COS 226 Lecture 15: Geometric algorithms

Important applications involve geometry

* models of physical world
* computer graphics
* mathematical models

Ancient mathematical foundations

Most geometric algorithms less than 20 years old

Knowledge of fundamental algorithms is critical

* use them directly
* use the same design strategies
* know how to compare and evaluate algs

Warning: intuition may mislead

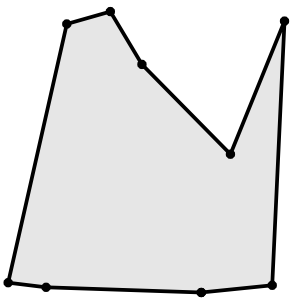
Humans have spatial intuition in 2D and 3D

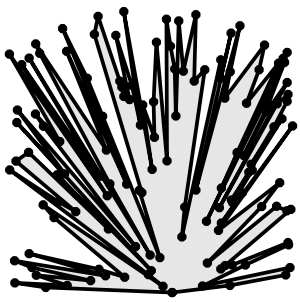
computers do not!

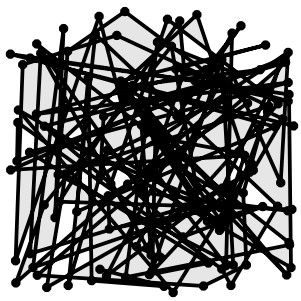
neither have good intuition in high dimensions

Ex: Is a polygon convex?

we think of this alg sees this or even this

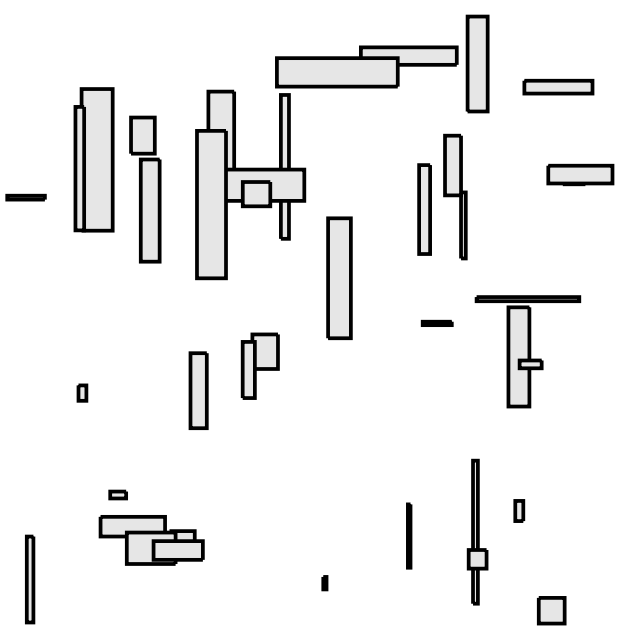


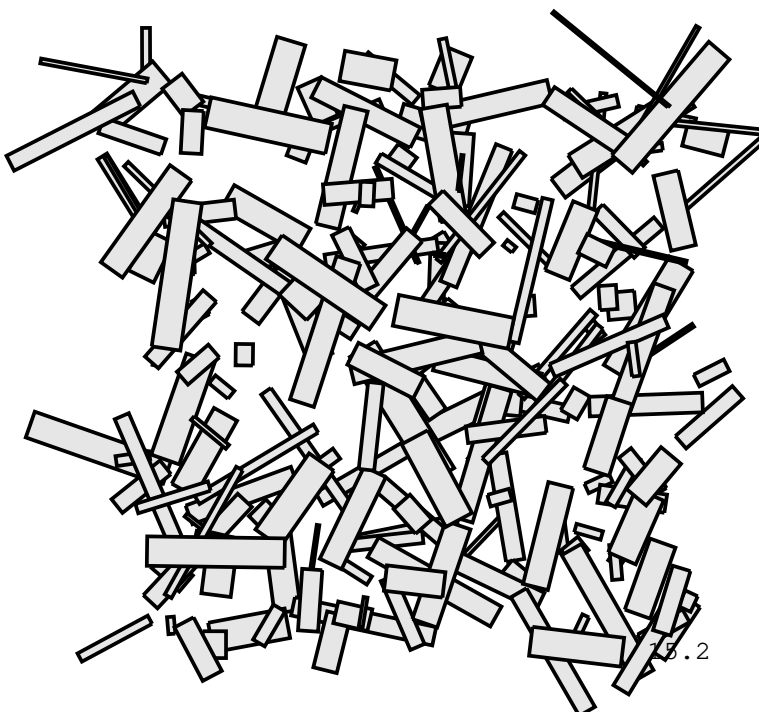




Ex: Find intersections among set of rectangles

we think of this algorithm sees this





Geometric algorithms: overview

New primitives

- points, lines, planes; polygons, circles

Primitive operations

- distance, angles
- "compare" point to line
- do two line segments intersect?

