tuple of integers, i.e. `(x,y)`. As is usually the case with computer graphics, the upper-left of the drawing canvas is the origin `(0,0)`. As a result of this, all coordinates are positive.

You must correctly implement `computeHull(points)` and then benchmark its running time.

You may of course write any helper functions you deem useful; they should also live in `convexhull.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'll also benchmark this naive implementation on larger inputs.

I have given you some of the functions I wrote for the trickier geometric tests, including a function for sorting a set of points counter-clockwise. But, what you

still have to implement took me roughly 6 hours. **Start now.**

For this assignment, your solution must be in Python.

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 `computeHull(points)` must return the points in the convex hull *in clockwise order*. We do not provide a test case for you on this assignment (though you are welcome to use the GUI provided for small tests). You would be wise to write your own test routine and a way to programmatically generate points; you'll need a way to benchmark your program, anyways.

Note: you may modify hullGUI.py in any way you see fit to assist with debugging. `hullGUI.py` need not be submitted to Gradescope.

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 running time of the algorithm, as well as its correctness.

- 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 all 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, but you need to come up with reasonable inputs. You should include a PDF in your upload to Gradescope, along with your program.

- Implement a way to generate a certain number of points (for example, if you decide to benchmark on a million points, you'd want a way to generate those points at random). You should **not** use the GUI for this. 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

- You may find it useful to modify `hullGUI.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!
- Remember that the intercept between two parallel lines does not exist. If you're seeing divide-by-zero errors, check this.
- 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.

Property-based testing

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

So, to preserve your sanity, you should use unit tests, and 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 falsifies your solution, but it will then try to shrink that to the *smallest* input it can find that falsifies your solution!

Consider the following:

```
from hypothesis import given
import hypothesis.strategies as st

"""
generates a random list of coordinates in the range (0,0) to (1000000,1000000)
where each list has at least three points,
and prohibiting duplicates.
one weakness: this is unlikely to generate collinear points
(it is possible but improbable), so
a test that includes collinear points might also be useful
"""
@given(
    st.lists(st.tuples(st.integers(0,1000000),
        st.integers(0,1000000)), 3, None, None, True)
)
```

```
"""
returns true if and only if hull is a proper
convex hull for points.
This is determined by consulting every line segment
on the hull, and verifying that every other point
(not the two endpoints of the line segment)
is on the same side of that line segment.
"""
def checkHull(hull, points):
    for i in range(0, len(hull) - 2):
        j = i + 1
        p = hull[i]
        q = hull[j]
        pos = 0
        neg = 0
        for r in points:
            if r==p or r == q: continue
            if cw(p,q,r):
                neg += 1
            elif ccw(p,q,r):
                pos += 1
        if (pos == 0 or neg == 0):
            return True
        else:
            return False

def test_hull(points):
    hull = computeHull(points)
    assert checkHull(hull, points)

if __name__ == "__main__":
    test_hull()
```