CPSC 335 Project 1

Spring 2015
Prof. Kevin Wortman, CSU Fullerton

Introduction

In this project you will design, implement, and analyze straightforward greedy algorithms for two problems. For each problem, you will design an algorithm, describe your algorithm using clear pseudocode, analyze it mathematically, implement your algorithm in Python, measure its performance, compare your experimental results with the efficiency class of your algorithm, and draw conclusions.

The hypothesis

This experiment will test the following hypothesis:

> *For large values of n, the mathematically-derived efficiency class of an algorithm accurately predicts the observed running time of an implementation of that algorithm.*

The problems

Both problems involve processing sets of (x, y) points in the Euclidean plane.

The *bounding box* problem is:
input: a list P of n two-dimensional points (x, y)
output: four coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ such that, for any $(x,y) \in P$,

$$x_{min} \leq x \leq x_{max}$$

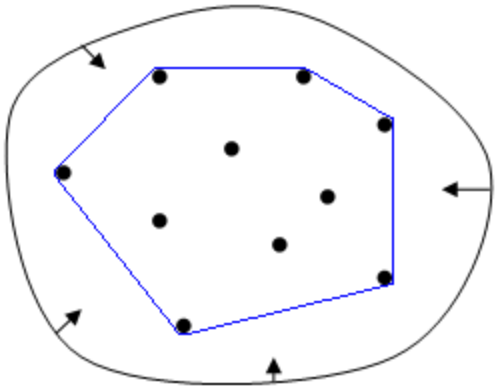
and

$$y_{min} \leq y \leq y_{max}.$$

There is a straightforward greedy algorithm that solves this problem in $O(n)$ time.

The *convex hull* problem is
input: a list P of n two-dimensional points (x, y)
output: a list of points H from P, such that H are the vertices of a convex polygon that contains all points in P

This diagram[1] shows how you can visualize forming a convex hull by stretching a rubber band around all the points, then releasing it and allowing it to conform to the most exterior points.



This instance of the convex hull problem has $n = 10$ input points, and the solution H is the 6 points along the blue outline of the hull.

The Wiki page for convex hulls has some additional background information.

A naïve convex hull algorithm

There is a (somewhat) straightforward greedy algorithm that solves this problem in $O(n^3)$ time. It is based on the following fact about convex hulls.

Claim: If $p,q \in P$, $p \neq q$, and l is the line passing through p and q, and the number of points of P strictly above l is either zero or $|P|-2$, then p and q are both on the convex hull boundary of P.

This claim is based upon the definition of what a convex hull is. By definition, all points in P are either inside the hull, or on its boundary. Consider two specific points in P, and form the line l passing through them. If there are input points on both sides of l, then l is not along the boundary of the hull. However if all of the points (except p and q) are on one side or the other of l, then l is indeed on the hull boundary, so p and q belong in H. Note that we need $p \neq q$ so that a proper line is defined between the two points.

[1] "ConvexHull" by Maksim (original); en:User:Pbroks3 (redraw) - http://commons.wikimedia.org/wiki/Image:ConvexHull.png. Licensed under Public Domain via Wikimedia Commons

```
def naïve_convex_hull(P):
     H = []          # points on the hull boundary
     for p in P:
          for q in P:
               if p != q:
                    l = <THE LINE PASSING THROUGH p AND q>
                    k = <THE NUMBER OF POINTS ABOVE l>
                    if k == 0 or k == |P|-2:
                         if p not in H:
                              H.append(p)
                         if q not in H:
                              H.append(q)
     return H
```

This is a partially-finished algorithm design. In particular, the `<THE NUMBER OF POINTS ABOVE l>` step is unclear, and you will need to expand that into clear pseudocode. That will probably involve testing each point to see whether it is above, colinear with, or below l. There are two conventional ways of doing this; one involves putting the line in slope-intercept form, and the other involves using a cross product.

I expect that counting `<THE NUMBER OF POINTS ABOVE l>` takes $O(n)$ time, as does the `p not in H` and `q not in H` expressions. So the total time efficiency of your convex hull algorithm will probably be $O(n^3)$. I am very aware that there are faster algorithms for this problem, but do not implement any of those. The whole point of implementing this algorithm is to empirically analyze an $O(n^3)$-time algorithm, not to solve convex hulls as efficiently as possible.