Problems extend to higher dimensions

- (algorithms sometimes do, sometiems don’t)

Higher level intrinsic structures arise

Basic problems

- intersection
- proximity
- point location
- range search

Standard approaches

- incremental (brute-force)
- divide-and-conquer
- sweep line algs
- multidimensional tree structures
- randomized algs
- discretized algorithms
- online and dynamic algs

Algorithm design paradigms

Draw from knowledge about fundamental algs
Move up one level of abstraction
    use fundamental algs and data structures
    know their performance characterisitics

More complicated primitives lead
    to wider range of problems
Some problems too complex
    to admit simple algorithms

For many important problems
    classical approaches give good algorithms
    need research to find "best" algorithms
    no excuse for using "dumb" algorithms

Progression of algorithm design

all possibilities	**double recursion**	**2^N**
brute force	**nested for loops**	**N^2**
divide-and-conquer	**recursion, trees**	**N log N**
elegant idea	**1 for loop**	**N**
randomization	**random choices**	**N**

Oversimplified scenario, but fits many problems

Many examples in geometric algorithms

Geometric primitives

point

two numbers (x, y)

line

two numbers a and b [ax + by = 1]

line segment

four numbers (x1, y1) (x2, y2)

polygon

sequence of points

No shortage of other geometric shapes

triangle, square, circle

Primitive operations can be challenging

is polygon simple?

is point on line?

is point inside polygon?

do two line segments intersect?

do two polygons intersect?

Algorithms search through SETS of primitives

all points in specified range

closest pair in set of points

intersecting pairs in set of line segments

overlapping areas in set of polygons

Line segment intersection

Do two line segments intersect?

To implement INTERSECT(l1, l2)

use simpler primitive SAME(p1, p2, l):

Given two points p1, p2 and a line l,
are p1 and p2 on the same side of l?

yes no no no

To implement SAME

use simpler primitive CCW(p1, p2, p3):

Given three points p1, p2, p3,
is the route p1–p2–p3 a ccw turn?

yes no

two ccw tests to implement SAME
four ccw tests to implement INTERSECT

CCW implementation

Compare dx, dy

```
int ccw(struct point p0, struct point p1, struct p
   {
      int dx1, dx2, dy1, dy2;
      dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
      dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
      if (dx1*dy2 > dy1*dx2) return 1;
      if (dx1*dy2 < dy1*dx2) return -1;
      return 0;
   }
```

Still not quite right! Bug in degenerate case

points on the same line

Does AB intersect CD?

on the line in the order ABCD: NO

on the line in the order ACDB: YES

Can’t just return 0 if dx1*dy2 = dx2*dy1 (see book

CCW is a useful basic primitive

Ex: is point inside convex N-gon? N CCW tests

Lesson:

geometric primitives are tricky to implement

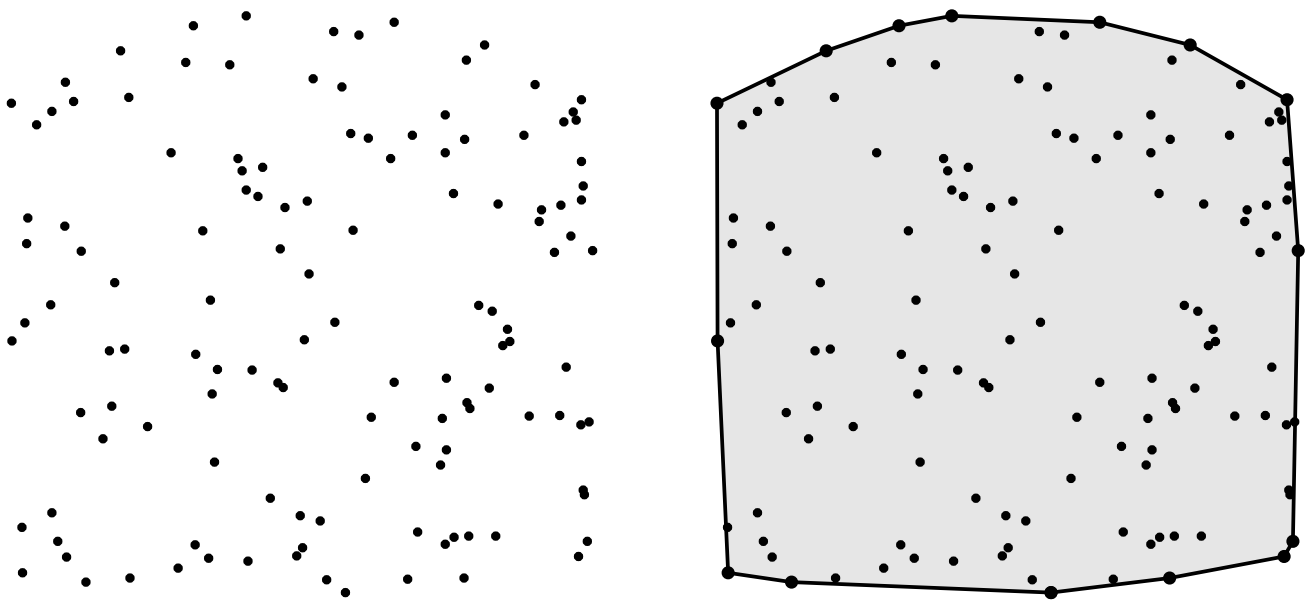
can’t ignore degenerate cases

Convex hull of a point set

Basic property of a set of points

CONVEX HULL:

* smallest convex polygon enclosing the points
* shortest path surrounding the points
* intersection of halfplanes defined by point pair



Running time of algorithm can depend on

N: number of points

M: number of points on the hull

point distribution

Package-wrap algorithm

Operates like selection sort

Abstract idea

- sweep line anchored at current point CCW
- first point hit is on hull

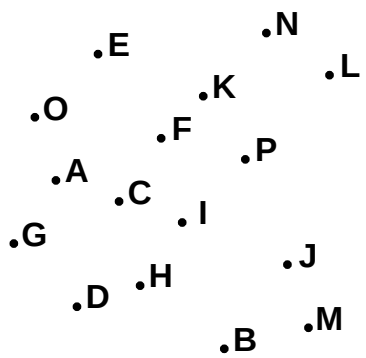
Implementation

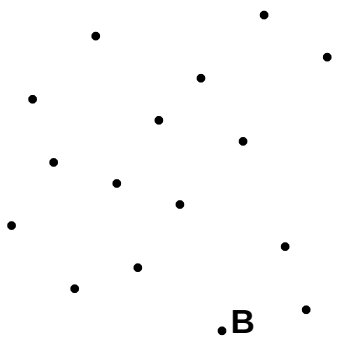
- compute angle to all points
- pick smallest angle larger than current one

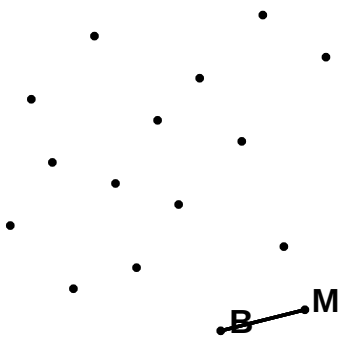
```
int wrap(struct point p[], int N)
  { int i, min, M; float th, v; struct point t;
    for (min = 0, i = 1; i < N; i++)
      if (p[i].y < p[min].y) min = i;
    p[N] = p[min]; th = 0.0;
    for (M = 0; M < N; M++)
      {
        t = p[M]; p[M] = p[min]; p[min] = t;
        min = N; v = th; th = 360.0;
        for (i = M+1; i <= N; i++)
          if (theta(p[M], p[i]) > v)
            if (theta(p[M], p[i]) < th)
            { min = i; th = theta(p[M], p[min])
        if (min == N) return M;
      }
  }
```

