Algorithms

Geometric Algorithms
(pp. 265~281)

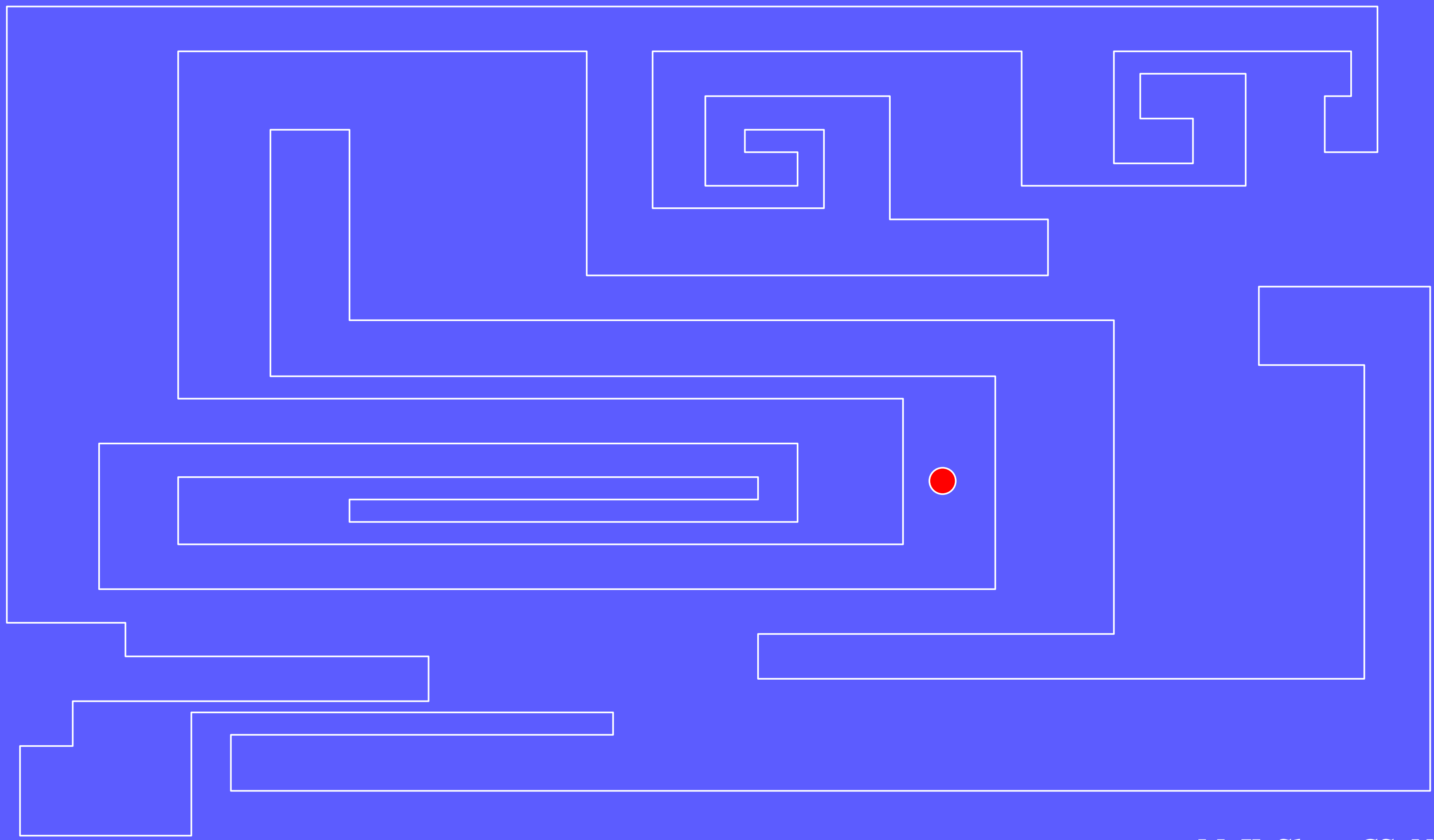
Geometric Algorithms

- Applications
  - computer graphics
  - computer-aided design
  - VLSI design
  - robotics
  - databases

Determining Whether a Point is Inside a Polygon

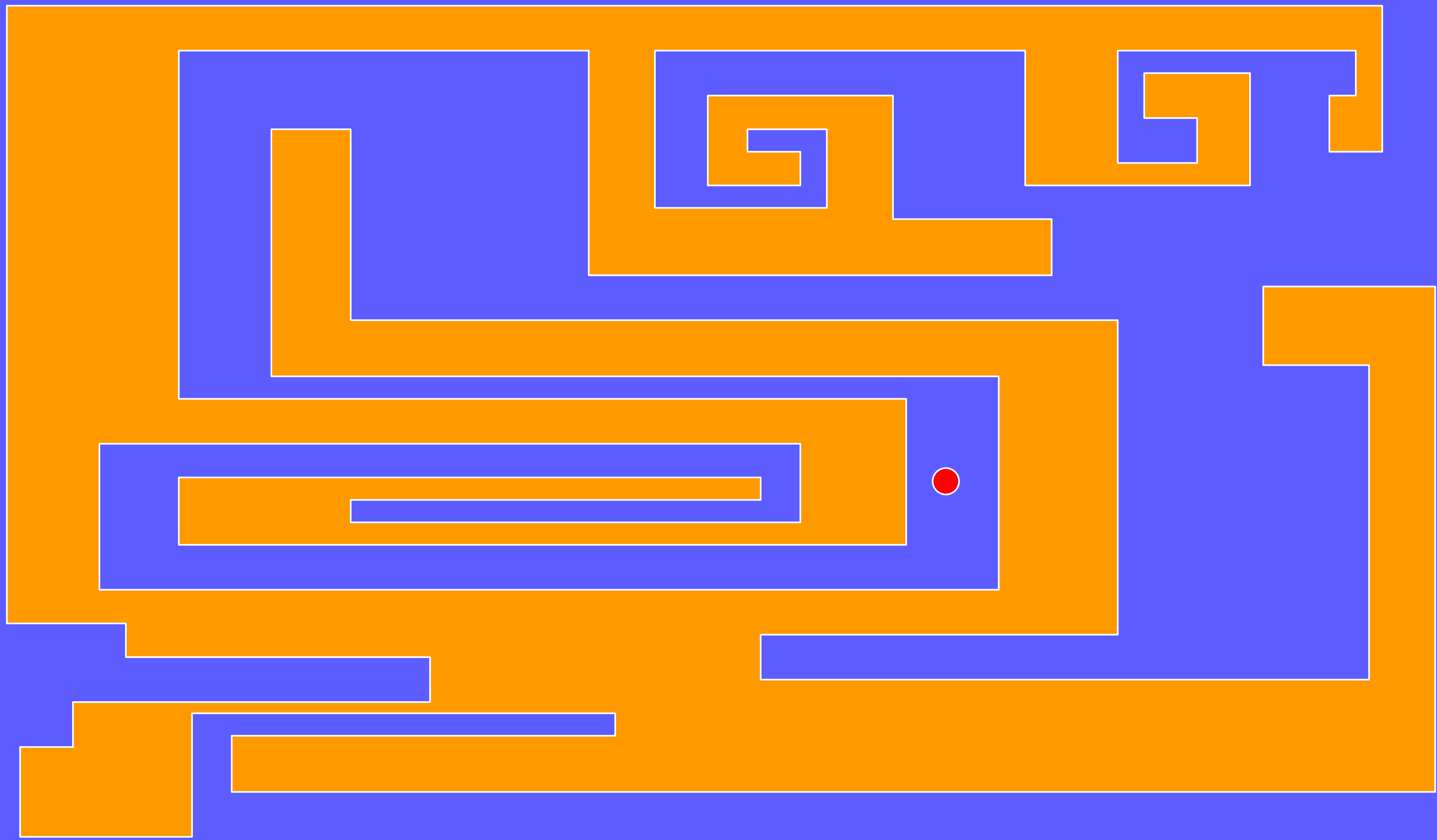
(pp. 266~270)