What to do

Repeat the following for each of the two problems:

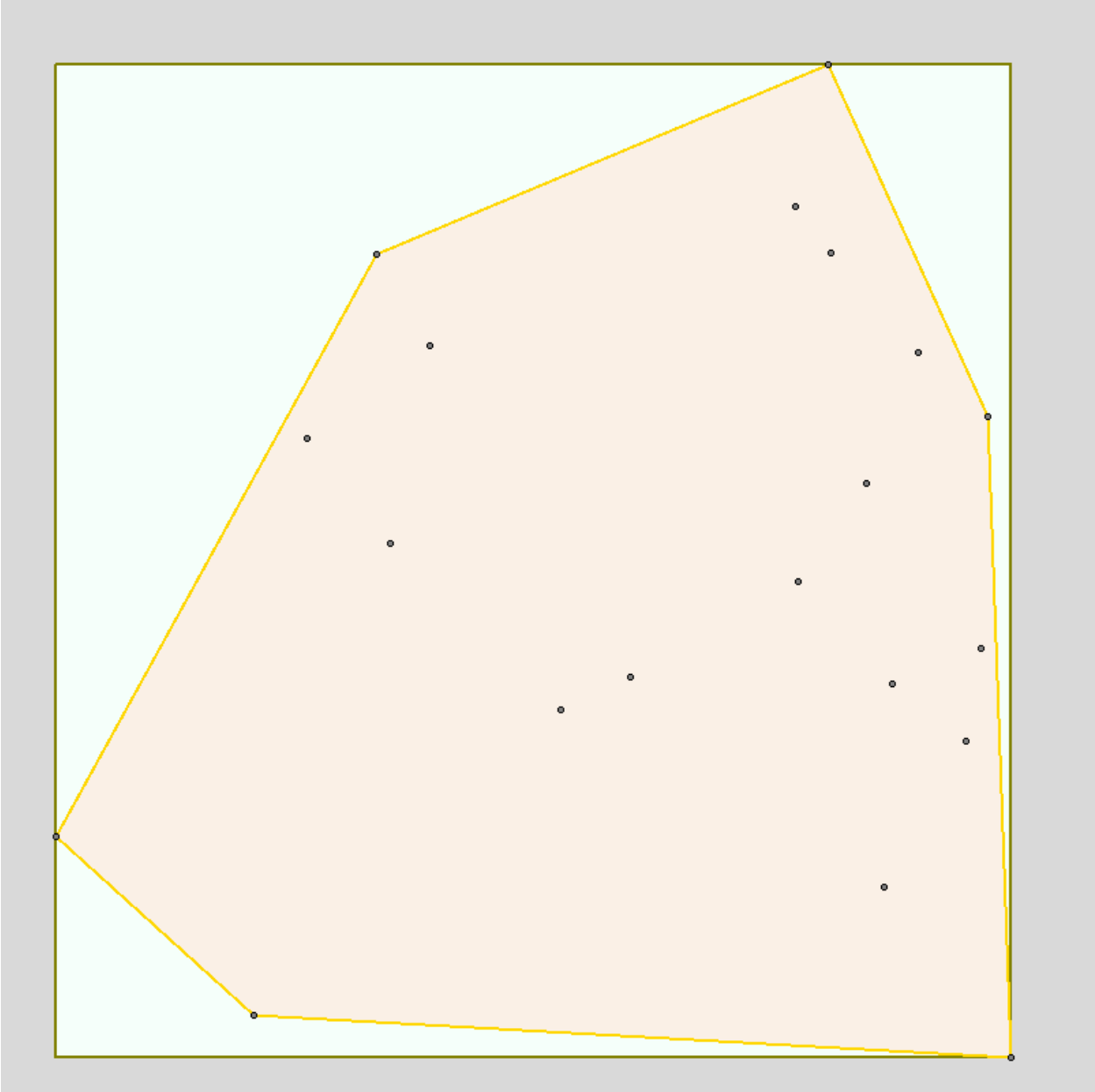
1. Design a greedy algorithm solving the problem.
2. Write clear pseudocode for your algorithm.
3. Analyze your pseudocode mathematically and prove its efficiency class. *Hint:* I expect the efficiency classes to be $O(n)$ an $O(n^3)$ respectively.
4. Implement your algorithm in Python 3, starting from the provided stub code.
5. Gather empirical timing data by running your implementation for various values of n. As discussed in class, you need enough data points to establish the shape of the best-fit curve (at least 5 data points, maybe more), and you should use n sizes that are large enough to produce large time values (ideally multiple seconds or even minutes) that minimize instrumental error.

6. Draw a scatter plot and fit line for your timing data. The instance size n should be on the horizontal axis and elapsed time should be on the vertical axis. Your plot should have a title, and labels and units on both axes.
7. Conclude whether or not your empirically-observed time efficiency data is consistent, or inconsistent, with your mathematically-derived big-O efficiency class.

I have provided you with a Python 3 source file, `project1_stub.py`, that you can use as the basis for your code. As explained in the comments, you need to rewrite the `bounding_box`, `convex_hull`, and `main` functions. You should be able to leave the rest of the code unchanged.