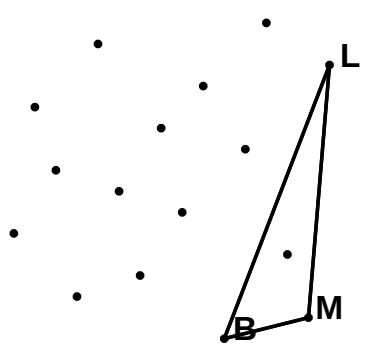
Use pseudo-angle to save time

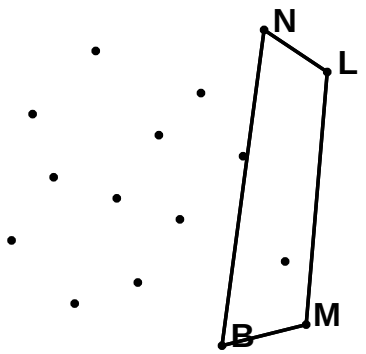
Package-wrap example

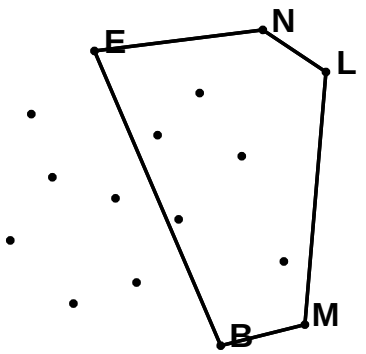


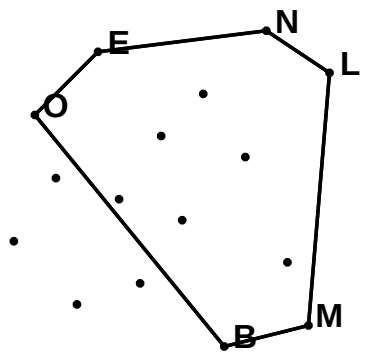


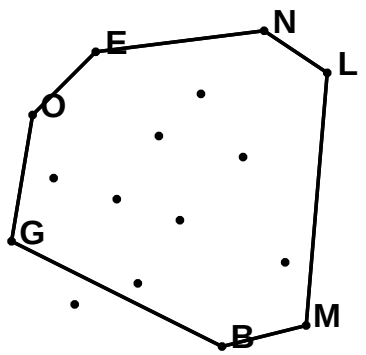


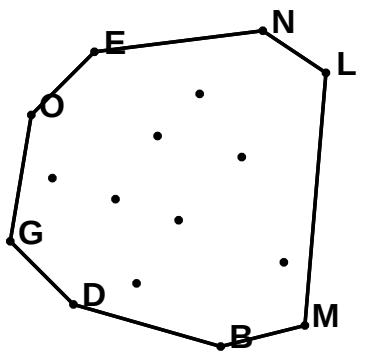












Graham Scan

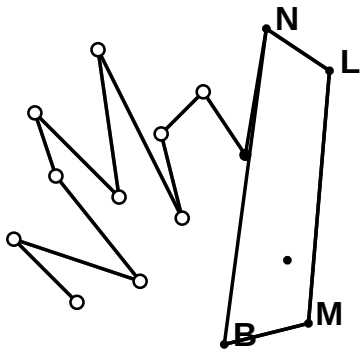
Sort points on angle with bottom point as origin
    forms simple closed polygon

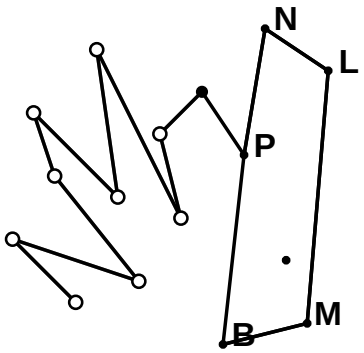
Proceed through polygon
    discard points that would cause a CW turn

