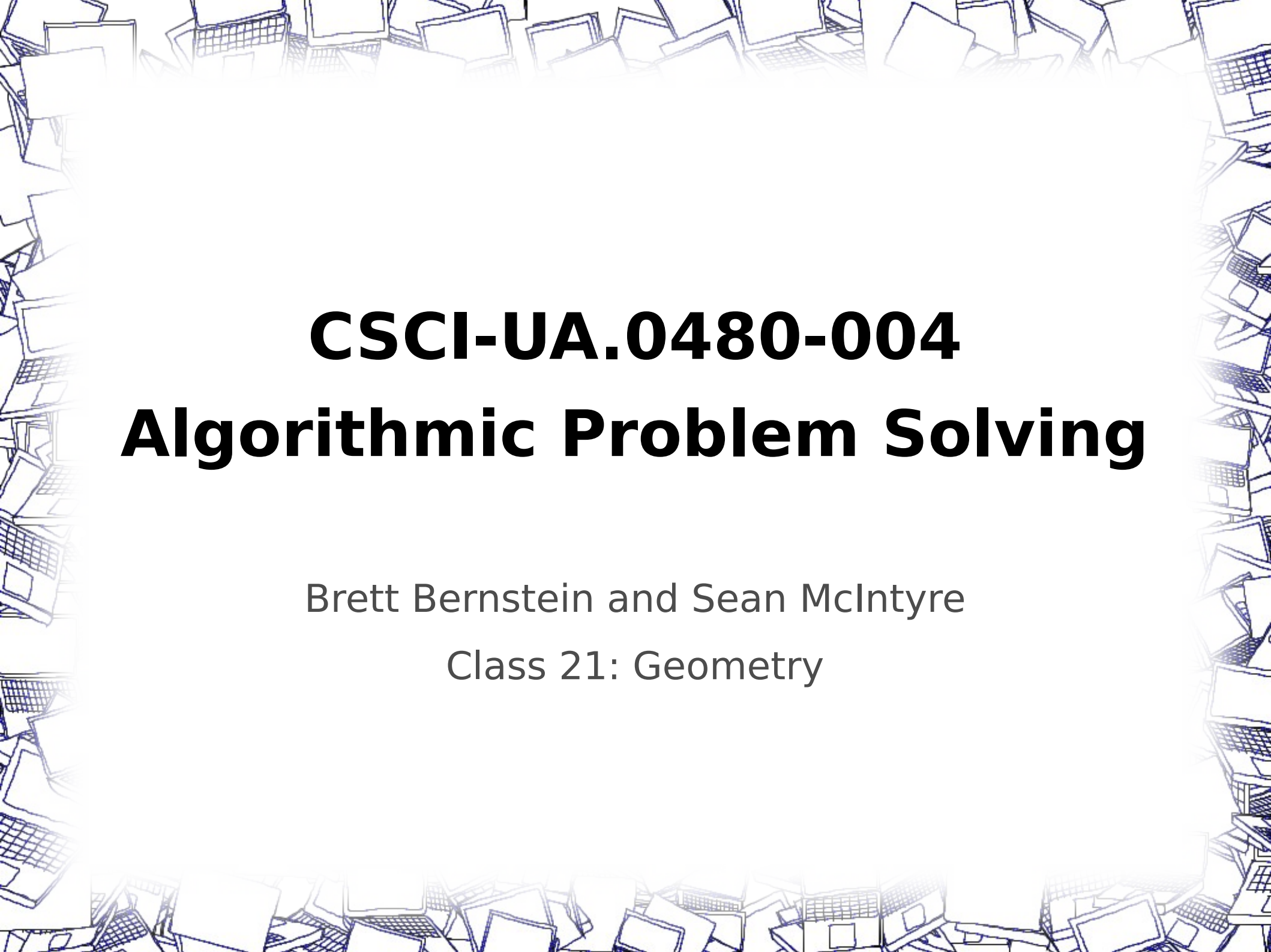


CSCI-UA.0480-004
Algorithmic Problem Solving

Brett Bernstein and Sean McIntyre

Class 21: Geometry

Geometry

```
class point implements Comparable<point> {
    double x, y; // only used if more precision is needed

    point() {
        x = y = 0.0;
    } // default constructor

    point(double _x, double _y) {
        x = _x;
        y = _y;
    } // user-defined

    // use EPS (1e-9) when testing equality of two floating points
    public int compareTo(point other) { // override less than operator
        if (Math.abs(x - other.x) > EPS) // useful for sorting
            return (int) Math.ceil(x - other.x); // first: by x-coordinate
        else if (Math.abs(y - other.y) > EPS)
            return (int) Math.ceil(y - other.y); // second: by y-coordinate
        else
            Return 0; // they are equal
    }
};
```

Geometry

- Euclidean distance

```
double dist(point p1, point p2) {                    // Euclidean distance
  return Math.hypot(p1.x - p2.x, p1.y - p2.y); }        // return double
```

Geometry

- Lines
  - Equation for a line, y = mx + b
  - Better representation with ax + by + c = 0

```
class line { double a, b, c; };                  // a way to represent a line

// the answer is stored in the third parameter
void pointsToLine(point p1, point p2, line l) {
  if (Math.abs(p1.x - p2.x) < EPS) {                  // vertical line is fine
    l.a = 1.0;   l.b = 0.0;   l.c = -p1.x;
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0;                      // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }
```

Geometry

- Lines
  - Do they intersect?
    - Parallel = no
    - Same line = yes, infinite number of times

```
boolean areParallel(line l1, line l2) {       // check coefficients a & b
  return (Math.abs(l1.a-l2.a) < EPS) && (Math.abs(l1.b-l2.b) < EPS); }

boolean areSame(line l1, line l2) {           // also check coefficient c
  return areParallel(l1 ,l2) && (Math.abs(l1.c - l2.c) < EPS); }
```

Geometry

- Lines
  - Do they intersect?
    - Otherwise, once, when
      - $a_1x + b_1y + c_1 = a_2x + b_2y + c_2$

```
boolean areIntersect(line l1, line l2, point p) {
  if (areParallel(l1, l2)) return false;                  // no intersection
  // solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
  // special case: test for vertical line to avoid division by zero
  if (Math.abs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else                      p.y = -(l2.a * p.x + l2.c);
  return true; }
```

Geometry

- Line Segments
  - Lines with two endpoints (finite length)
- Vectors
  - Line segment with a direction, starting from (0, 0)

```
class vec { double x, y;      // name: `vec' is different from Java Vector
  vec(double _x, double _y) { x = _x; y = _y; } };

vec toVec(point a, point b) {                  // convert 2 points to vector
  return new vec(b.x - a.x, b.y - a.y); }

vec scale(vec v, double s) {             // nonnegative s = [<1 .. 1 .. >1]
  return new vec(v.x * s, v.y * s); }                  // shorter.same.longer

point translate(point p, vec v) {              // translate p according to v
  return new point(p.x + v.x , p.y + v.y); }
```

Geometry

- Motivation, Codeforces 227A, Where do I Turn?
  - A knight travels from point A to point B. He wants to travel to point C but does not know the direction.
  - An eagle helps the knight by flying up and spotting point C.
  - The eagle responds with “TOWARDS” (straight ahead), “RIGHT”, or “LEFT”

Geometry

- How to solve
  - Find the cross product of AB, BC
  - The magnitude of this vector is the area of the parallelogram that these vectors span
    - If it's zero, the points A, B, and C are collinear
    - If nonzero, then the sign of the magnitude indicates what side C is on of line AB

```
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
```

Geometry

- Circles
  - Defined by a point P and a radius r
  - Diameter is twice the radius
  - Circumference = length of the circle's edge
    - C = PI * d
  - Area is A = PI * r^2

Geometry

- Example problem, Trace
  - Given a list of circles all with difference radii but the same point
  - The circles are painted on a wall, alternating in color, red and blue
  - The circles are sorted from large to small (largest painted first)
  - What is the area of the red on the wall?
    - The wall itself starts blue

Geometry

- Triangles
  - A polygon with three points and three edges
  - Perimeter of triangle is p = a + b + c
  - Area of right angle triangle:
    - A = b * h / 2
  - Area of general triangle:
    - A = sqrt(s * (s – a) * (s – b) * (s – c))
    - s = 0.5 * p, the semi-perimeter
    - Called Heron's Formula

