The University of Danang
University of Science and Technology

MATHEMATICS FOR COMPUTER SCIENCE

Chap 3. Geometry

Faculty of Information Technology
PHAM Cong Thang, PhD

References

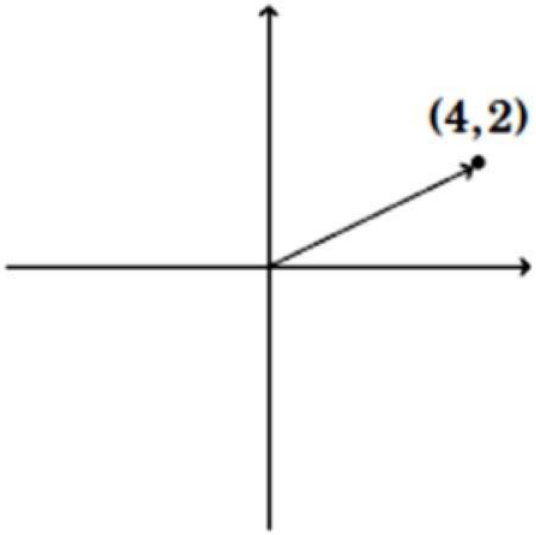
1. M. P. Deisenroth, A. A. Faisal, and C. S. Ong, Mathematics for Machine Learning, Cambridge University Press; 1 edition (April 23, 2020), 398 pages.
2. E. Lehman, F. T. Leighton, A. R. Meyer, 2017, Mathematics for Computer Science, Eric Lehman Google Inc, 998 pages
3. A. Laaksonen, Competitive Programmer's Handbook, 2018, 286 pages.
4. W. H. Press, S. A. Teukolsky, W.T. Vetterling, B. P. Flannery Numerical Recipes: The Art of Scientific Computing, Third Edition, Cambridge University Press, 1262 pages.
5. Other online/offline learning resources

Geometry

- **Complex numbers**
- Points and lines
- Polygon area
- Distance functions
- Intersection points
- Closest pair problem
- Convex hull problem

Geometry - Complex number

- A **complex number** is a number of the form $x + yi$, where $i = \sqrt{-1}$ is the **imaginary unit**.
  - A geometric interpretation of a complex number is that it represents a two-dimensional point (x, y) or a vector from the origin to a point (x, y)
  - For example, $4 + 2i$ corresponds to the following point and vector:



Geometry - Complex number

- The C++ complex number class complex is useful when solving geometric problems.
  - Using the class we can represent points and vectors as complex numbers, and the class contains tools that are useful in geometry.
  - C is the type of a coordinate and P is the type of a point or a vector. In addition, the code defines macros X and Y that can be used to refer to x and y coordinates.

```
typedef long long C;
typedef complex<C> P;
#define X real()
#define Y imag()
```

Geometry - Complex number

- For example, the following code defines a point $p = (4,2)$ and prints its x and y coordinates:

```
P p = {4,2};
cout << p.X << " " << p.Y << "\n"; // 4 2
```

- The following code defines vectors $v = (3,1)$ and $u = (2,2)$, and after that calculates the sum $s = v + u$

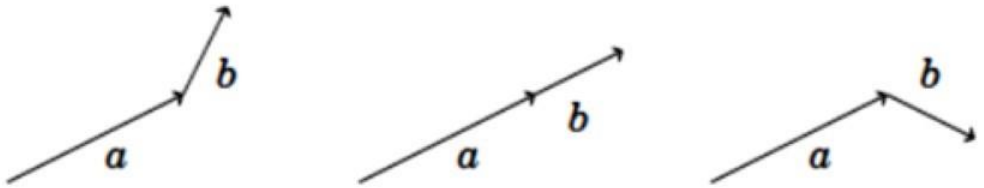
```
P v = {3,1};
P u = {2,2};
P s = v+u;
cout << s.X << " " << s.Y << "\n"; // 5 3
```

Geometry

- Complex numbers
- **Points and lines**
- Polygon area
- Distance functions
- Intersection points
- Closest pair problem
- Convex hull problem