Determining Whether a Point is Inside a Polygon

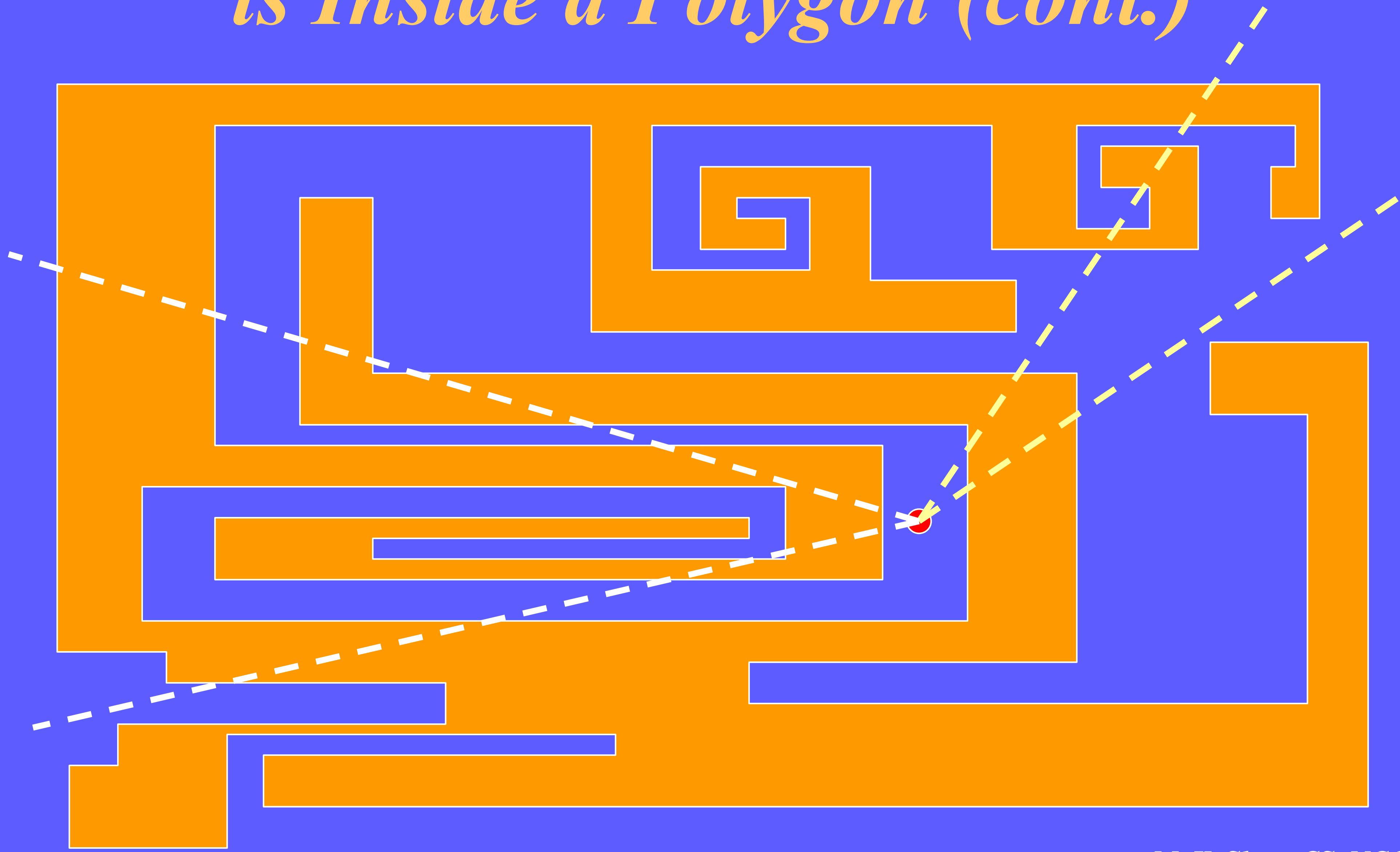


M. K. Shan, CS, NCCU

Determining Whether a Point is Inside a Polygon (cont.)

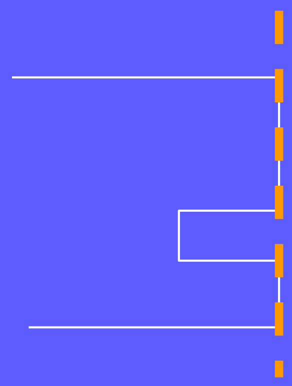


Determining Whether a Point is Inside a Polygon (cont.)

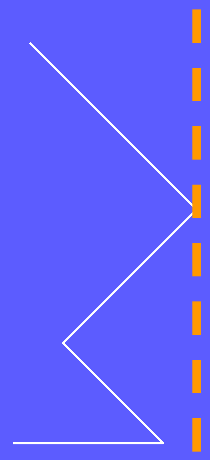


Observation

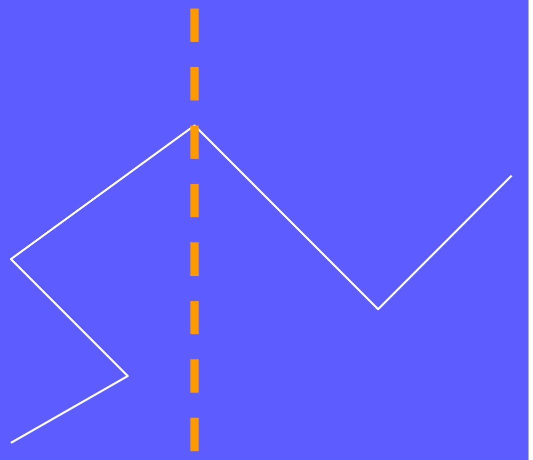
- Point is inside the polygon
  iff #(intersections) is odd
- Special cases
  - (1)



(2)



(3)



Algorithm Point_in_Polygon_1(P, q);

Input: P(a polygon with vertices p_1 , p_2, …, p_n and edges e_1, e_2,…, e_n)

q=(x_0, y_0)

Output: Inside

Begin

Pick an arbitrary point s outside the polygon

Let L be the line segment q-s;

Count:=0;

For all edges e_i of the polygon do

If e_i intersects L then

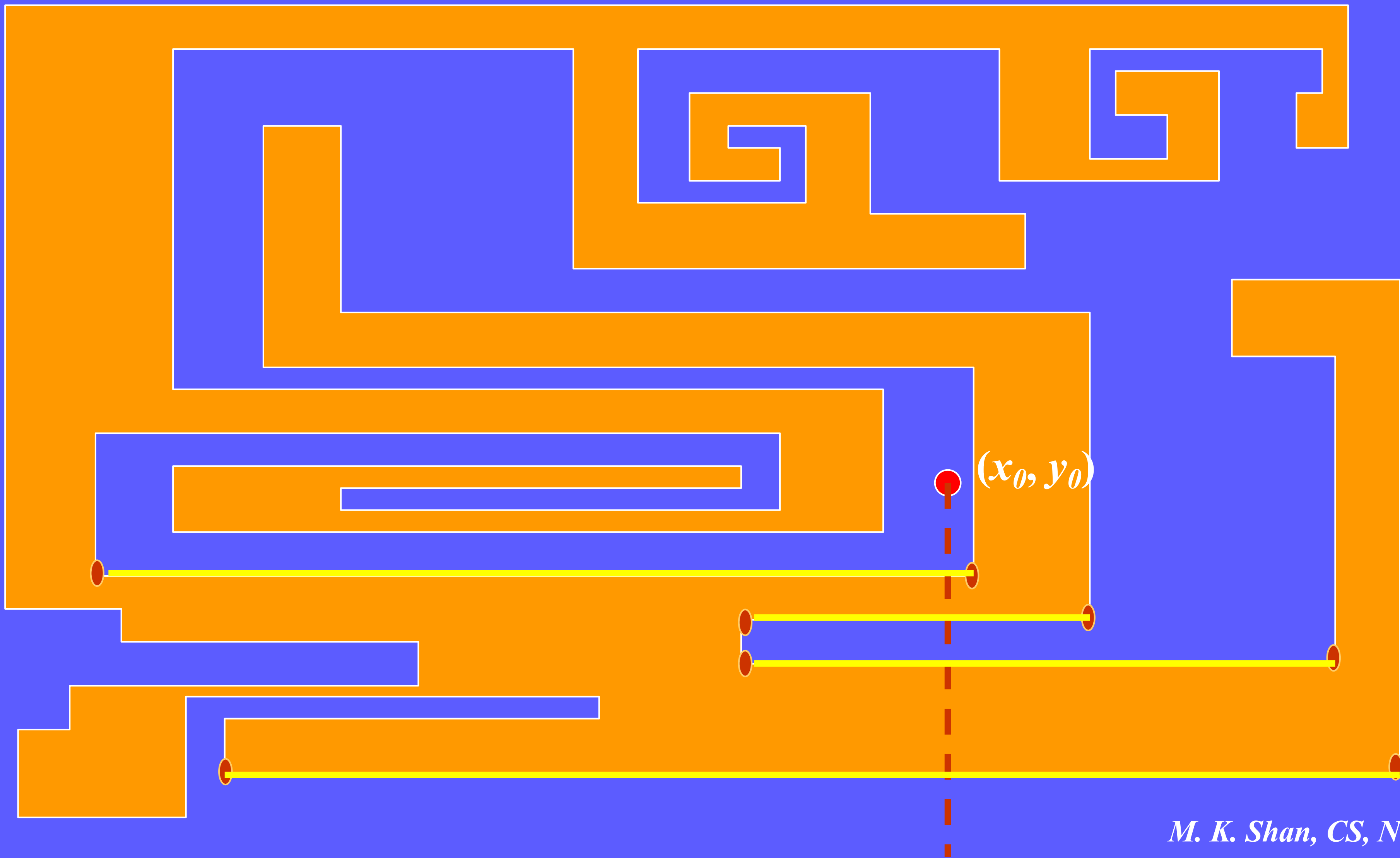
Increment count;

If count is odd then Inside:=true;

else Inside:=false

End

Determining Whether a Point is Inside a Polygon (cont.)