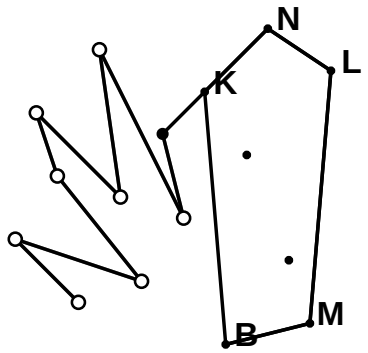
```
int grahamscan(struct point p[], int N)
  { int i, min, M; struct point t;
    for (min = 1, i = 2; i <= N; i++)
      if (p[i].y < p[min].y) min = i;
    for (i = 1; i <= N; i++)
      if (p[i].y == p[min].y)
        if (p[i].x > p[min].x) min = i;
    t = p[1]; p[1] = p[min]; p[min] = t;
    shellsort(p, N);
    p[0] = p[N];
    for (M = 3, i = 4; i <= N; i++)
      {
        while (ccw(p[M],p[M-1],p[i]) >= 0) M--;
        M++; t = p[M]; p[M] = p[i]; p[i] = t;
      }
    return M;
  }
```

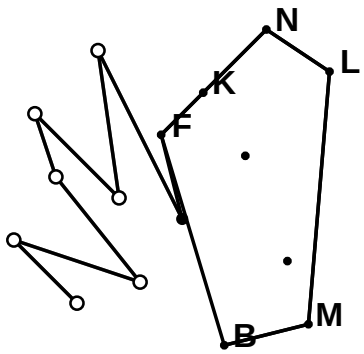
Running time: N log N for sort; N for scan