Sample output

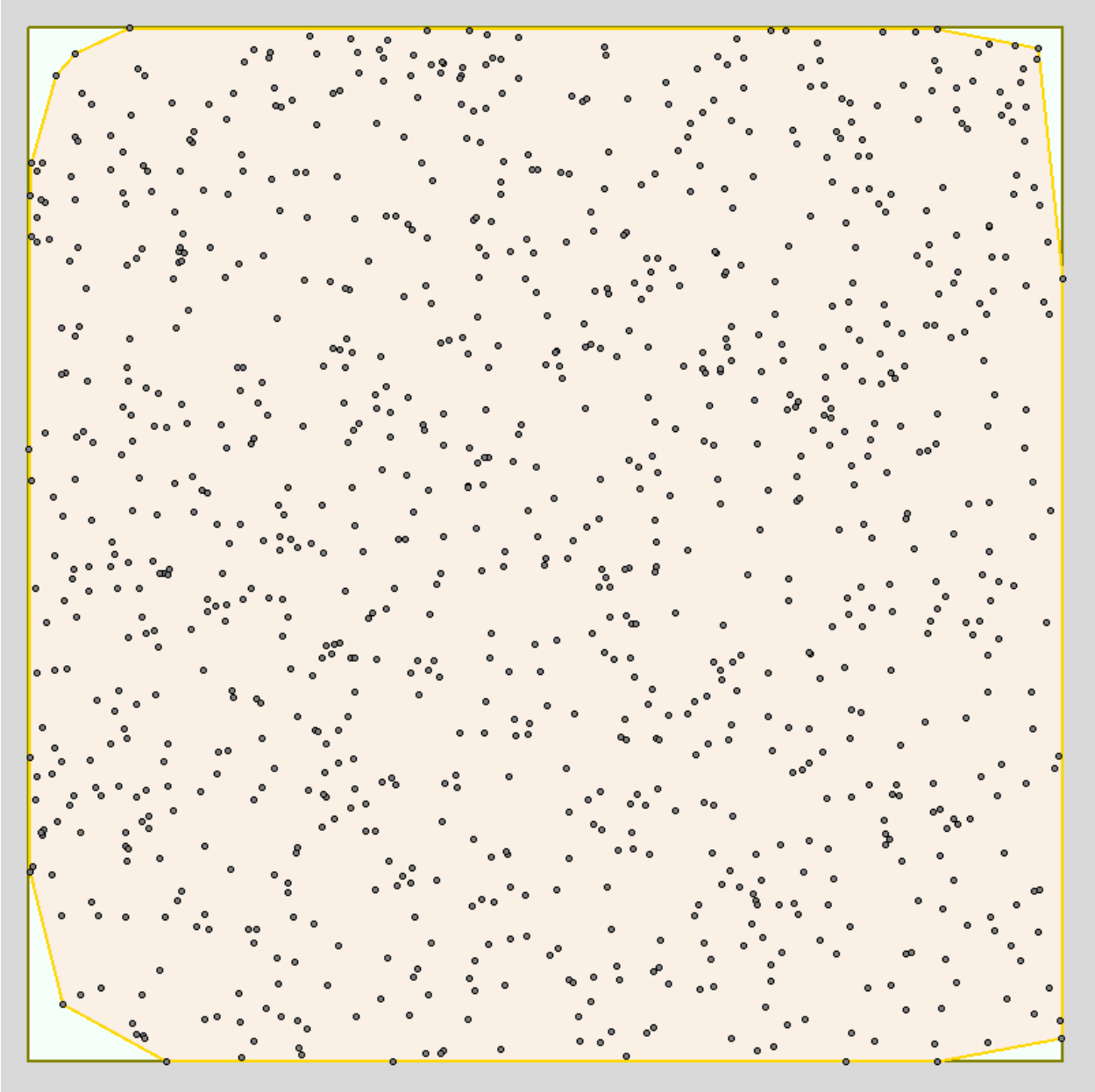
The following is the visualization of the output for a small instance with $n = 20$. The visualization code draws the bounding box in green, the convex hull in linen white, and the input points as black dots.



Here is the timing data printed to standard output:

```
generating n=20 points...
bounding box...
elapsed time = 1.4076940715312958e-05 seconds
convex hull...
elapsed time = 0.002223942894488573 seconds
```

Here is the same output for $n = 1000$:



```
generating n=1000 points...
bounding box...
elapsed time = 0.0005309698171913624 seconds
convex hull...
elapsed time = 326.751867525978 seconds
```

Deliverables

Produce a written project report ***in PDF format***. Your report should include the following:

1. Your name(s)[2], CSUF-supplied email address(es), and an indication that the submission is for project 1.
2. Two scatter plots meeting the requirements stated above.
3. Your pseudocode for both algorithms.
4. Screenshots and standard output from your code for two instances, $n = 20$ and $n = 200$.
5. Answers to the following questions, using complete sentences.
    a. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?
    b. What is the efficiency class of each of your algorithms, according to your own mathematical analysis? (You are not required to include all your math work, just state the classes you derived and proved.)
    c. Are the fit lines on your scatter plots consistent with these efficiency classes? Justify your answer.
    d. Is this evidence *consistent* or *inconsistent* with the hypothesis stated on the first page? Justify your answer.
6. Your complete Python source code.

Your document ***must be uploaded to TITANium as a single PDF file.***

Deadline

The project deadline is Thursday, 3/12, 11:55 pm. Late submissions will not be accepted.

License

[2] Recall that, as stated on the syllabus, you may work in a group of up to three students.