Check for Intersection

- Given q=(368, 308)

  look at all edges & check edges whose

  - y coordinate < 308 and
  - x coordinate cross 368

e.g. (208, 280)-(384, 280)

(416, 272)-(320, 272)

(320, 256)-(448, 256)

(452, 224)-(256,224)

Algorithm Point_in_Polygon_2(P, q);

Input: P(a polygon with vertices p_1 , p_2, …, p_n and edges e_1, e_2,…, e_n)

$q=(x_0, y_0)$

Output: Inside

Begin

count:=0;

For all edges e_i of the polygon do

If the line $x = x_0$ intersects e_i then

Let y_i be the y coordinates of the intersection between line x= x_0 and e_i

If $y_i < y_0$ then

Increment count;

If count is odd then Inside:=true;

else Inside:=false

End

Complexity

- O(n) for n segment polygon

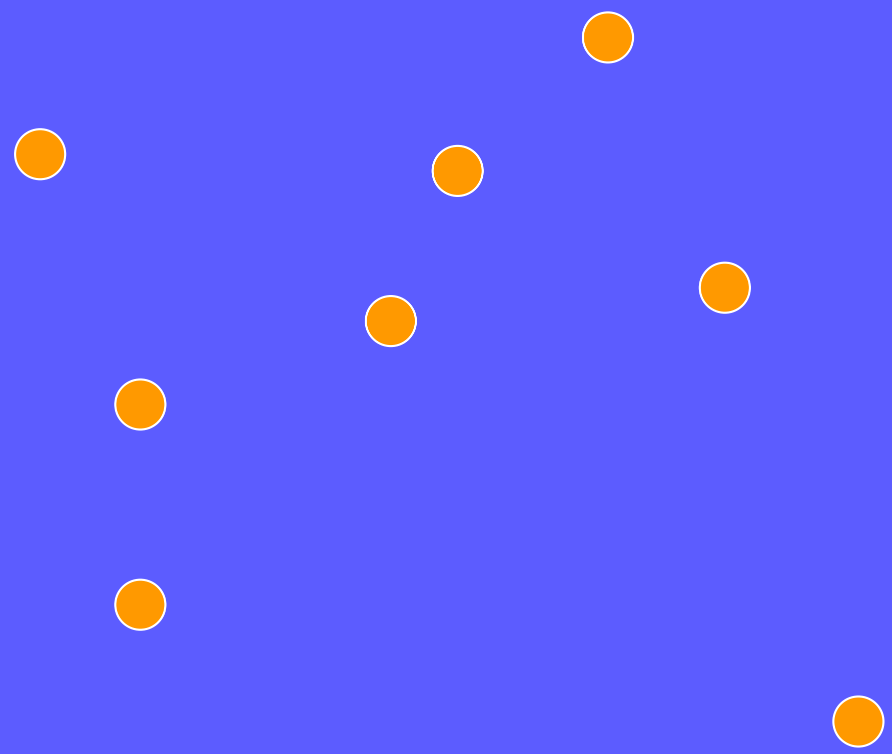
=> Compute n intersections

Constructing Simple Polygons

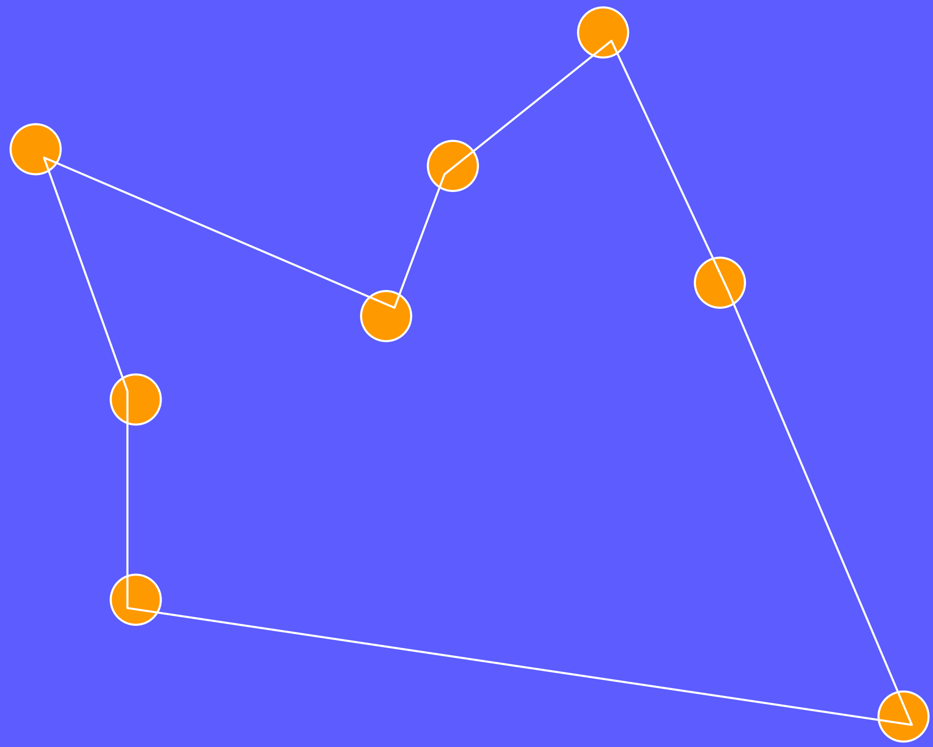
Geometric Algorithms

Constructing Simple Polygons

- Given a set of n points, connect them in a simple closed path

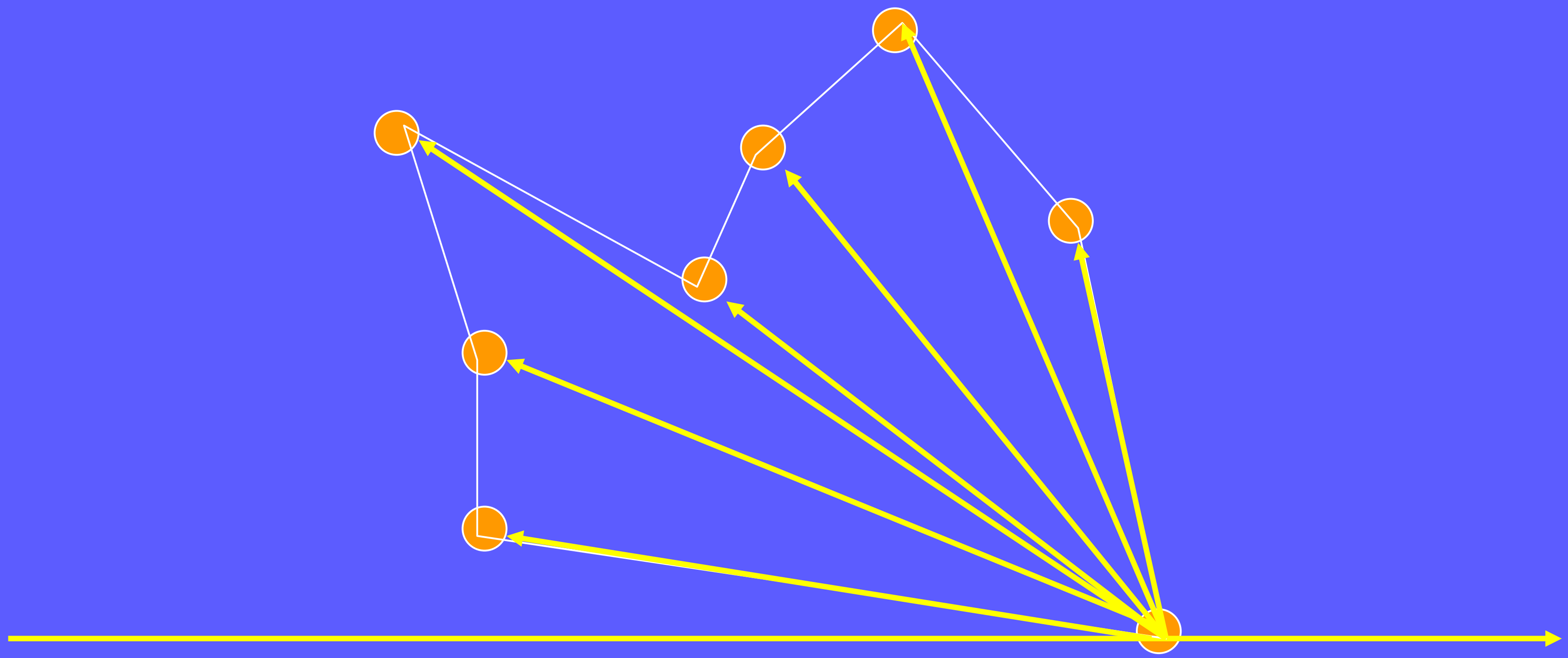


Constructing Simple Polygons



M. K. Shan, CS, NCCU

Constructing Simple Polygons



Algorithm Simple_Polygon

Input: p_1 , p_2, …, p_n

Output: P

Begin

For i:=2 to n do

Compute the angle α_i between line -p_1- p_i- and the x-axis

Sort the points according to the angles α_1, α_2, …, α_n

P is the polygon defined by the list of points in sorted order

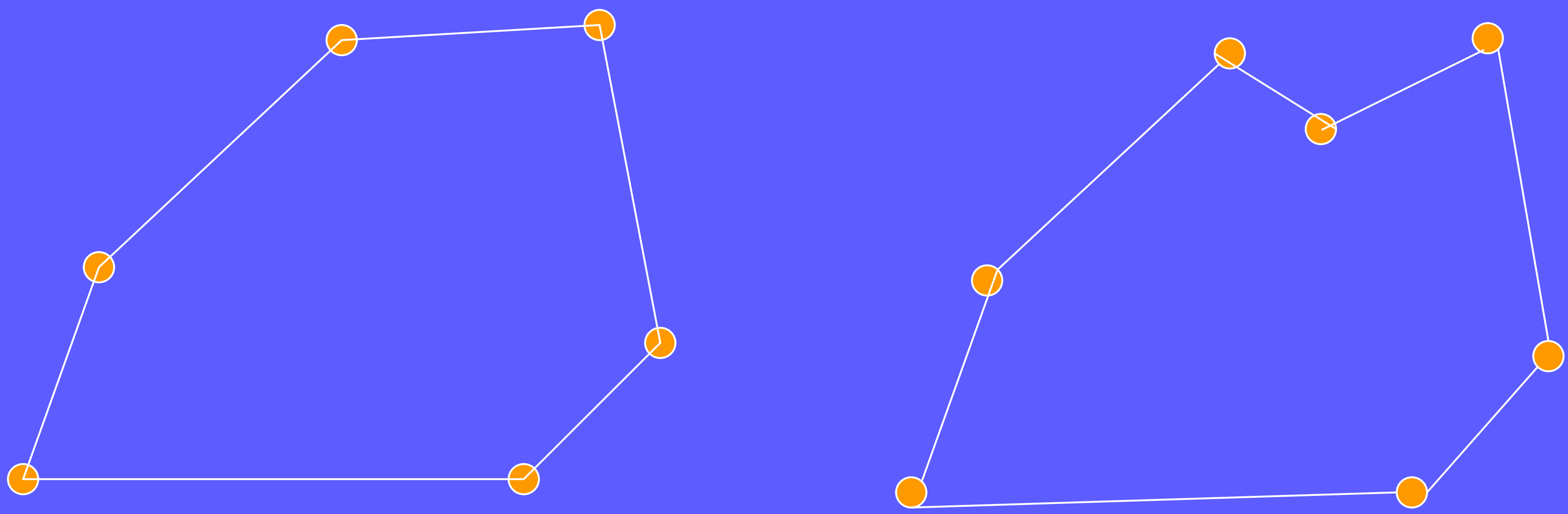
End