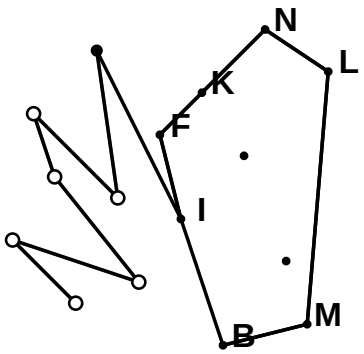
Graham scan example

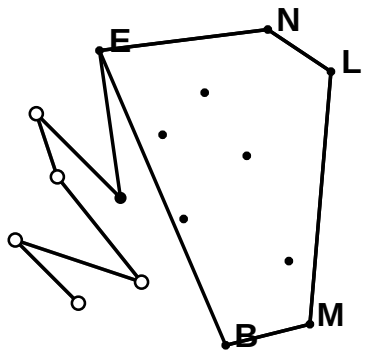


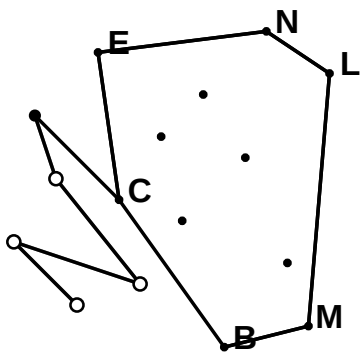


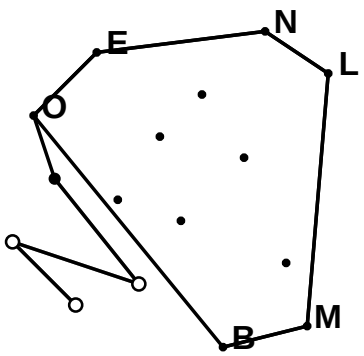


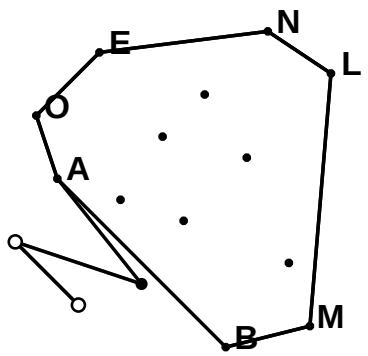


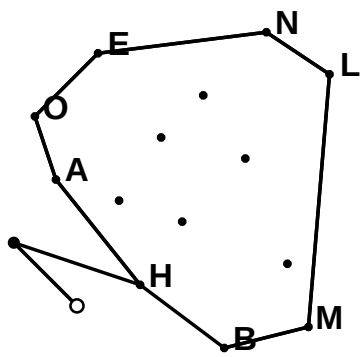


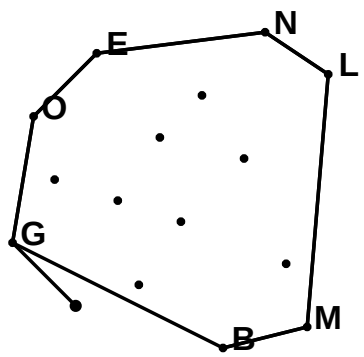


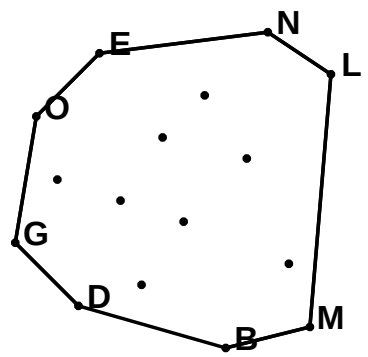






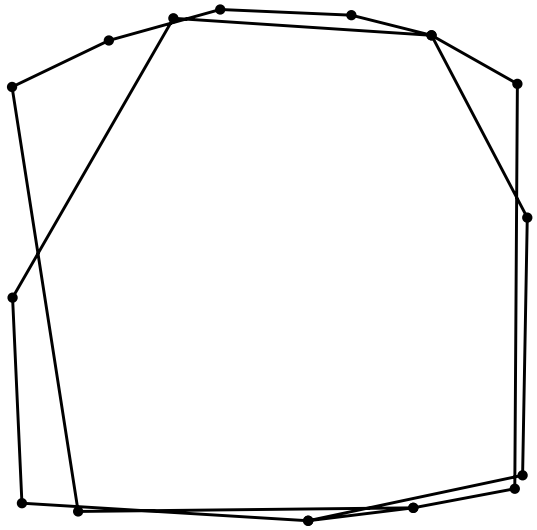




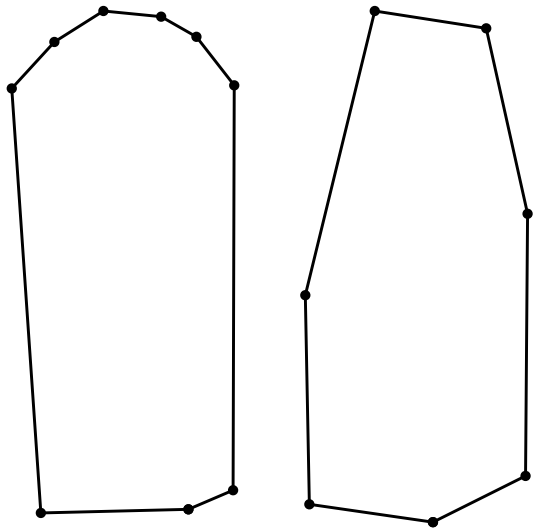


Divide-and-conquer convex hull algorithms

divide points



divide space



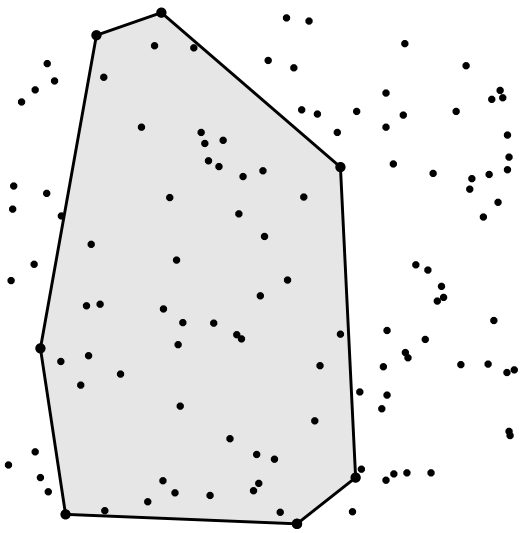
switch dimensions to avoid strange degeneracies

Relatively complex to code in both cases

Incremental convex hull algorithm

Consider next point

- if inside hull of previous points, ignore
- if outside, update hull



Two subproblems to solve

- test if point inside or outside polygon
- update hull for outside points

Both subproblems

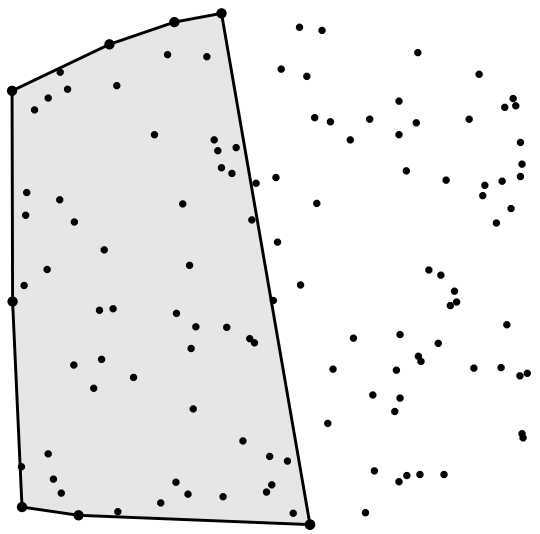
- can be solved by looking at all hull points
- can be improved with binary search