Geometry - Points and lines

- The **cross product** $a \times b$ of vectors $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is calculated using the formula $x_1\, y_2 - x_2\, y_1$.
  - The cross product tells us whether b turns left (positive value), does not turn (zero) or turns right (negative value) when it is placed directly after a
    - The following picture illustrates the above cases:



$a \times b = 6$ $\quad$ $a \times b = 0$ $\quad$ $a \times b = -8$

Geometry - Points and lines

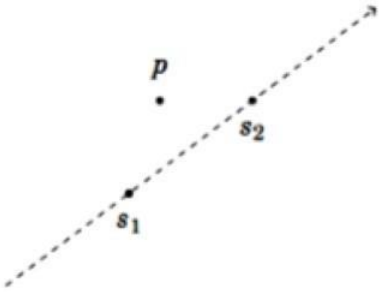
- For example, in the first case $a = (4,2)$ and $b = (1,2)$.
  - The following code calculates the cross product using the class complex:

```
P a = {4,2};
P b = {1,2};
C p = (conj(a)*b).Y; // 6
```

  - The above code works, because the function ***conj*** negates the y coordinate of a vector, and when the vectors (x_1, y_1) and (x_2, y_2) are multiplied together, the y coordinate of the result is $x_1\ y_2 - x_2\ y1$

Geometry - Points and lines

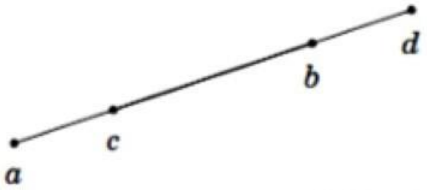
- Point location
  - Cross products can be used to test whether a point is located on the left or right side of a line. Assume that the line goes through points s1 and s2, we are looking from s_1 to s_2 and the point is p



  - The cross product $(p - s_1) \times (p - s_2)$ tells us the location of the point p. If the cross product is positive, p is located on the left side, and if the cross product is negative, p is located on the right side. Finally, if the cross product is zero, points s_1, s_2 and p are on the same line.

Geometry - Points and lines

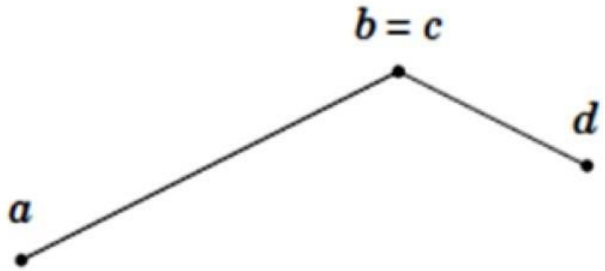
- Line segment intersection
  - We consider the problem of testing whether two line segments ab and cd intersect.
  - Case 1: The line segments are on the same line and they overlap each other.
    - In this case, there is an infinite number of intersection points. For example, all points between c and b are intersection points:



    - In this case, we can use cross products to check if all points are on the same line. After this, we can sort the points and check whether the line segments overlap each other.

Geometry - Points and lines

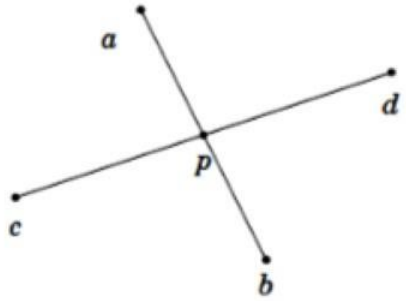
- Line segment intersection
  - Case 2: The line segments have a common vertex that is the only intersection point. For example, the intersection point is $b = c$



  - This case is easy to check, because there are only four possibilities for the intersection point: $a = c, a = d, b = c$ and $b = d$

Geometry - Points and lines

- Line segment intersection
  - Case 3: There is exactly one intersection point that is not a vertex of any line segment, the point p is the intersection point:



  - In this case, the line segments intersect exactly when both points c and d are on different sides of a line through a and b, and points a and b are on different sides of a line through c and d. We can use cross products to check this.

