Lecture 1: September 6, 2001

Welcome to 6.838J, Geometric Computation!

- Introductions
- Overview and Goals
- General Information
- Syllabus
- 2D Convex Hull
- Signup sheets (return by end of class)

Course Overview

Geometric Computation is Pervasive

- Robotics, Graphics, CAD/CAM, GIS, Medicine, Net
- Geographic resource discovery
  - Classic examples: nearest post office, hospital
  - GIS overlay: efficient geometric co-occurrence
- Resource simulation and management
  - Art-gallery problem: place minimal set of guards
  - Air-traffic control: when is next collision
- Computer graphics
  - Scalable visibility tests for rendering
  - Global lighting simulations (ray-tracing, radiosity
  - Collision detection for realistic physically-based a
- Computational drug design and discovery
  - Spatial indexing for protein folding
  - Spatial signatures for docking studies
- Robotics
  - Motion planning among obstacles
  - Scalable map construction and localization
- Location-aware computing
  - Location-based services, Mobile dynamic network

Course Goals

Solid introduction to fundamental geometric data struc
Experience with algorithm design and analysis (and op
Intuition about what methods might be applicable as y

General Information

Class has no final, or final project

Components of course grade:

- 50% Lecture/presentation (typically, one per studen
  - Developed jointly with one or both professors
  - Start with list of concepts (two weeks beforehan
  - Review slides, figures, demos (one week beforeh
  - Expect to devote significant time with us, and o
  - Use signup sheet to rank your preference for eac
- 50% Assignments (part mandatory, part optional)
  - Four assignments, roughly one every three week
    - Two weeks to complete each one (see syllab
  - Mandatory component: problems with written
  - Optional component:
    - Either: additional written problems
    - Or: a Java programming assignment
- Open Problems (Optional)
  - Problems stated upon request
  - Solving a significant open problem yields an A+

Syllabus

Low-dimensional computational geometry

- L1: 2D Convex Hulls
- L2: GIS Overlay and Segment Intersection
- L3: Low-Dimensional Linear Programming
- L4: Polygon Triangulation

Organizing Objects and Spaces

- L5: Orthogonal Range Searching
- L6: Point Location / Spatial Indexing
- L7: Voronoi Diagrams
- L8: Robustness and Perturbation Schemes
- L9: Arrangements and Duality
- L10: Delaunay Triangulations, CDTs

Surface Representations and Algorithms

- L11: Representing Polyhedra
- L12: 3D Convex Hulls
- L13: Representing Smooth Surfaces
- L14: Binary Space Partitions

Syllabus (Cont.)

Accounting for Motion

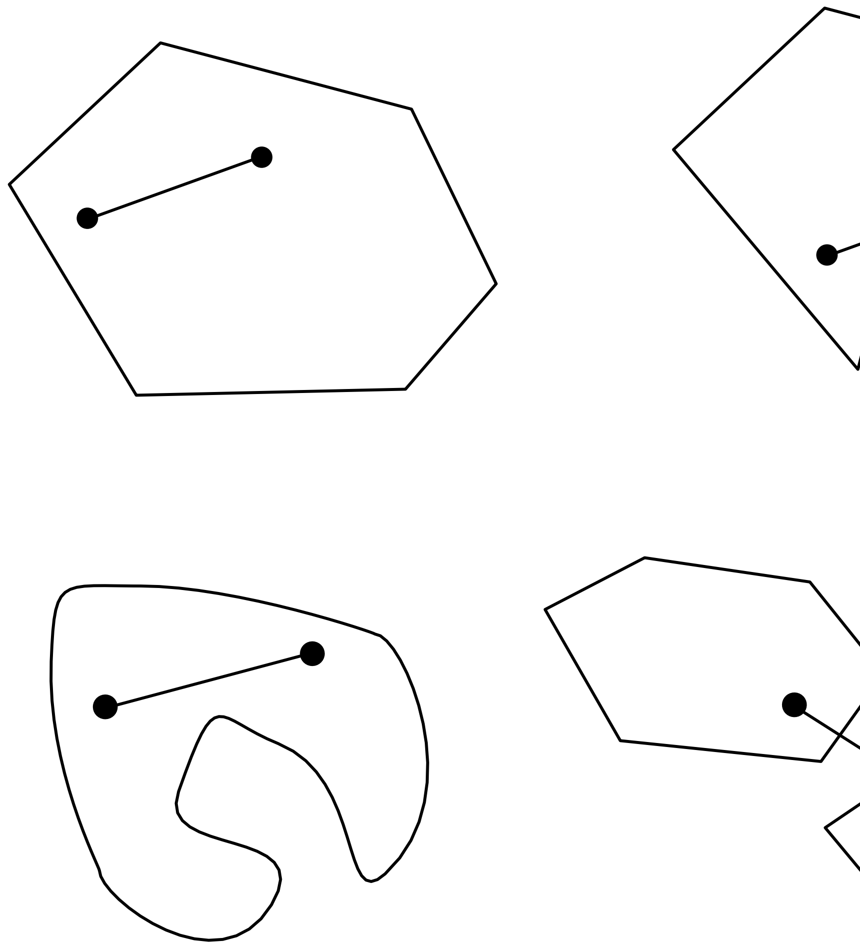
- L15: Kinetic Algorithms
- L16: Robot Motion Planning
- L17: Quadtrees and Non-Uniform Meshing
- L18: Visibility Data Structures