Geometry

- Covered in book
  - Quadrilaterals
    - Rectangles, squares, parallelograms, etc.
  - Basic trigonometry
    - Pythagorean theorem
    - Law of sines
    - Law of cosines

Geometry

- Polygons
  - Perimeter of polygons
    - Cycle through all edges and add their lengths

```
double perimeter(point[] pts) {
    double p = 0.0;
    for (int i = 0; i < pts.length; i++) {
        p += dist(pts[i], pts[(i+1) % pts.length]);
    }
    return p;
}
```

Geometry

- Polygons
  - Area of polygons, given in cw or ccw order

$$A = \frac{1}{2}\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ . & . \\ . & . \\ . & . \\ x_n & y_n \end{vmatrix} = \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1 \bmod n} - x_{i+1 \bmod n} y_i)$$

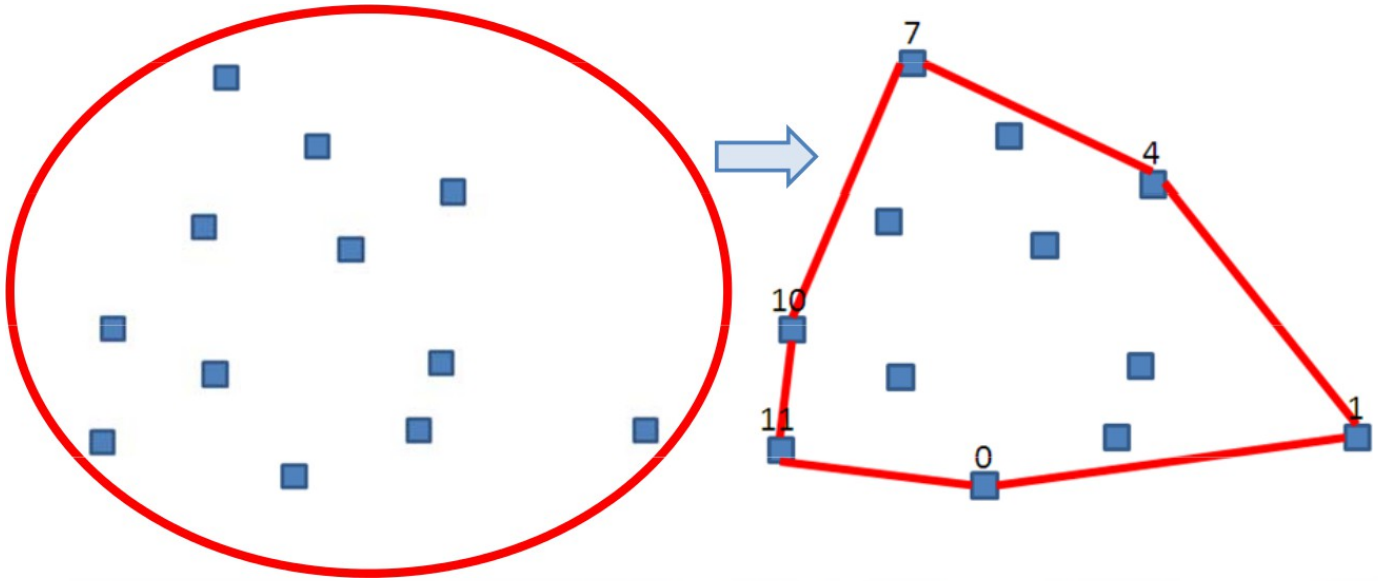
Geometry

- Polygons
  - Area of polygons, given in cw or ccw order
    - Works with convex and concave polygons

```
public static double signedArea(point[] p, int n) {
    double sum = 0;
    for (int i = n - 1, j = 0; j < n; i = j++) {
        sum += p[i].x * p[j].y - p[i].y * p[j].x;
    }
    return 0.5 * sum;
}
```

Geometry

- Convex Hull
  - The convex hull of a set of points P is the smallest convex polygon CH(P) for which each point in P is either on the boundary of CH(P) or in its interior



Geometry

- Convex Hull
  - Algorithm sketch:
    - Find bottom most, right most point (“the pivot”)
    - Angular sort all points with respect to the pivot
    - Traverse points, add points that are CCW