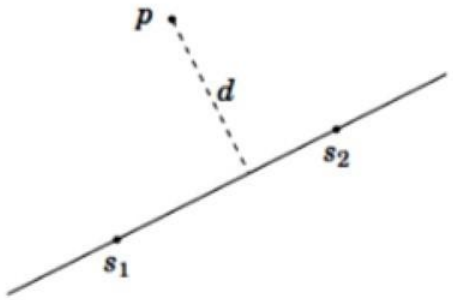
Geometry - Points and lines

- Point distance from a line
  - Another feature of cross products is that the area of a triangle can be calculated using the formula

$$\frac{|(a-c)\times(b-c)}{2}$$

where a, b and c are the vertices of the triangle

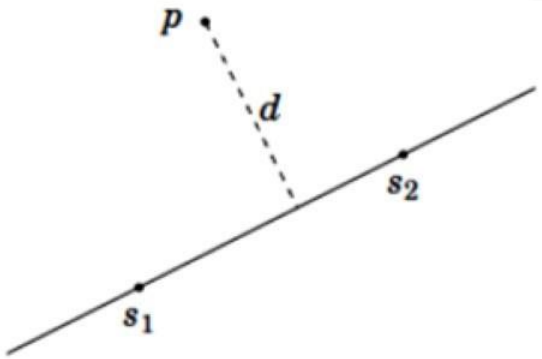
  - Using this fact, we can derive a formula for calculating the shortest distance between a point and a line. For example, d is the shortest distance between the point p and the line that is defined by the points s_1 and s_2



Geometry - Points and lines

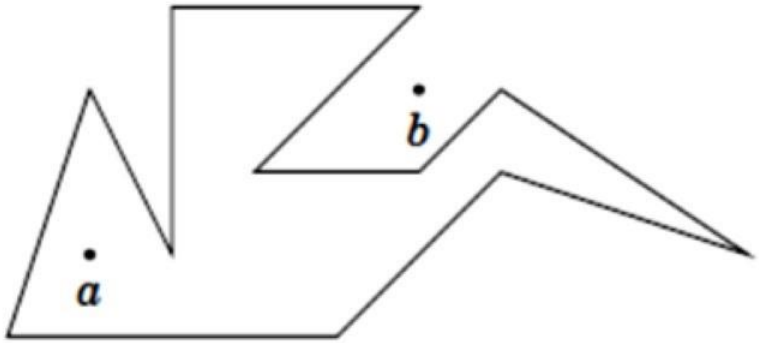
- Point distance from a line
  - The area of the triangle whose vertices are s_1, s_2 and p can be calculated in two ways: it is both $\frac{1}{2}|s_2 - s_1|d$ and $\frac{1}{2}((s_1 - p) \times (s_2 - p))$.
    - Thus, the shortest distance is

$$d = \frac{(s_1 - p) \times (s_2 - p)}{|s_2 - p|}$$



Geometry - Points and lines

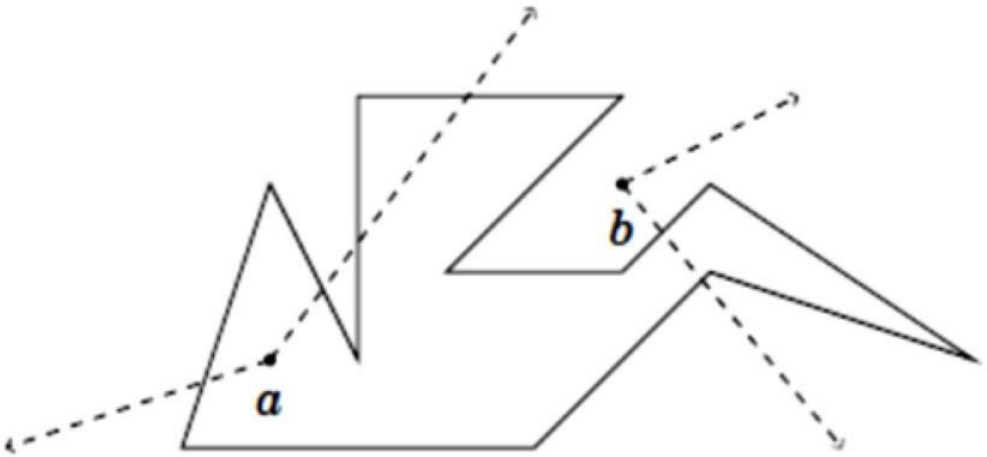
- Point inside a polygon
  - Let us now consider the problem of testing whether a point is located inside or outside a polygon. For example, point a is inside the polygon and point b is outside the polygon.