Higher Dimensions

- L19: Medial Axis, Surface Reconstruction
- L20: Higher- and High-Dimensional LP
- L21: Closest-Pair Algorithms
- L22: Approximate Nearest Neighbor
- L23: Iterative Algorithms
- L24: Approximate Nearest Neighbor (Hamming)
- L25: Low-Distortion Embeddings
- L26: Reductions to Approximate Nearest Neighbor

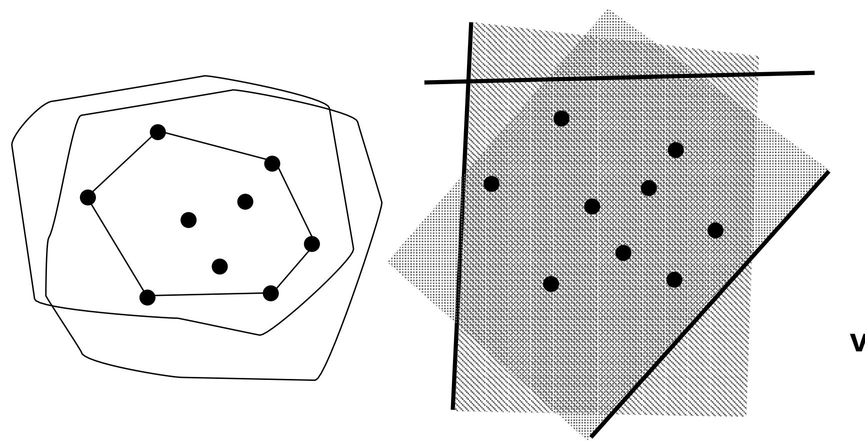
Convexity

A set is convex when every line segment connecting two points in the set is itself contained in the set



Convex Hull

What is the convex hull of a set of points?
Several equivalent definitions:



1. The smallest convex set containing the points
2. The *intersection* of all convex sets containing the poin
3. The *union* of all points expressible as convex combinat

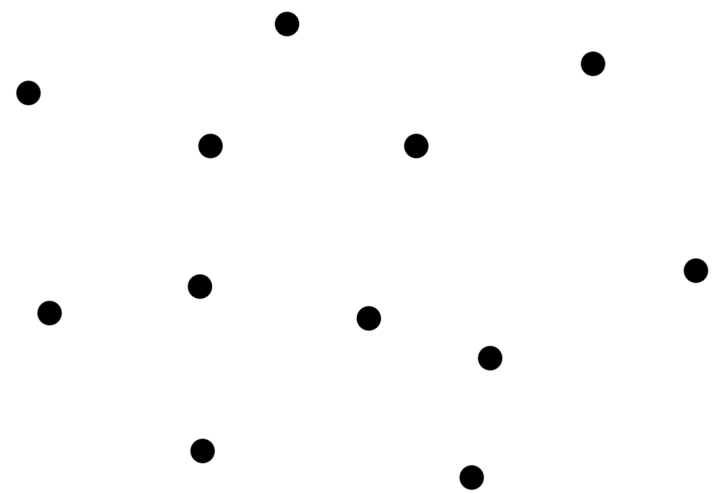
$$\cup\, \mathbf{v} = \sum_{i=1}^{n} c_i \mathbf{p}_i; \;\; \forall\, i, c_i \geq 0; \text{ and } \sum_{i=}^{n}$$

("Convex combination" means coefficients c_i are non-n
Relax non-negativity requirement: get "affine combinat

None of these are particularly well-suited to algorithm

2D Convex Hull

The **2D Convex Hull** problem:



Given a finite set $S \subset \Re^2$ of n points on plane, determine the convex hull of S, denoted $\text{Conv}(S)$.