Complexity: O(nlogn)

Convex Hull

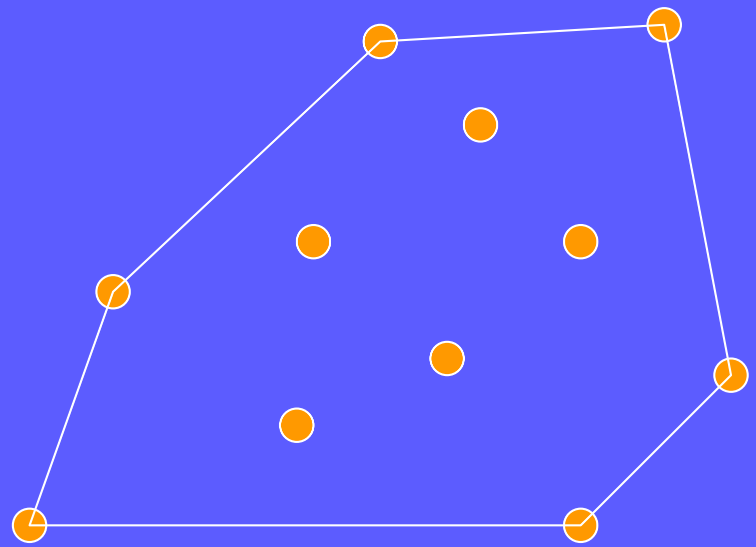
(pp. 273~277)

Convex Hull



Convex Hull

- Convex hull of a set of points
  - the smallest convex polygon enclosing all the points
  - be represented as a regular polygon, i.e., vertices should be listed in cyclic order
  - the vertices of the convex hull are points from the set

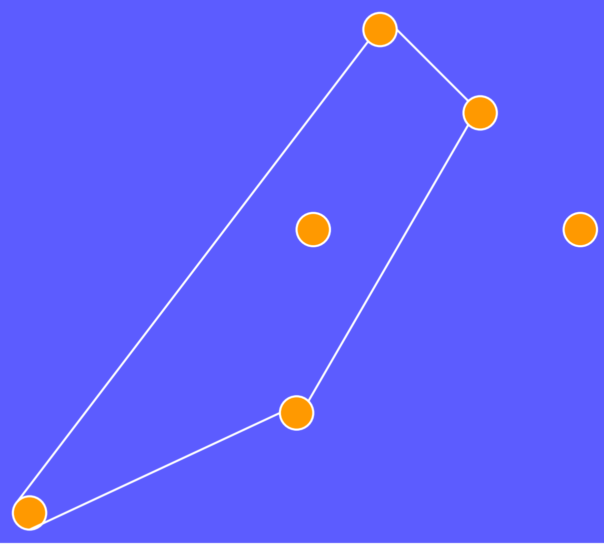


Convex Hull Problem

- **Given n points in the plane, compute the convex hull of the given points**
- **Approaches**
  - **straightforward: $O(n^2)$**
  - **gift wrapping: $O(n^2)$**
  - **Graham's scan: $O(n\log n)$**

Straightforward Approach

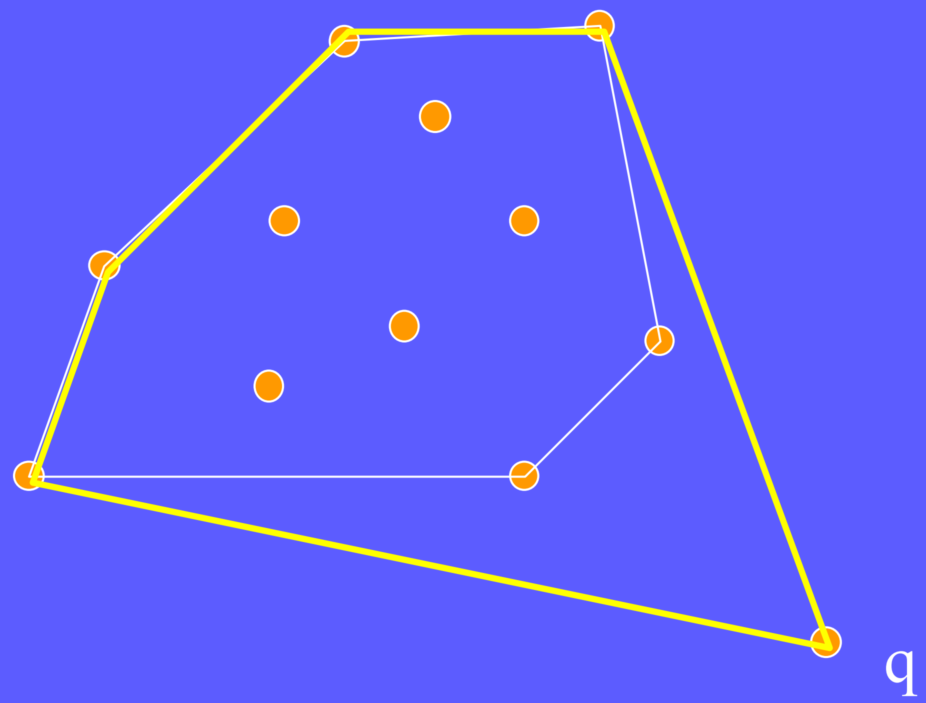
- **By induction**
  - **n=3, convex hull of 3 points**
  - **hypothesis: we know how to compute convex hull of < n points**
  - **Induction: the n-th point + convex hull of (n-1) points?**
    - Case 1: the n-th point is inside the convex hull
      => the new convex hull unchanged
    - Case 2: the n-th point is outside the convex hull
      => the new convex hull is stretched to reach that point



Straightforward Approach

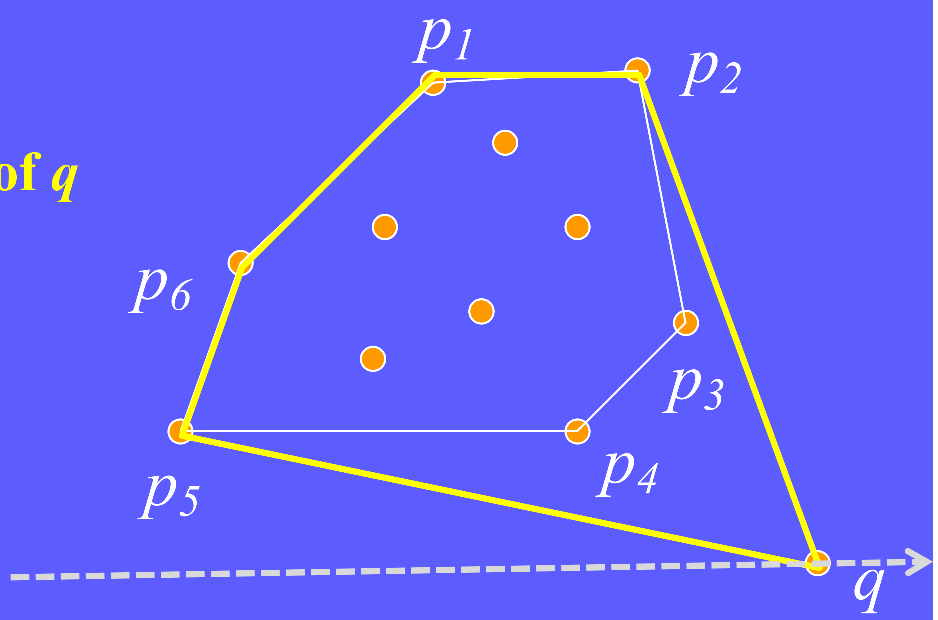
- Problem: determine whether a point is inside the hull?
- Improvement
  - choose a special n-th point rather than an arbitrary one
  - choose extreme point, e.g., point with maximal x coordinate

Stretching a Convex Polygon



Stretching a Convex Polygon (cont.)

- **Stretch the hull to include point *q***
  - **remove the vertices of old hull that are inside the new hull**
  - **=> remove the vertices between two intersection points of supporting line**
  - **insert *q* between two existing vertices**
  - **=> Insert *q* between two intersection points of supporting line**
- **Supporting line**
  - **line that intersects hull at one vertex**
  - **given point *q***
  - **=> hull lies between two supporting line of *q***
  - **the maximal & minimal angles from points to *q***

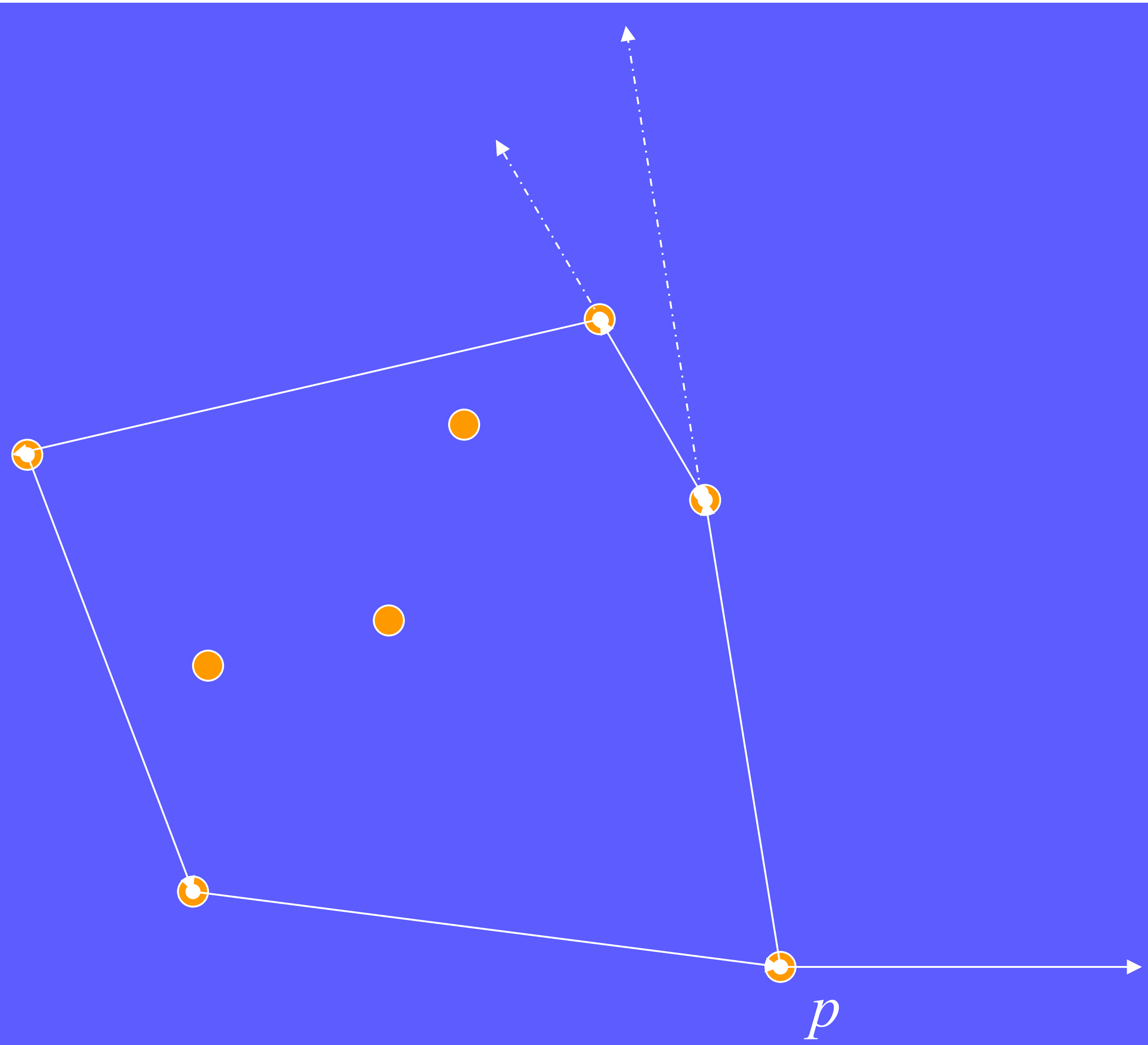


Complexity of Straight Forward Approach

- **For each point, compute angles to all previous points to find the maximal & minimal angles**
- **the k-th point takes O(k)**
- **$T(n)=T(n-1)+O(n)$**
- **complexity $O(n^2)$**

Gift Wrapping

- **Observation of straightforward approach**
- **Improvement: gift wrapping**
  - **start with an extreme point**
  - **find its neighbors in the hull by finding supporting lines**


p

Gift Wrapping

- **Induction hypothesis**

  Given a set of n points,

  we can find a convex path of length k < n

  (that is part of convex hull of this set)

- **Extending a convex path,**

  rather than extending the hull

- **Finding a part of the convex hull,**

  rather than finding convex hulls of smaller sets

Algorithm Gift-Wrapping($p_1, p_2, \ldots, p_n$)

Input: $p_1, p_2, \ldots, p_n$

Output: P **(the convex hull of** $p_1, p_2, \ldots, p_n$**)**

Begin

  P:={ };

  Let p **be the point in the set with the largest x coordinate**

  Add p **to** P

  Let L **be the line containing p which is parallel to the x-axis**

  While P **is not complete do**

    Let q **be the point such that the angle between line** –p-q- **and** L **is minimal**

    Add q **to** P

    L:=line –p-q-;

    p:=q

End

Complexity of Gift Wrapping

- **To add the k-th point to the hull**

  => find the minimal & maximal angles among (n-k) lines
- **Complexity of gift-wrapping: $O(n^2)$**

Graham's Scan

- **Comparing with gift wrapping,**
  - **Similar: maintains the convex path**
  - **Different**
    - the convex path is part of the convex hull of points that were scanned so far
    - the path may contains points that are not on the final convex hull

Graham's Scan

■Induction hypothesis

Given a set of *n* points, ordered according to Simple_Polygon, we can find a convex path among the first *k* points whose corresponding convex hull encloses the first *k*-points

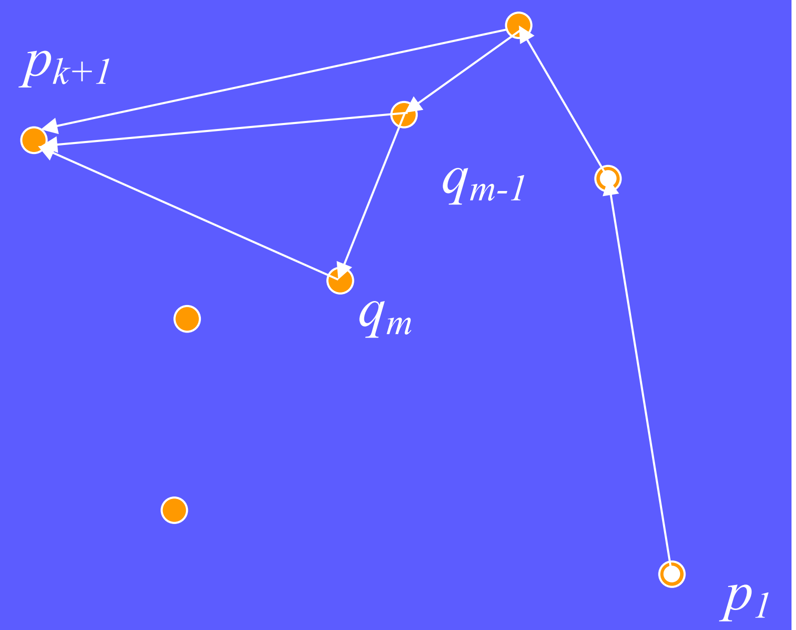
- □ **base case, k=1, trivial**
- □ **induction from k to k+1**

If $\angle q_{m-1} q_m p_{k+1} \le 180°$

Then add p_{k+1} to existing path

Else remove q_m and add p_{k+1}

continue the process



Algorithm Graham's Scan($p_1, p_2, \ldots, p_n$)

Input: $p_1, p_2, \ldots, p_n$

Output: $q_1, q_2, \ldots, q_m$ **(the convex hull of** $p_1, p_2, \ldots, p_n$**)**

Begin

  Let p_1 **be the point in the set with the largest x coordinate**

  Sort points around p_1 **to** $p_1, p_2, \ldots, p_n$

  $q_1 := p_1$;

  $q_2 := p_2$;

  $q_3 := p_3$;

  For k:= 4 to n do

    while the angle between $-q_{m-1}- q_m-$ **and** $-q_m- p_k-$ **is** $\geq 180°$ **do**

      $m := m-1$;

    $m := m+1$;

    $q_m := p_k$;

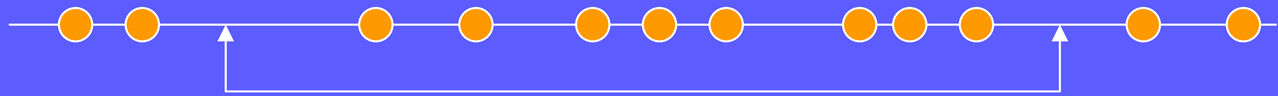
End

Complexity of Graham's Scan

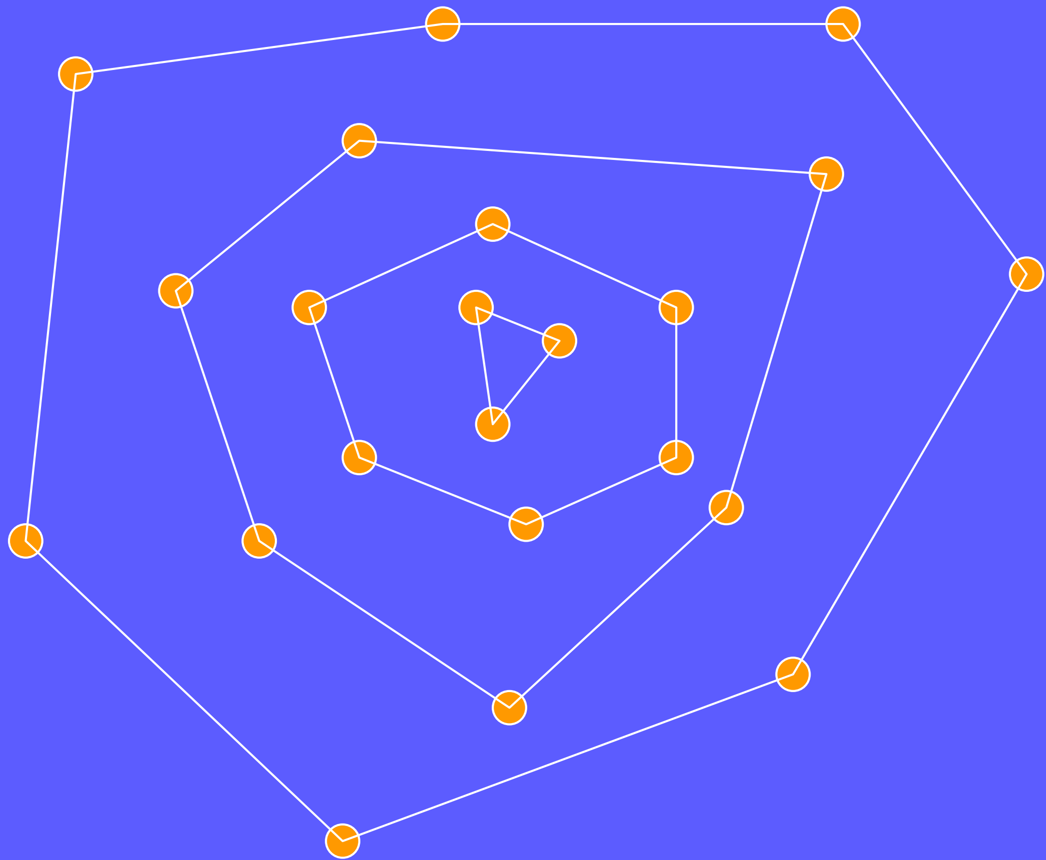
- **O($n$log$n$): O($n$log$n$)+O(n)**
  - **sorting: O($n$log$n$)**
  - **induction steps: O(n)**
    - Constant time to add point
    - Backward test: constant time to eliminate points

Applications of Convex Hulls to Statistics

- Robust estimation: remove outlier
  - 1D:



  - 2D:



Closest Pair

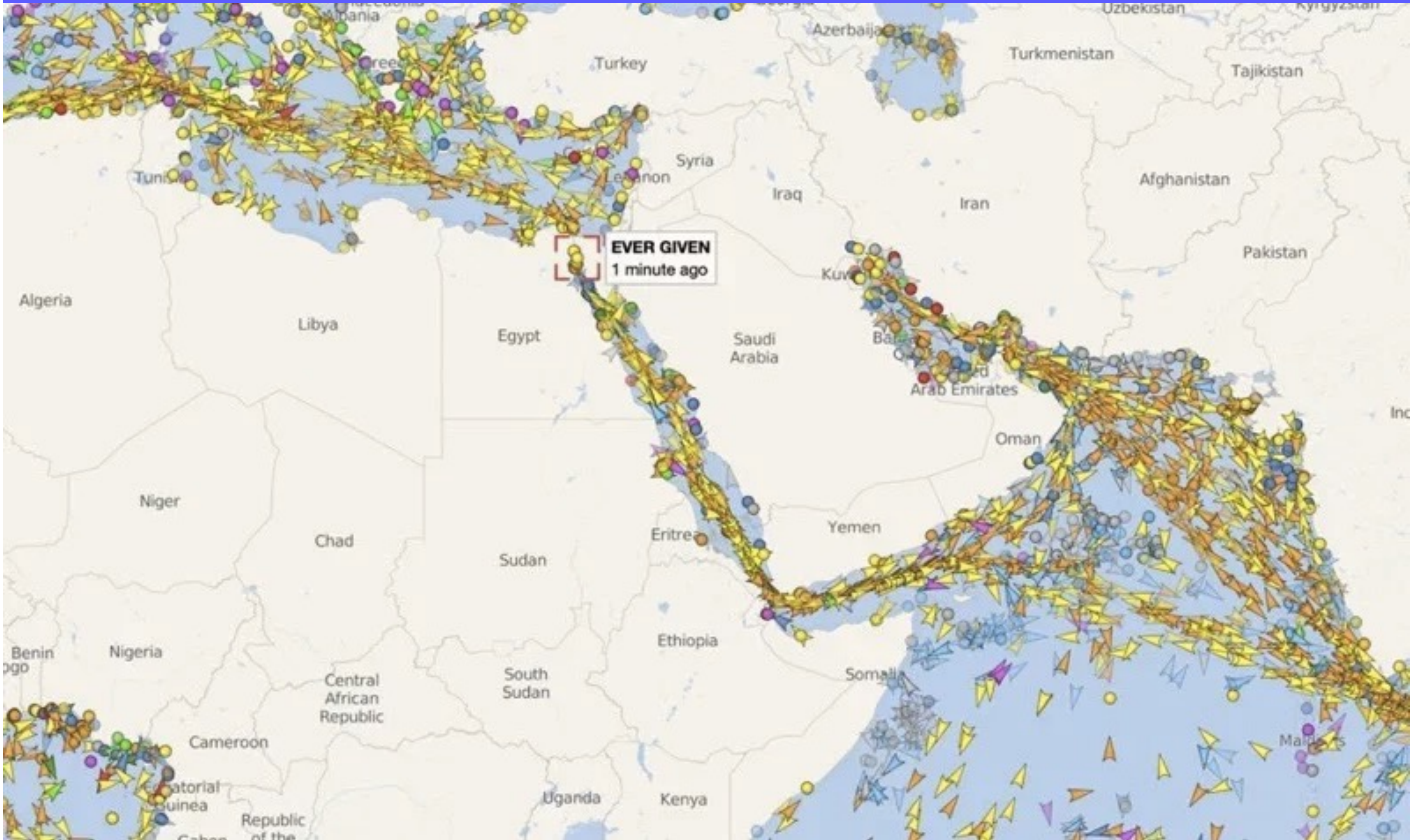
Geometric Algorithms

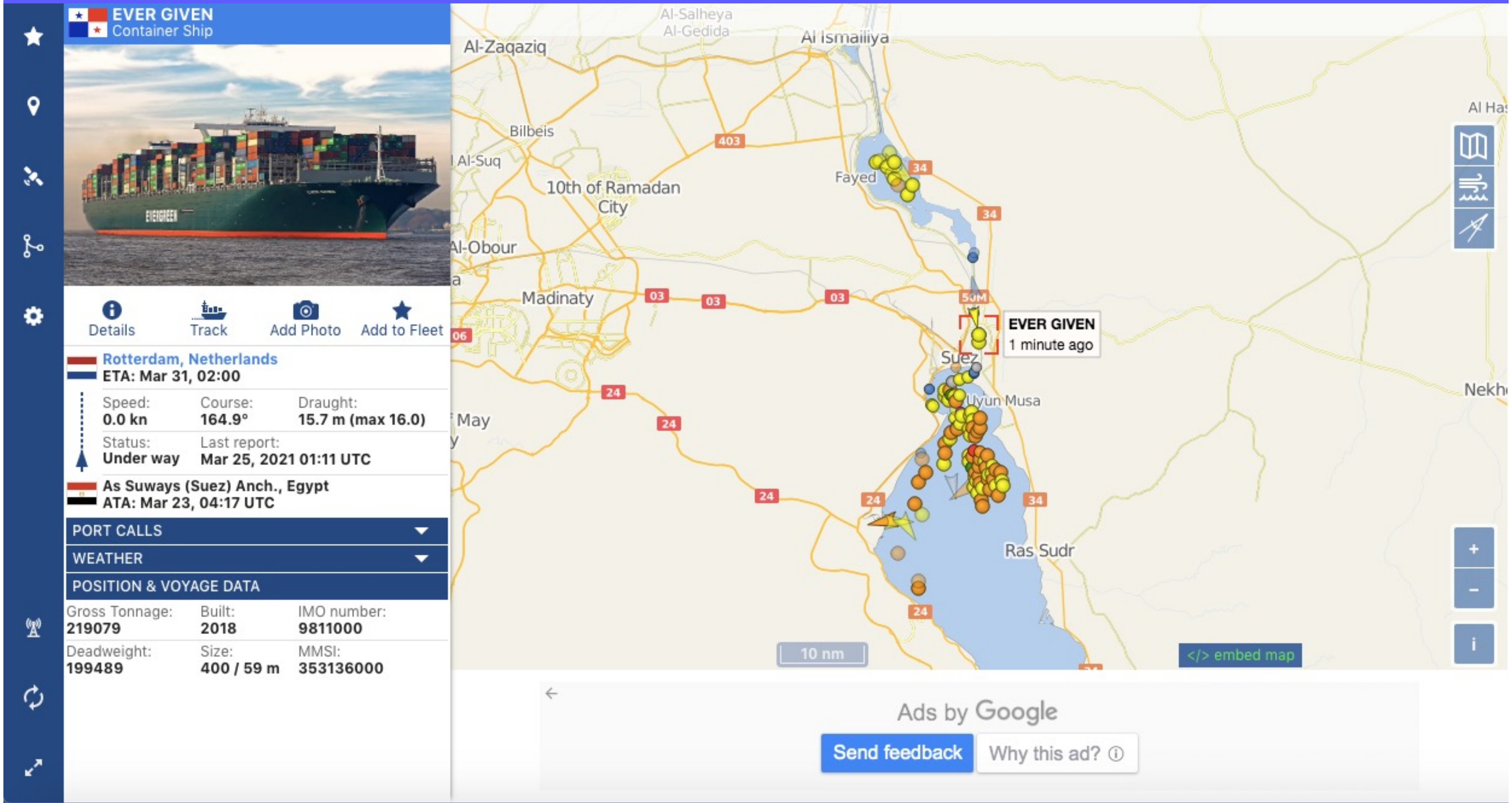

CAL6 B77W
9450 989
TPE LAX 13:40
EVA316 A332
430 278
BNE TPE 18:50
CAL122 B744
10050 952
TPE OKA 20:04
CSZ9070 B738
ESTIMATED
TPE JJN 19:20
VJC842 A320
10050 859
SGN TPE 19:24
100 km
50 mi
© 2017 FlightAware
© OpenStreetMap contributors
Weather: 2017年 06月 14日 10:40 UTC

US GOVT VESSEL (US) 15.5 kn, 2h 47' ago
LIANJIANG
连江县
Shijinggang
石井港
Sejiang Harbor
PINGTAN
平潭县
Xinghua Bay
兴化湾
Taipei
台北
Taoyuan City
Hsinchu City
新竹市
YILAN COUNTY
宜蘭縣
10 NM
Google


EVERGREEN


BARAKA 1
MOSAED 2
CURRENT BUOYS
KM 151.000
EVER GIVEN
1 minute ago
KM 151.000
SVITZER PORT SAID 2
TAHIA MISR2
TAHIA MISR 1
MOSAHEB 2


Turkey
Azerbaijan
Turkmenistan
Tajikistan
Syria
Iraq
Iran
Afghanistan
Pakistan
EVER GIVEN
1 minute ago
Algeria
Libya
Egypt
Saudi
Arabia
Arab Emirates
Oman
Niger
Chad
Sudan
Eritrea
Yemen
Ethiopia
Benin
Nigeria
Central
African
Republic
South
Sudan
Cameroon
Uganda
Kenya



M. K. Shan, CS, NCCU

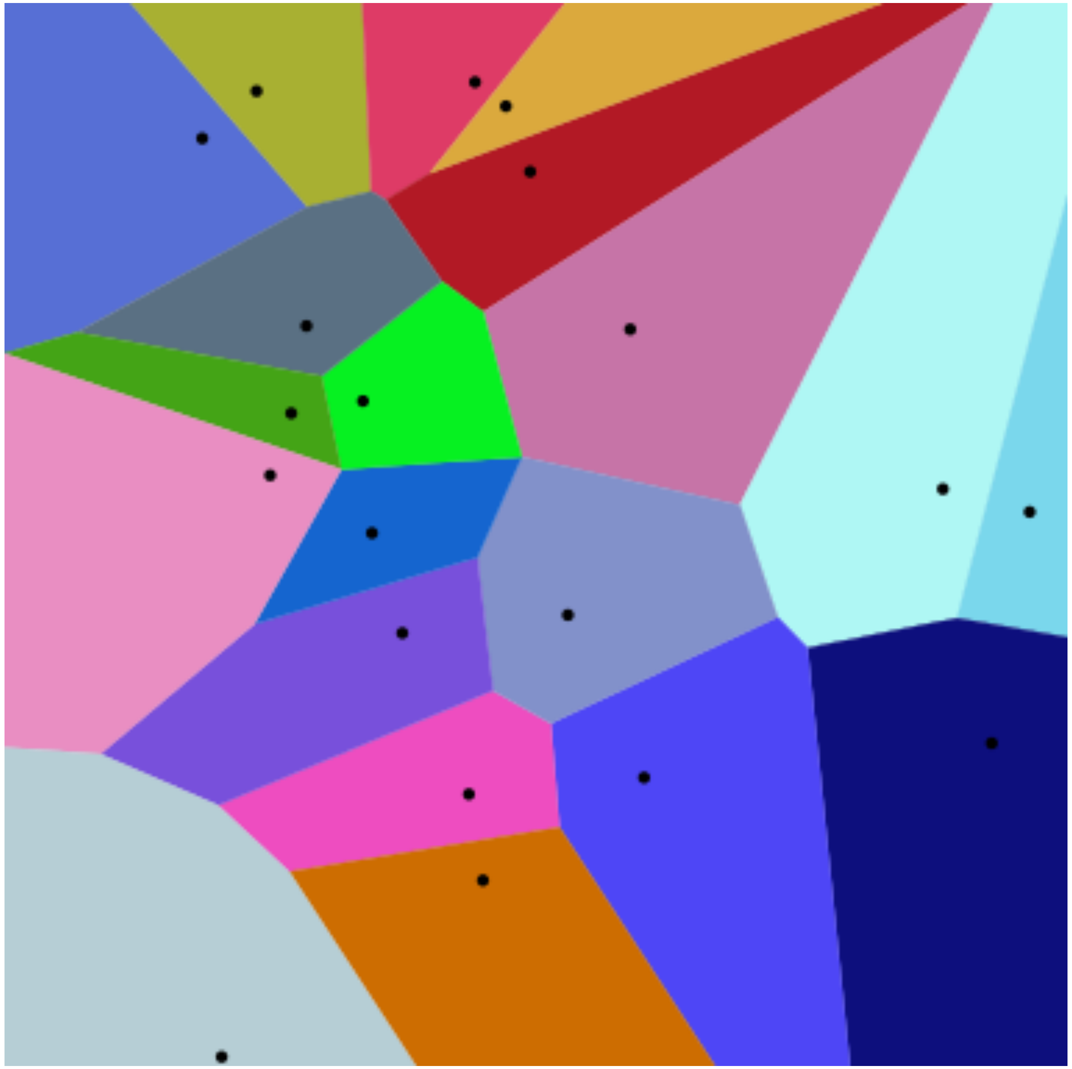
Closest Pair

- Given a set of points in the plane
  find a pair of closet points.
- Other proximity problems
  - finding the closet point to the query point
  - finding k-closet points to the query point (K-Nearest Neighbor)
  - finding the similar objects to the query object
  - finding k-similar objects to the query object
  - finding the objects intersecting with the query object
  - finding the objects enclosed in the query object
  - finding the objects enclosing the query object

Voronoi Diagram

- A Voronoi diagram for a given set of points
  - a division of plane into regions such that each region contains all points that are closest to one of the points from the set
  - are useful for a variety of proximity problems
  - constructed based on perpendicular bisector (垂直平分線)
  - can be constructed in O(nlogn) time
  - informal use of Voronoi diagrams can be traced back to Descartes in 1644