  - A convenient way to solve the problem is to send a ray from the point to an arbitrary direction and calculate the number of times it touches the boundary of the polygon. If the number is odd, the point is inside the polygon, and if the number is even, the point is outside the polygon.

Geometry - Points and lines

- Point inside a polygon
  - The rays from a touch 1 and 3 times the boundary of the polygon, so a is inside the polygon. Correspondingly, the rays from b touch 0 and 2 times the boundary of the polygon, so b is outside the polygon.



Geometry

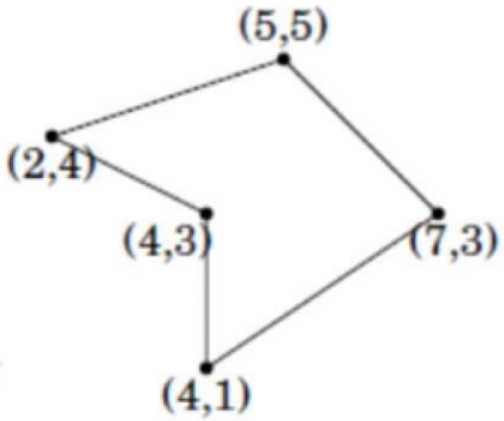
- Complex numbers
- Points and lines
- **Polygon area**
- Distance functions
- Intersection points
- Closest pair problem
- Convex hull problem

Polygon area

- A general formula for calculating the area of a polygon, sometimes called the shoelace formula, is as follows

$$\frac{1}{2}\left|\sum_{i=1}^{n-1}(p_i \times p_{i+1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

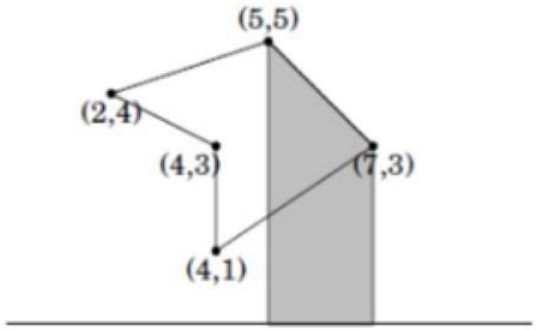
  - Here the vertices are $p_1 = (x_1, y_1), p2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$ in such an order that p_i and p_{i+1} are adjacent vertices on the boundary of the polygon, and the first and last vertex is the same, i.e., $p_1 = p_n$



$$\frac{|(2\cdot5-5\cdot4)+(5\cdot3-7\cdot5)+(7\cdot1-4\cdot3)+(4\cdot3-4\cdot1)+(4\cdot4-2\cdot3)|}{2} = 17/2$$

Polygon area

- The idea of the formula is to go through trapezoids whose one side is a side of the polygon, and another side lies on the horizontal line $y = 0$. For example:



- The area of such a trapezoid is

$$(x_{i+1} - x_i)\,\frac{y_i + y_{i+1}}{2}$$

where the vertices of the polygon are p_i and p_{i+1}. If $x_{i+1} > x_i$, the area is positive, and if $x_{i+1} < x_i$, the area is negative

Polygon area

- The area of the polygon is the sum of areas of all such trapezoids, which yields the formula

$$\left| \sum_{i=1}^{n-1} (x_{i+1} - x_i) \frac{y_i + y_{i+1}}{2} \right| = \frac{1}{2} \left| \sum_{i=1}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|$$

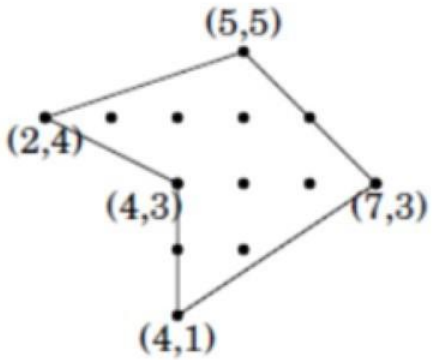
  - Note that the absolute value of the sum is taken, because the value of the sum may be positive or negative, depending on whether we walk clockwise or counterclockwise along the boundary of the polygon.