We'll compute the *boundary* of the convex hull:

In this case, a closed polygonal chain of vertices and

(or simply an ordered list of vertices, with edges implie

For a set S of n points:

What is worst-case complexity of $\text{Conv}(S)$?

... Best-case?

What if all n points are distinct?

2D Convex Hull

Seemingly simple, but illustrates several recurring issue

Algorithm design, analysis, correctness

Progression from brute-force to efficient algorithms

Underlying geometric predicates

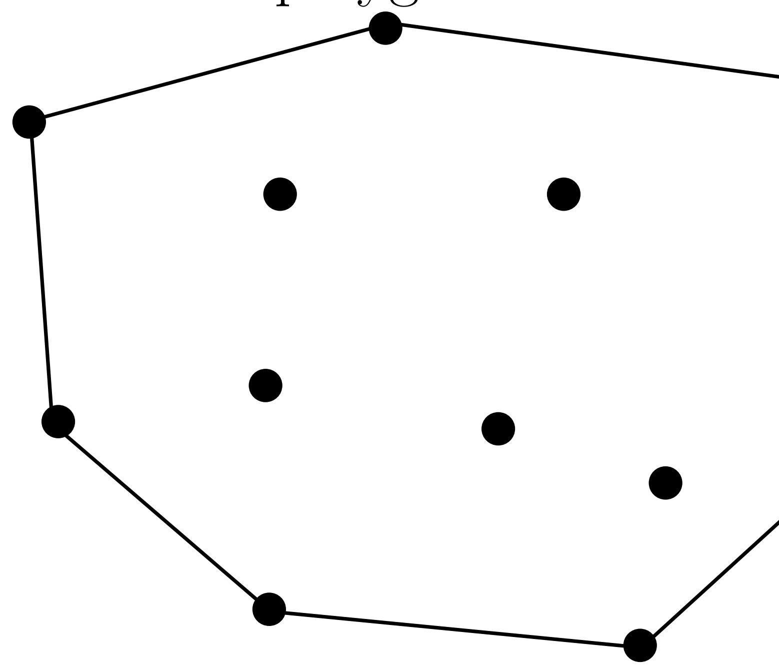
Robustness / Underlying number representations

Genericity assumptions / input degeneracies

Output-sensitive running time

Extremal points

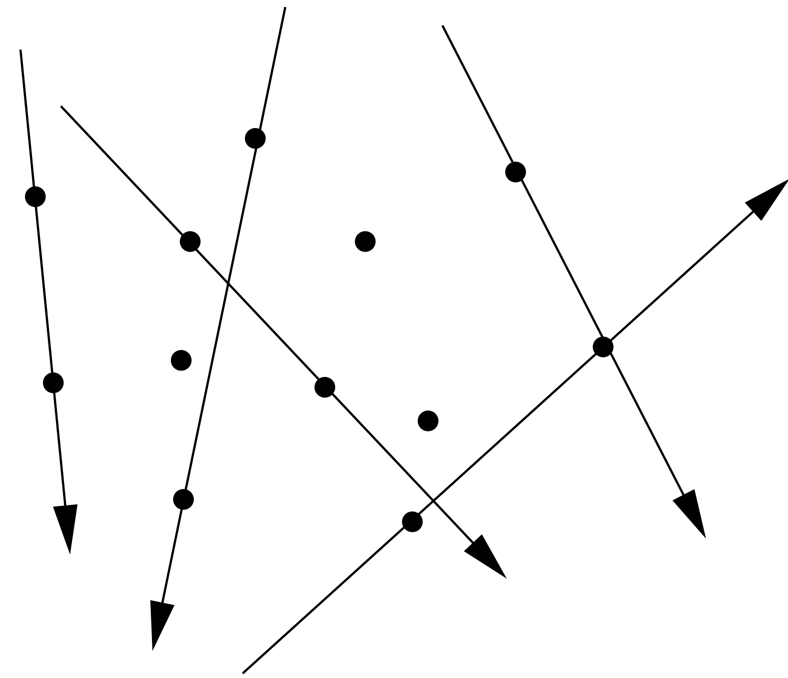
The planar convex hull is a convex polygon.