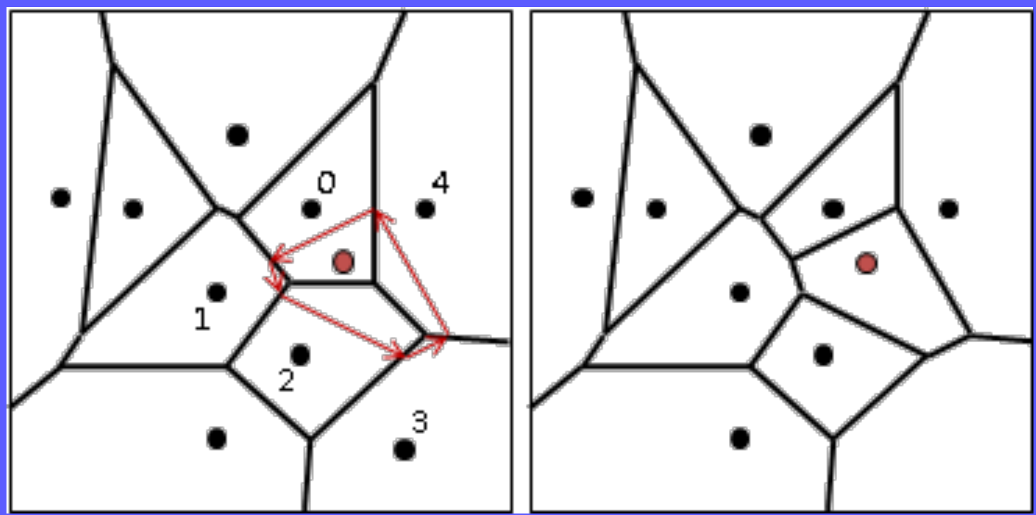
Voronoi Diagram
(Euclidean Distance)



Voronoi Diagram
（Manhattan Distance）

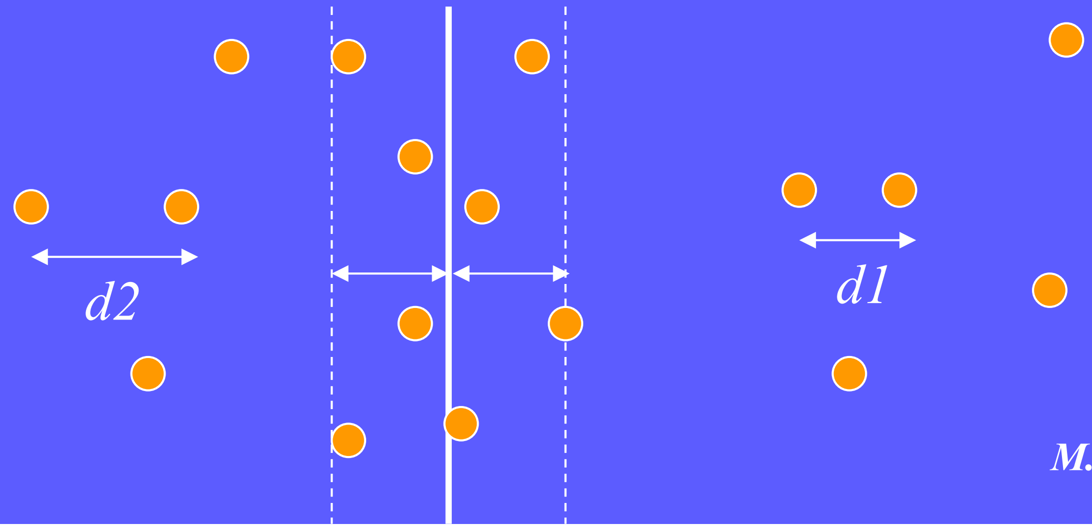
Voronoi Diagram (cont.)

- constructed based on perpendicular bisector (垂直平分線)
- can be constructed in O(nlogn) time
- informal use of Voronoi diagrams can be traced back to Descartes in 1644



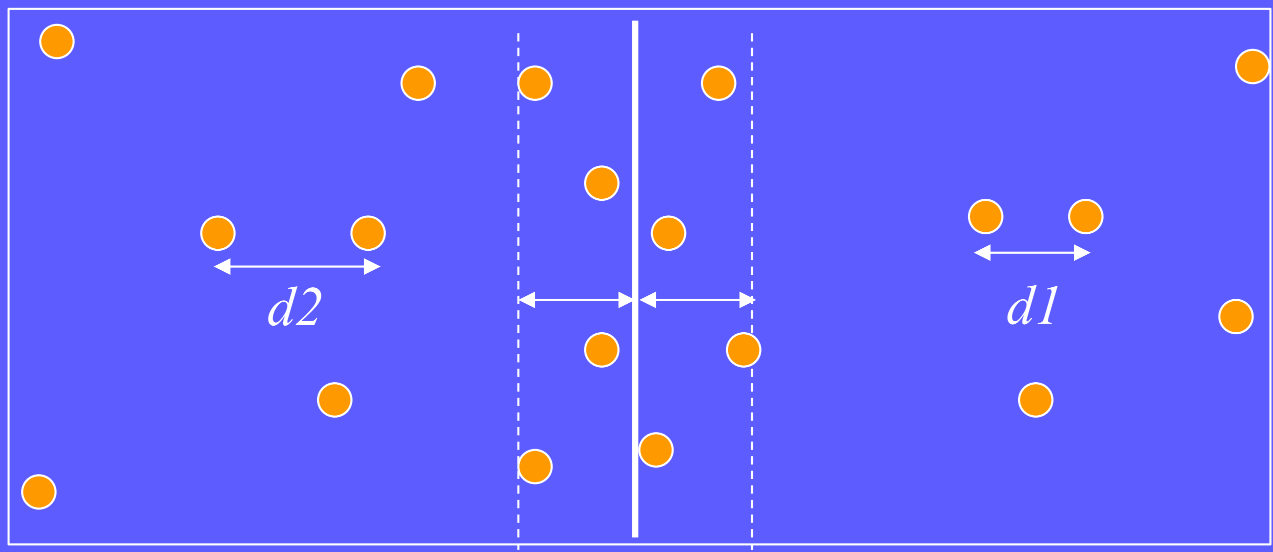
Approaches to Closest_Pair

- **Straightforward approach**
  - **check distance between each pairs**
  - **$O(n^2)$**
- **Divide-and-Conquer approach**
  - **divide the point set by dividing the plane into two disjoint parts**
  - **finding the minimal distance in each part recursively**
  - **need to concern with distance between points close to boundaries**



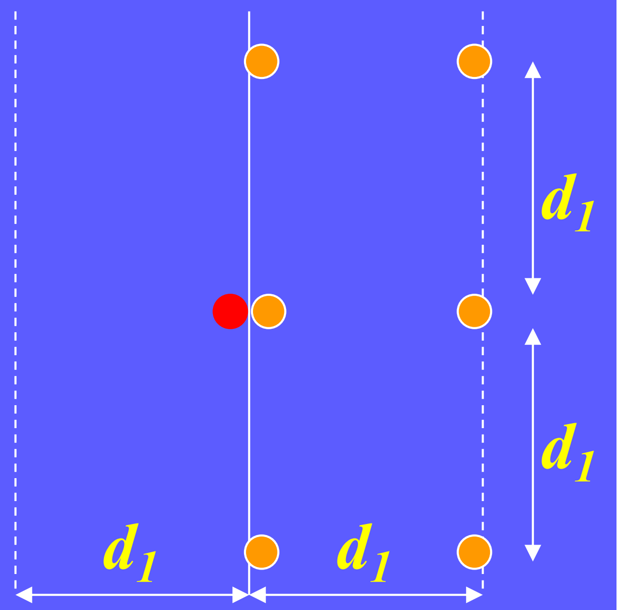
Approaches to Closest_Pair

- **Divide-and-Conquer approach**
  - **divide the point set by dividing the plane into two disjoint parts**
  - **finding the minimal distance in each part recursively**
  - **need to concern with distance between points close to boundaries**



Check Boundary

- Let the minimal distance in two subsets be d_1, d_2, respectively, without loss of generality, assume , $d_1 \leq d_2$
- it is sufficient to consider only points lie in the strip of width $2d_1$
  - sort all points in strip by y coordinate
  - if point p is in strip with y coordinate y_p, then only points on the other side with y coordinate y_q $|y_p - y_q| < d_1$ need to be considered
  - for each point lies in the strip, at most 6 neighbors need to be considered.



Algorithm Closest_Pair

Input: $p_1 , p_2, \ldots, p_n$

Output: d

Begin

Sort the points according to their x coordinates //O($n$log$n$)

Divide the set into two equal-sized parts

d1=Closest_Pair(Left_Part),

d2=Closest_Pair(Right_Part)

d=min(d1, d2)

Eliminate points that lie farther than d

apart from the separation line //O(n)

Sort the remaining points according to their y coordinates //O($n$log$n$)

Scan the remaining points in the y order and //O(n)

compute the distance of each point to its neighbors

If any of these distance is less than d then Update d

End

Complexity of Algorithm Closest_Pair

- **Complexity: O($n\log^2 n$)**
  - **O($n\log n$): sort according to the x-coordinates, once**
  - **O(n): eliminating points outside strips**
  - **O($n\log n$): sort according to the y-coordinate**
  - **O(n): scan points inside strips & compare each one to its 5 neighbors**
  - **$T(n)=2T(n/2)+O(n\log n)$, $T(2)=1$**

  $\Rightarrow T(n)=O(n\log^2 n)$
- **Improvement: O($n\log n$)**
  - **Embed sorting in each two subsets**
  - **Merge step = merge sort**