Randomized algorithm

- consider points in random order
- N + M log M

"Sweep line" convex hull algorithm

Sort points on x-coordinate first



Eliminates inside test

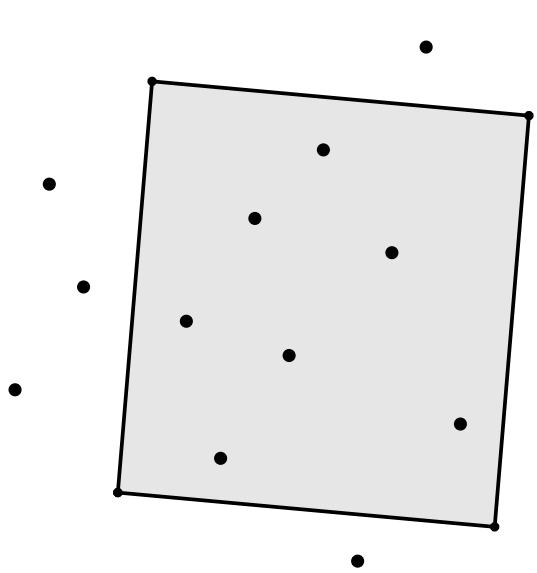
Total time proportional to N log N (for sort)

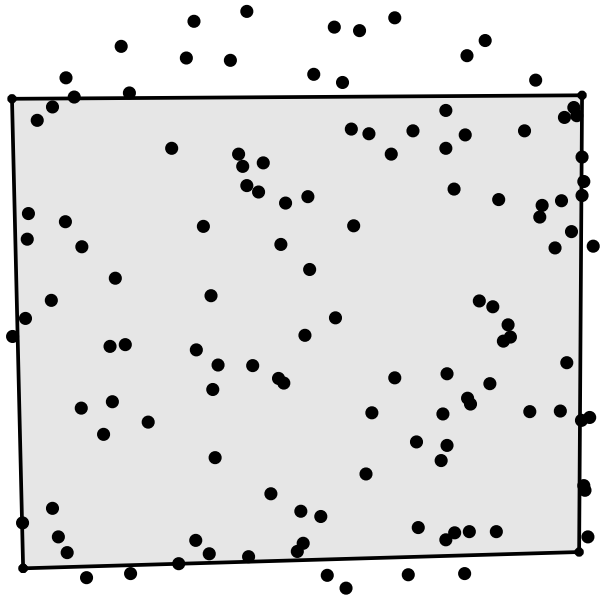
Quick elimination

Improve the performance of any convex hull algorithm by quickly eliminating most points (known not to be on the hull)

Use points at "corners"

max, min x+y, x−y





Check if point inside quadrilateral

four CCW tests

Check if point inside rectangle

four comparisons

Almost all points eliminated if points random

number of points left proportional to N^(1/2)

Total time proportional to N

Summary of 2D convex hull algs

Package wrap

NM

Graham scan

N log N (sort time)

Divide-and-conquer

N log N (with work)

Quick elimination

N (fast average-case)

Brute-force

N log M

Sweep line

N log N (sort time)

How many points on the hull?

Worst case: N

Average case: depends on distribution

* uniform in a convex polygon: log N

* uniform in a circle: N^(1/3)

Average-case analysis requires understanding

of basic properties of DATA, not just algs

Higher dimensions

Multifaceted convex polytope encloses points

NOT a simple object

Ex: N points d dimensions

- d=2: convex hull
- d=3: Euler's formula (v − e + f = 2)
- d>3: exponential number of facets at worst

EXTREME POINTS

- return points on the hull, not necc in order

Package-wrap

Divide-and-conquer

Randomized

Interior elimination

Geometric models of mathematical problems

Impact of geometric algorithms extends far beyond models of physical objects

Two important examples:

Geometric problem
- find point where two lines intersect in 2D
- find point where three planes intersect in 3D
- ...

Mathematical equivalent
- solve simultaneous equations
- algorithm: gaussian elimination

Geometric problem
- find convex polytope defined by intersecting half-planes
- find vertex hit by line of given slope moving in from infinity

Mathmatical equivalent
- linear programming
- algorithm: simplex

(stay tuned)

Linear programming example

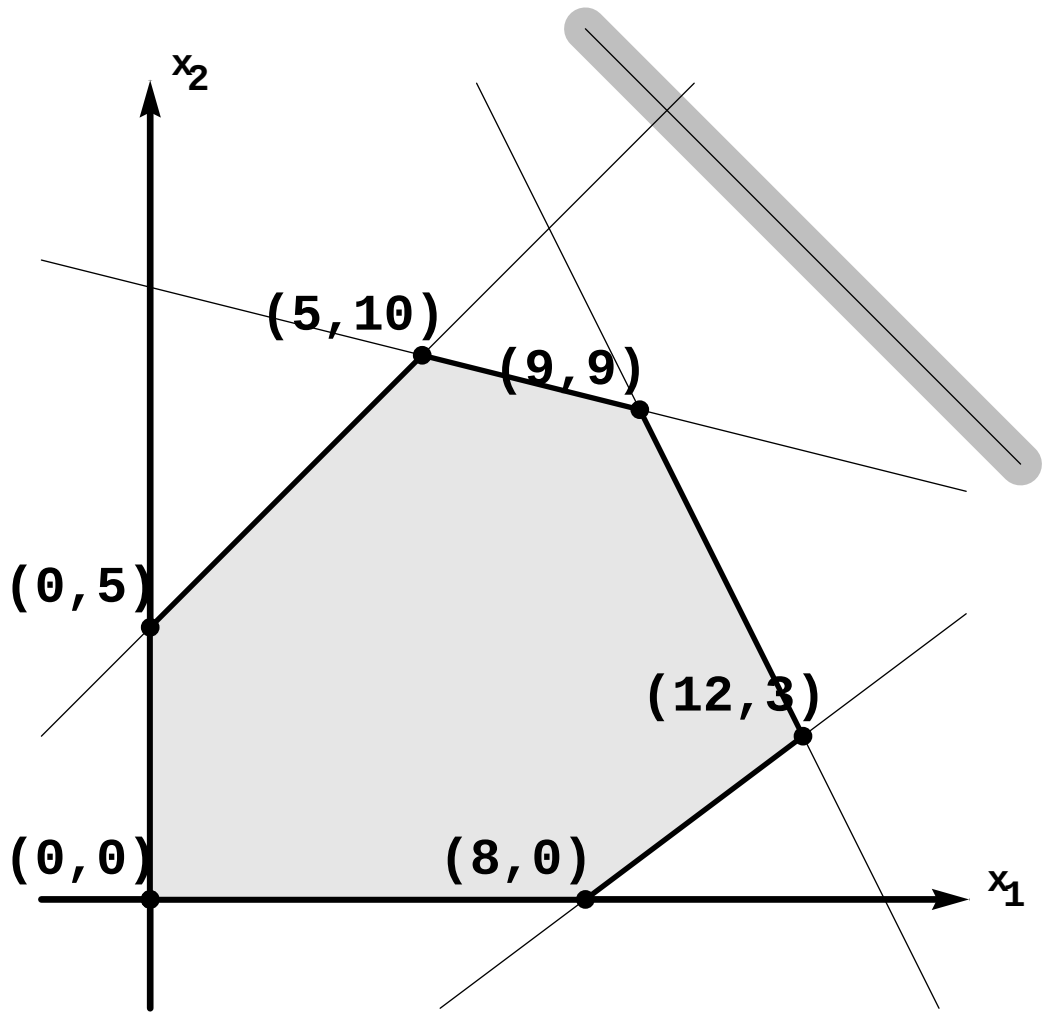
Maximize a+b subject to the constraints

$b - a < 5$

$a + 4b < 45$

$2a + b < 27$

$3a - 4b < 24$

$a > 0$

$b > 0$