It can therefore be specified completely by a list of its **e**
These corner points are drawn from the set S.
For now, assume S contains n distinct points

Brute-Force Algorithm

Check each point pair: does it form a boundary edge?

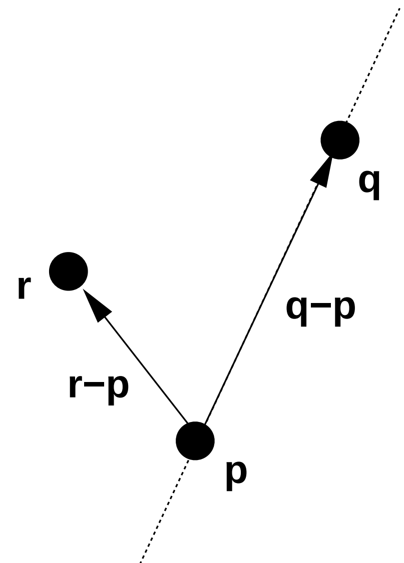


I.e. for all pairs $p, q \in S$, if for *all* $x \in S - \{p, q\}$, x lies
of the oriented line $\overline{pq}$, emit an edge pq on the boundary o

To determine whether point $r = (r_x, r_y)$ lies to the left o
(where $p = (p_x, p_y)$ and $q = (q_x, q_y)$), compute the *sign* o

$$\begin{vmatrix} 1 & r_x & r_y \\ 1 & p_x & p_y \\ 1 & q_x & q_y \end{vmatrix}$$

Leftof Predicate



Why? Just z component of $(\mathbf{q}-\mathbf{p}) \times (\mathbf{r}-\mathbf{p})$:

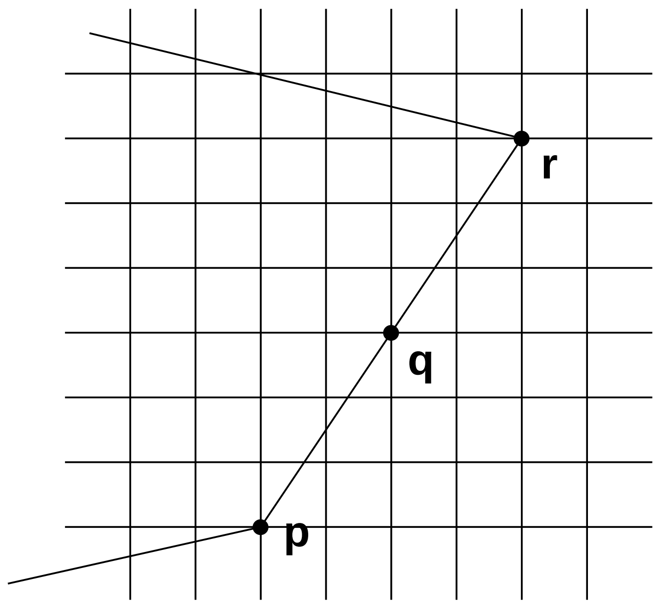
$$\begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ q_x - p_x & q_y - p_y & 0 \\ r_x - p_x & r_y - p_y & 0 \end{vmatrix}$$

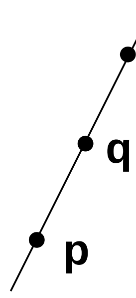
Running time of brute-force algorithm?

Each LeftOf predicate takes $O(1)$ time
There are $\binom{n}{2}$ candidate point pairs
Checking one candidate edge against $n-2$ points ta
Chaining isolated hull edges takes $O(n^2)$ time, naivel
Thus total time is $\binom{n}{2} \cdot n + O(n^2) = O(n^3)$

Degeneracy

As defined, LEFTOF must return either true or false
What if three input points happen to be collinear?
    Suppose LEFTOF returns true. What happens?
    Suppose LEFTOF returns false. What happens?
Problem: sidedness test alone isn't sufficient.



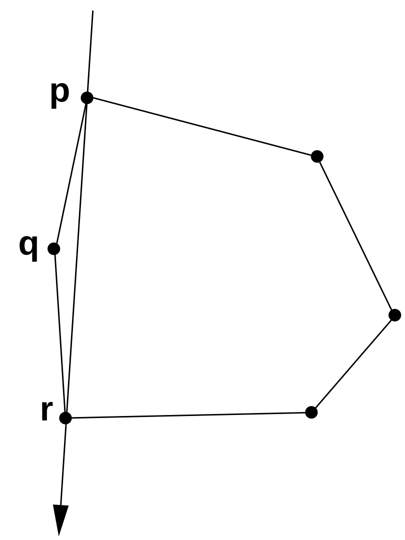


r leftof pq

How can this be fixed?