Polygon area

- Pick's theorem
  - Pick's theorem provides another way to calculate the area of a polygon provided that all vertices of the polygon have integer coordinates.
  - According to Pick's theorem, the area of the polygon is

$$a + \frac{b}{2} - 1$$

where a is the number of integer points inside the polygon and b is the number of integer points on the boundary of the polygon.



$6 + 7/2 - 1 = 17/2.$

Geometry

- Complex numbers
- Points and lines
- Polygon area
- **Distance functions**
- Intersection points
- Closest pair problem
- Convex hull problem

Distance functions

- A distance function defines the distance between two points.
  - The usual distance function is the Euclidean distance where the distance between points (x_1, y_1) and (x_2, y_2) is

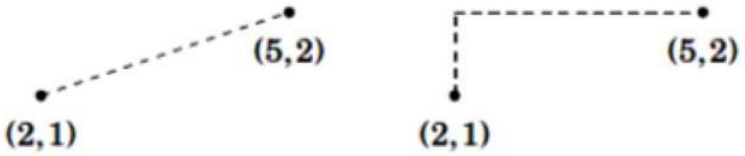
$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

  - An alternative distance function is the **Manhattan distance** where the distance between points (x_1, y_1) and (x_2, y_2) is

$$|x_2 - x_1| + |y_2 - y_1|$$

Distance functions

- For example
  - The Euclidean distance and the Manhattan distance



Euclidean distance $\sqrt{(5-2)^2+(2-1)^2}=\sqrt{10}$

Manhattan distance $|5-2|+|2-1|=4.$

  - Regions that are within a distance of 1 from the center point, using the Euclidean and Manhattan distances

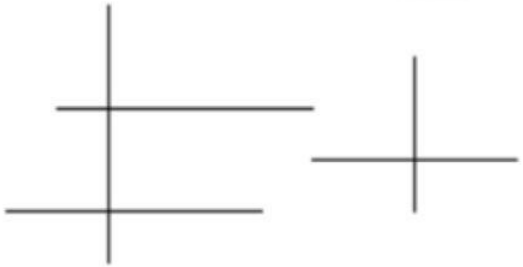
Euclidean distance **Manhattan distance**

Geometry

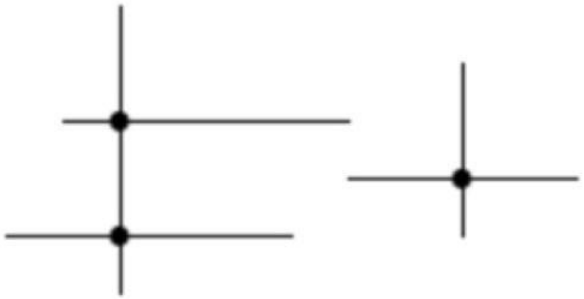
- Complex numbers
- Points and lines
- Polygon area
- Distance functions
- **Intersection points**
- Closest pair problem
- Convex hull problem

Intersection points

- Given a set of n line segments, each of them being either horizontal or vertical, consider the problem of counting the total number of intersection points.
  - For example, when the line segments are:



  - There are three intersection points



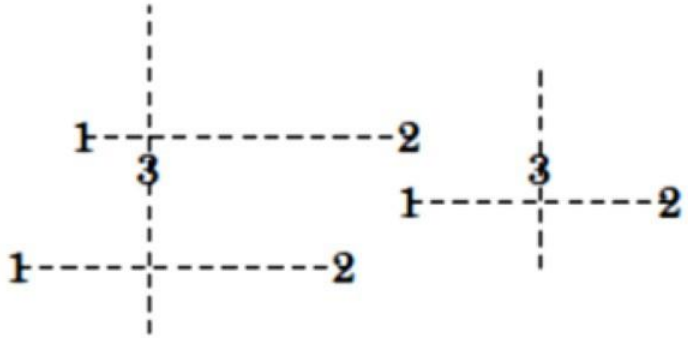
Intersection points

- It is easy to solve the problem in $O(n^2)$ time, because we can go through all possible pairs of line segments and check if they intersect.
  - However, we can solve the problem more efficiently in $O(nlogn)$ time using a sweep line algorithm and a range query data structure.
  - The idea is to process the endpoints of the line segments from left to right and focus on three types of events:
    - **horizontal segment begins**
    - **horizontal segment ends**
    - **vertical segment**

Intersection points

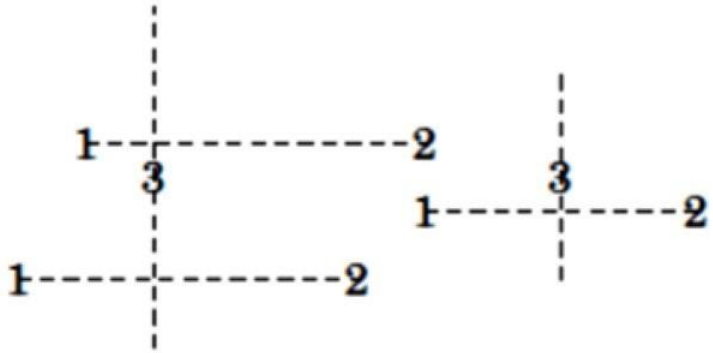
- We go through the events from left to right and use a data structure that maintains a set of y coordinates where there is an active horizontal segment.
  - At event 1, we add the y coordinate of the segment to the set, and at event 2, we remove the y coordinate from the set.
  - Intersection points are calculated at event 3.

-



Intersection points

- We go through the events from left to right and use a data structure that maintains a set of y coordinates where there is an active horizontal segment.
  - When there is a vertical segment between points y_1 and y_2, we count the number of active horizontal segments whose y coordinate is between y_1 and y_2 and add this number to the total number of intersection points.
  - To store y coordinates of horizontal segments, we can use a binary indexed or segment tree, possibly with index compression
  - When such structures are used, processing each event takes O(logn) time, so the total running time of the algorithm is O(nlogn)



Intersection points

- Sweep line algorithms
  - Many geometric problems can be solved using sweep line algorithms. The idea in such algorithms is to represent an instance of the problem as a set of events that correspond to points in the plane. The events are processed in increasing order according to their x or y coordinates.
  - As an example, consider the following problem: There is a company that has n employees, and we know for each employee their arrival and leaving times on a certain day. **Our task is to calculate the maximum number of employees that were in the office at the same time.**

Intersection points

- Sweep line algorithms
  - Example: The problem can be solved by modeling the situation so that each employee is assigned two events that correspond to their arrival and leaving times. After sorting the events, we go through them and keep track of the number of people in the office.

person	arrival time	leaving time
John	10	15
Maria	6	12
Peter	14	16
Lisa	5	13

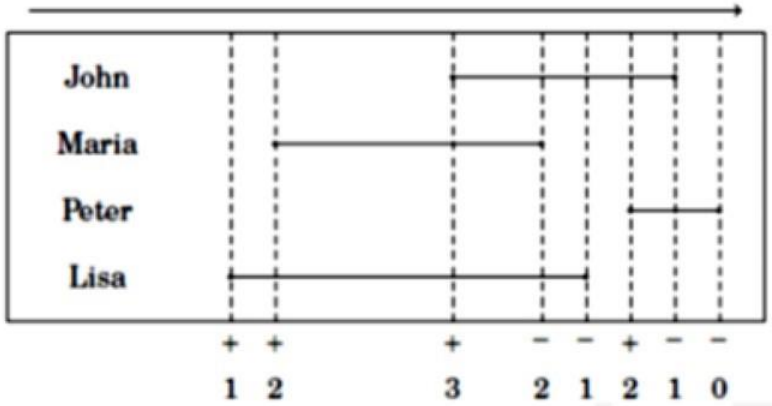
corresponds to the following events:

Intersection points

- Sweep line algorithms
  - Example:
    - We go through the events from left to right and maintain a counter.
      - Always when a person arrives, we increase the value of the counter by one, and
      - when a person leaves, we decrease the value of the counter by one. The answer to the problem is the maximum value of the counter during the algorithm

Intersection points

- Sweep line algorithms
  - Example:
    - The symbols + and − indicate whether the value of the counter increases or decreases, and the value of the counter is shown below.
    - The maximum value of the counter is **3 between John's arrival and Maria's leaving**
    - The running time of the algorithm is O(nlogn), because sorting the events takes O(nlogn) time and the rest of the algorithm takes O(n) time.

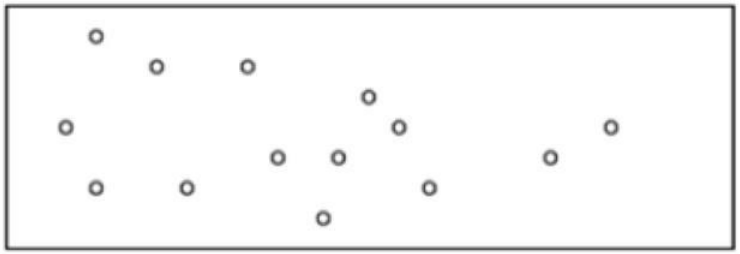


Geometry

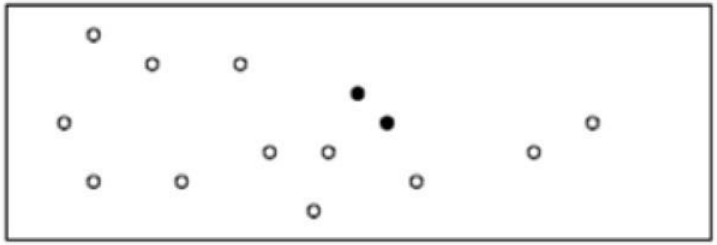
- Complex numbers
- Points and lines
- Polygon area
- Distance functions
- Intersection points
- **Closest pair problem**
- Convex hull problem

Closest pair problem

- Given a set of n points, our next problem is to find two points whose Euclidean distance is minimum.
  - For example, if the points are



  - Should find the following points:

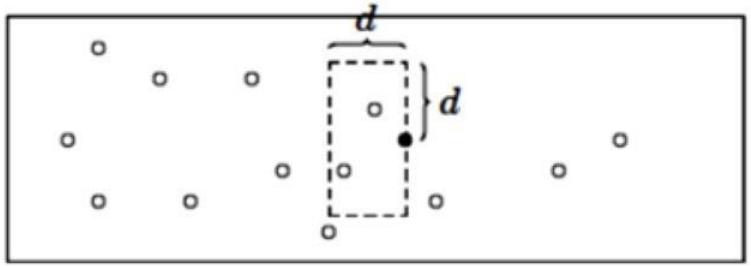


Closest pair problem

- This is another example of a problem that can be solved in $O(nlogn)$ time using a sweep line algorithm.
  - We go through the points from left to right and maintain a value d: the minimum distance between two points seen so far
  - At each point, we find the nearest point to the left.
    - If the distance is less than d, it is the new minimum distance and we update the value of d

Closest pair problem

- If the current point is (x, y) and there is a point to the left within a distance of less than d,
  - the x coordinate of such a point must be between $[x - d, x]$ and the y coordinate must be between $[y - d, y + d]$.
  - Thus, it suffices to only consider points that are located in those ranges, which makes the algorithm efficient.



the region marked with dashed lines contains the points that can be within a distance of d from the active point