Robustness

Suppose three input points are *nearly* collinear



Finite-precision (integer, floating-point) arithmetic can

r is LEFTOF **pq**, and

p is LEFTOF **qr**, **and**

q is LEFTOF **pr** !

What happens?

Algorithm is not **robust**.

It can produce non-sensical output.

Robustness Issues

How can this be fixed?

Several options:

- Use arbitrary-precision arithmetic
  - Often overkill
  - Incompatibility with downstream implementatio
- Use precision as necessitated by data
  - Composite quantities
  - Custom predicates
  - Overhead in ordinary case

Robustness is a major, recurring issue in design of geor

Andrews' (1979) modification of Graham

Idea: sort points left to right, then add to hull increme
Particularly simple; handles degeneracies; robust.

```
CONVEXIFY(𝒮, p)
    /* 𝒮 is a stack of points; TOS is t */
    while (𝒮.len ≥ 2) and (p LEFTOF (p_{T-1}, p_T))
        POP(𝒮)
    PUSH(𝒮, p)

SWEEP-HULL( Array p_i )
    Sort p_i in place, by x coordinate
    STACK UpperHull = { p_1 }
    For i = 2 to n
        CONVEXIFY ( UpperHull, p_i )
    STACK LowerHull = { p_n }
    For i = n − 1 downto 1
        CONVEXIFY ( LowerHull, p_i )
    Remove first, last points of LowerHull
    Output UpperHull concat LowerHull
```

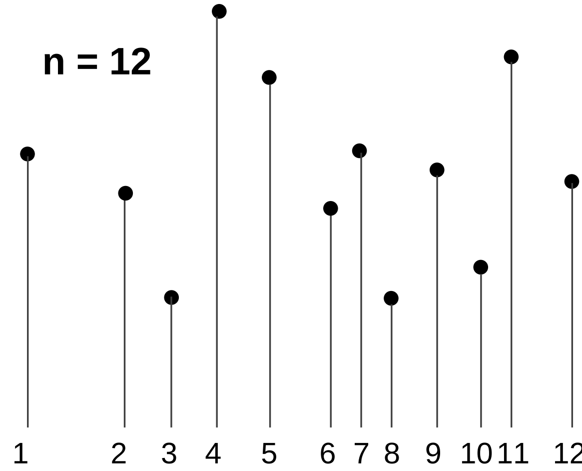
Andrews' Algorithm: Example

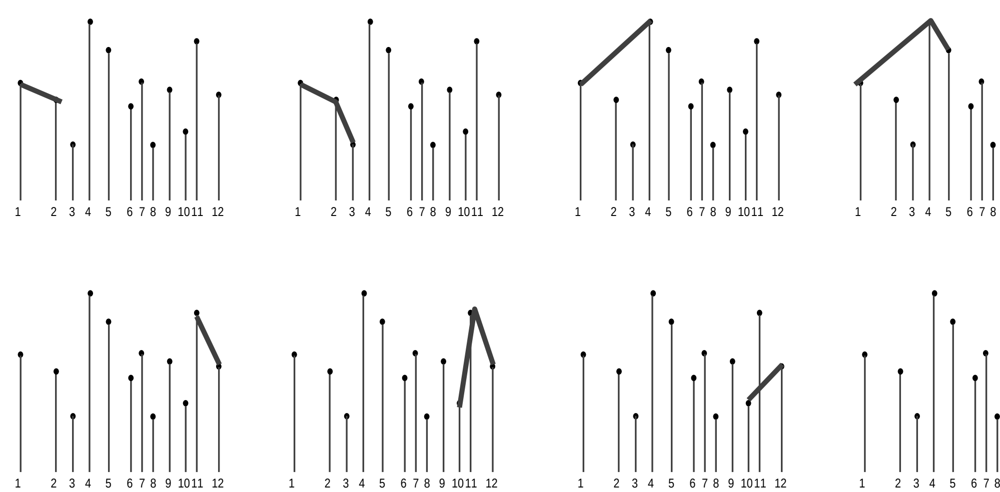
Intuition: Repeatedly

Compare $\mathbf{p}_k$ to directed line from $\mathbf{p}_{T-1}$ to $\mathbf{p}_T$

If $\mathbf{p}_k$ is leftof this line, pop $\mathbf{p}_T$ off of stack

Otherwise, push $\mathbf{p}_k$ onto stack



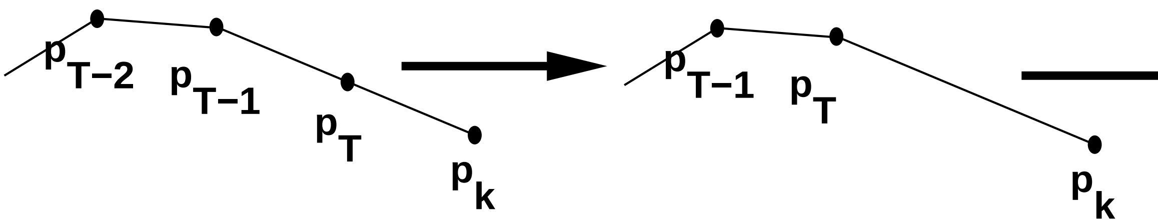


Running time: $O(n \lg n) + 2 \cdot O(n) = O(n \lg n)$

Degeneracy

Are three collinear points a problem?

- Middle point should not occur on output hull
- Make LEFTOF true for this case
- Note how x ordering simplifies things

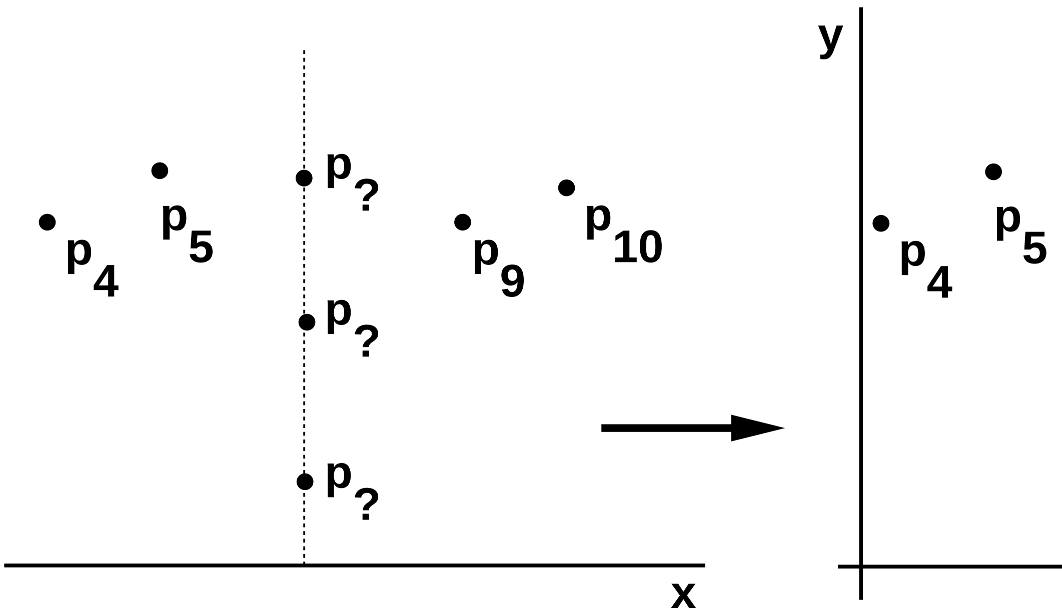


Is this algorithm well-defined?

Degeneracy (cont.)

Algorithm assumed that all x coordinates distinct

What if multiple points with same x coordinate oc



What is solution?

Called "lexicographic sorting"

Robustness, Running Time

Algorithm is guaranteed to produce a closed polygonal

Insufficient precision can still cause erroneous output:

Omission of input point that should occur on hull

Inclusion of input point lying inside hull ("dent" in

So the algorithm is robust, but not necessarily correct.

Running time:

Dominated by initial sort, $O(n \lg n)$ time

Pop and Push can happen at most $O(n)$ times

Example homework questions

Written questions:

Give a linear-time algorithm to compute the conve

Make reasonable assumptions, and state them

How can isolated hull edges be chained together in

Assume each input vertex occurs on either zero

Programming questions:

Implement and animate Andrews' convex hull algo

Free to use public-domain GUI, graphics code

Implement a solution to either of the written quest

OK to work with someone who is writing up th