Closest pair problem

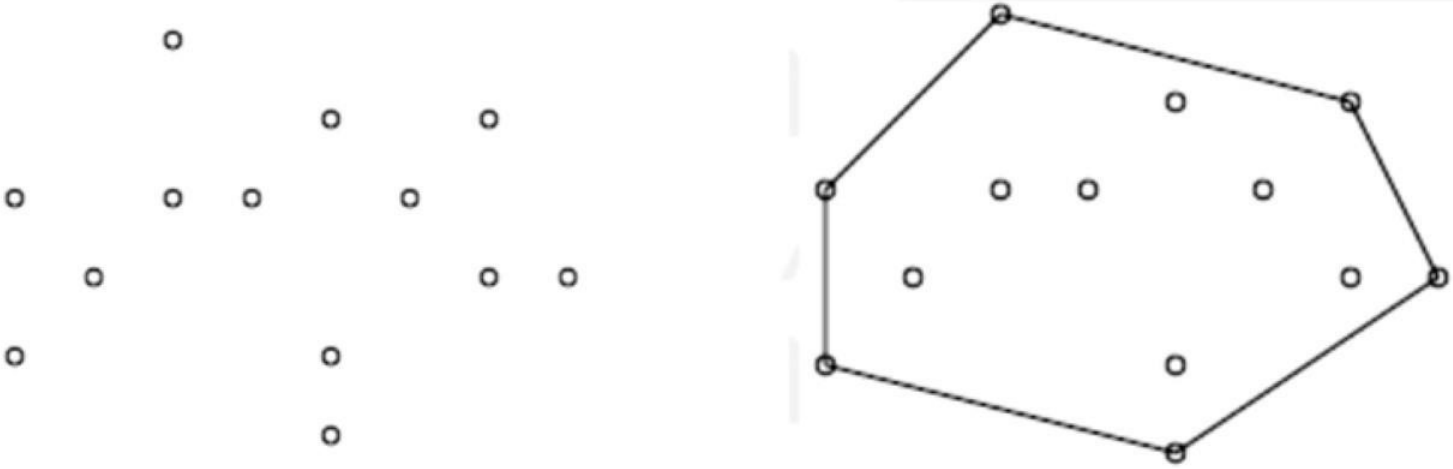
- The efficiency of the algorithm is based on the fact that the region always contains only $O(1)$ points.
  - We can go through those points in $O(logn)$ time by maintaining a set of points whose x coordinate is between $[x-d, x]$, in increasing order according to their y coordinates.
  - The time complexity of the algorithm is $O(nlogn)$ because we go through n points and find for each point the nearest point to the left in $O(logn)$ time.

Geometry

- Complex numbers
- Points and lines
- Polygon area
- Distance functions
- Intersection points
- Closest pair problem
- **Convex hull problem**

Convex hull problem

- A **convex hull** is the smallest convex polygon that contains all points of a given set.
  - Convexity means that a line segment between any two vertices of the polygon is completely inside the polygon.
  - For example, for the points and the convex hull



Convex hull problem

- Closure operator
  - The convex-hull operator has the characteristic properties of a closure operator
    - It is extensive, meaning that the convex hull of every set X is a superset of X
    - It is non-decreasing, meaning that, for every two sets X and Y with $X \subseteq Y$, the convex hull of X is a subset of the convex hull of Y
    - It is idempotent, meaning that for every X, the convex hull of the convex hull of X is the same as the convex hull X.

Convex hull problem

- Andrew's algorithm:
  - Provides an easy way to construct the convex hull for a set of points in $O(nlogn)$ time.
  - The algorithm first locates the leftmost and rightmost points, and then constructs the convex hull in two parts:
    - first the upper hull and then the lower hull.
    - Both parts are similar, so we can focus on constructing the upper hull.

Convex hull problem

- Andrew's algorithm:
  - First, we sort the points primarily according to x coordinates and secondarily according to y coordinates
  - After this, we go through the points and add each point to the hull. Always after adding a point to the hull, we make sure that the last line segment in the hull does not turn left
  - As long as it turns left, we repeatedly remove the second last point from the hull

Convex hull problem

- Andrew's algorithm: