CS60064 Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya
Partha Bhowmick

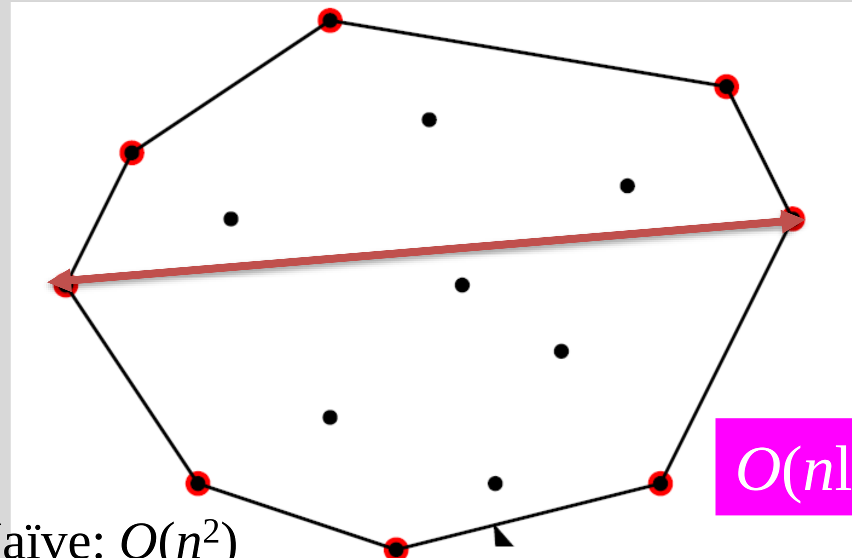
16 February 2022

Lecture #18

Indian Institute of Technology Kharagpur
Computer Science and Engineering

Problem of the day: Find the farthest pair of points

diameter of a point set ($|n|$)



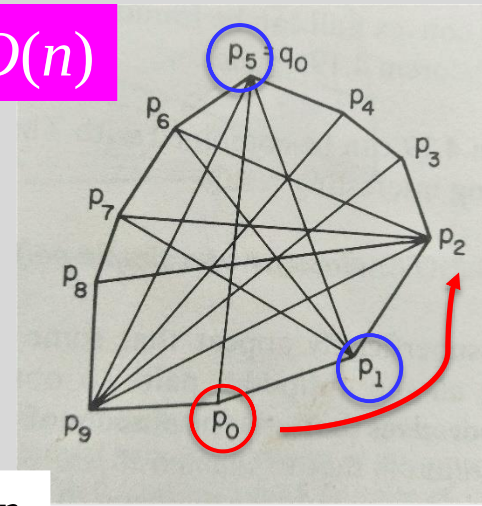
Naïve: $O(n^2)$

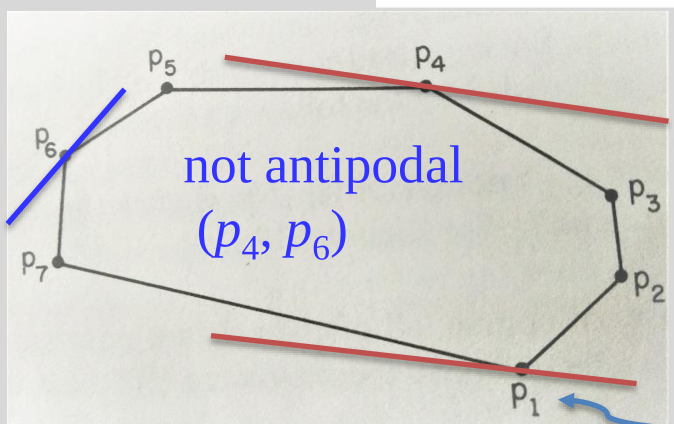
$O(n\log n)$

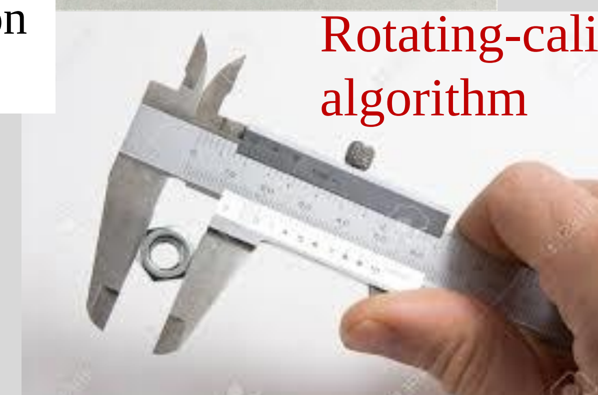
unlike circles, a tangent to a polygon through a vertex is not unique

Claim: Farthest pair on a convex hull must be antipodal points

$O(n)$







Rotating-calipers algorithm

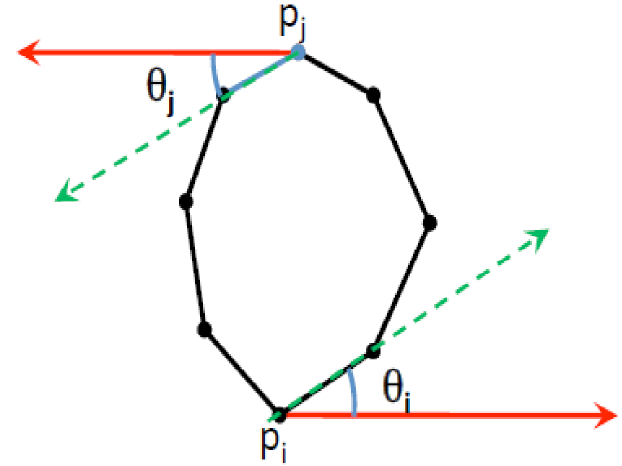
antipodal pairs (p_1, p_4), i.e, they admit parallel tangents (lines of support)

Antipodal Points

Antipodal pairs of a set *S* must be extreme points and therefore must be the vertices of CH(*S*)

Rotating Calipers (Shamos)

- Find the convex hull
- Find an antipodal pair (p_i, p_j) by marching along the hull
- Generate the next antipodal pair:
  - Determine θ_i & θ_j
  - (suppose $\theta_i < \theta_j$) rotate the lines of support by θ_i
  - Output (p_{i+1}, p_j) as the next antipodal pair
- Repeat step 2 until L_1 or L_2 is rotated by 180°



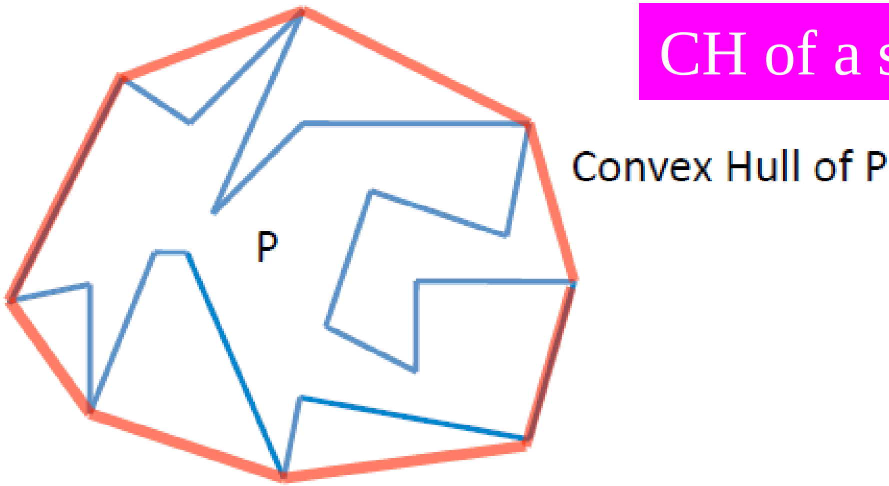
Farthest or nearest pairs between two parallel lines of support can thus be found in *O*(*n*) time

2D Convex Hull: Summary

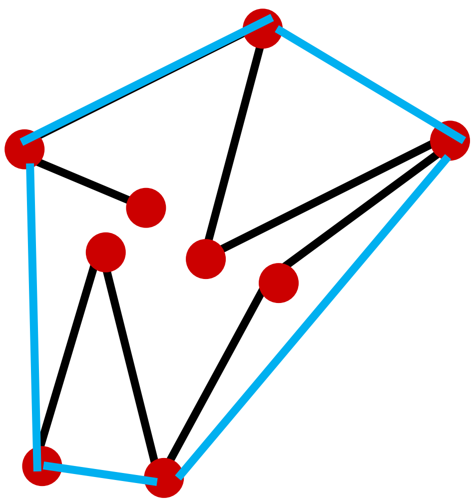
- Brute-force algorithm: $O(n^3)$
- Quick hull: $O(n^2)$
- Jarvis' march (gift wrapping): $O(nh)$; output-sensitive
- Incremental hull: $O(n \log n)$
- Divide-and-conquer: $O(n \log n)$
- Graham's scan: $O(n \log n)$
- Lower bound: $\Omega(n \log n)$
- Chan's algorithm: $O(n \log h)$; output-sensitive;
- Convex hull of a simple polygon/poly-chain with n vertices: ?

Convex Hull of a Simple Polygon

- **Problem:** Given a simple polygon P, determine its convex hull.



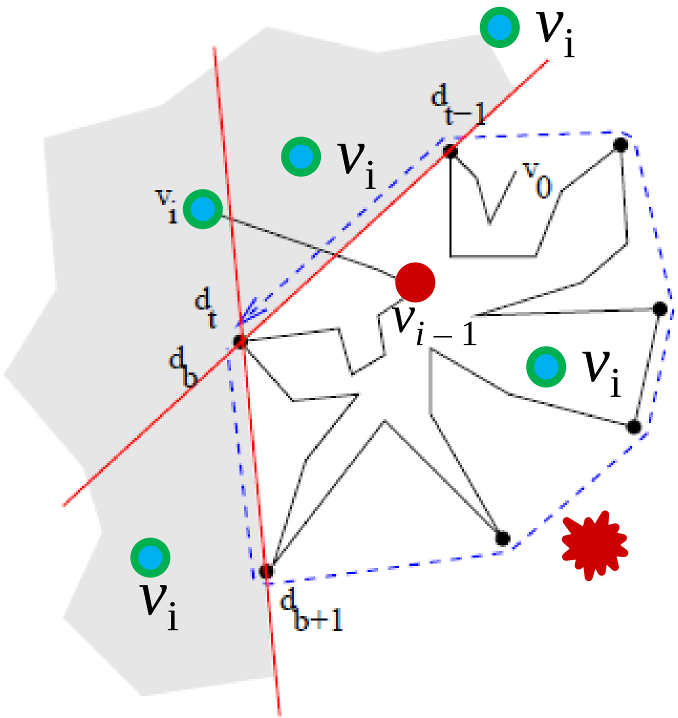
CH of a simple polygonal chain



- O(nlogn) algorithm is easy
- Linear time algorithm is tricky but can be designed.
- **Note:** Ω(nlogn) lower bound does not apply since the points of P are not unorganized points

A. Melkman, On-line Construction of the Convex Hull of a Simple Polyline, *Information Processing Letters*, Vol. 24, pp. 11-12, 1987; $O(n)$-time algorithm

Melkman's Algorithm: Key Ideas



Process the poly-chain/polygon sequentially: $v_0, v_1, \ldots, v_{i-1} \ldots$, and construct incremental hull;

let $d_b = d_t$ = last vertex that is added to the hull;

let the next vertex be v_i; can we draw tangents and proceed as in incremental hull?

partial hull is stored as "deque"

Q1: Where can v_i appear??

Q2: Why deque??

push v: $(t \leftarrow t+1;\ d_t \leftarrow v)$

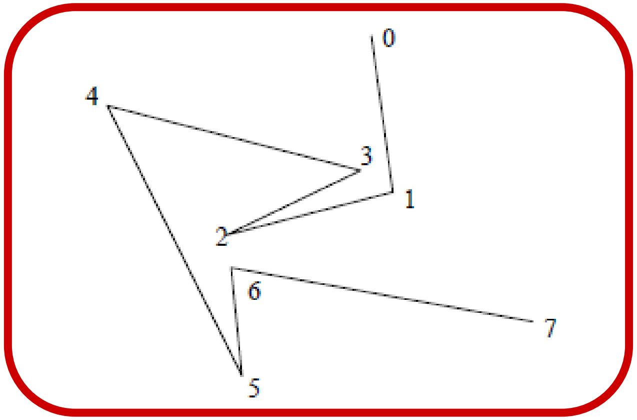
pop d_t : $t \leftarrow t-1$

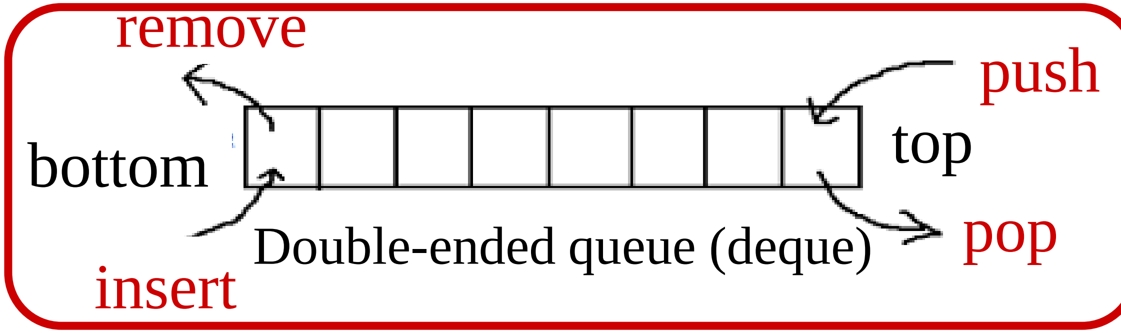
insert v: $(b \leftarrow b-1;\ d_b \leftarrow v)$

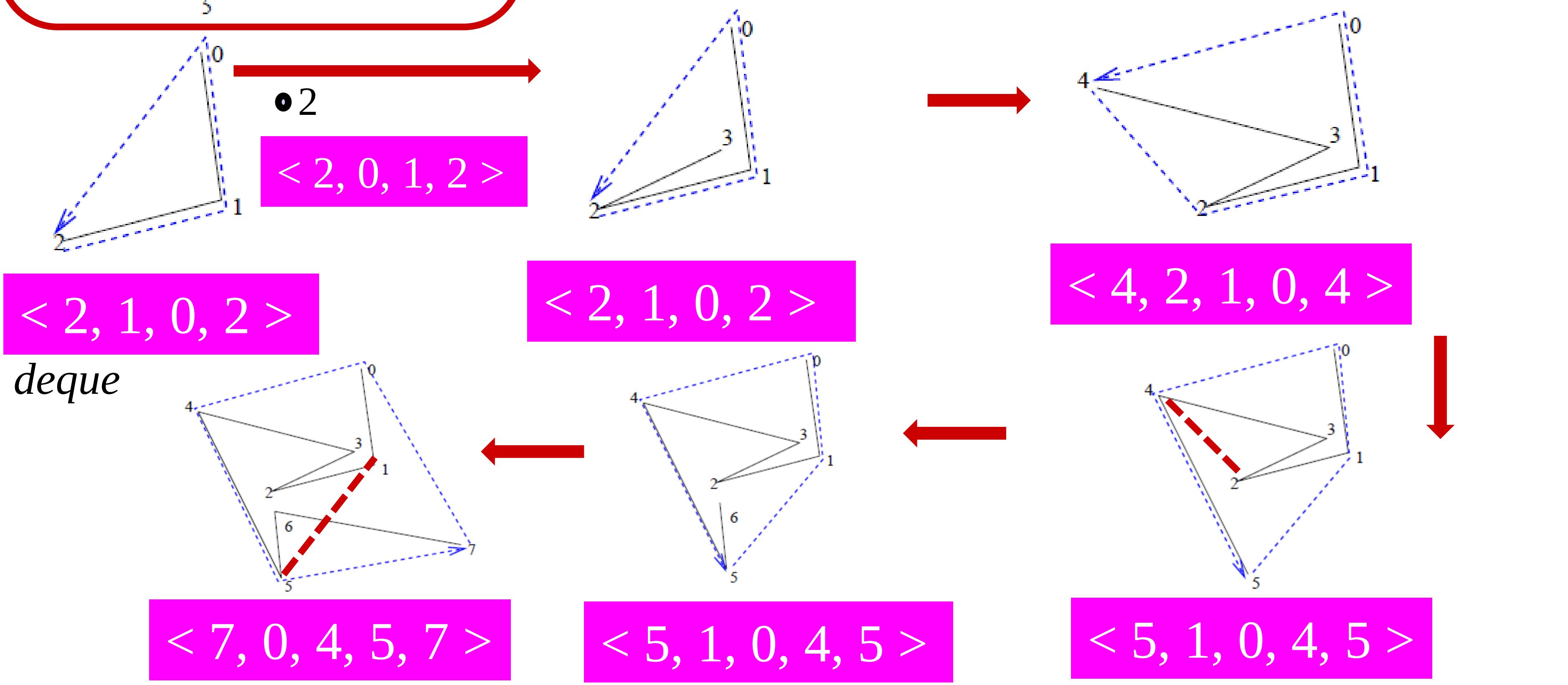
remove d_b : $b \leftarrow b+1$

- If v_i lies in the interior, do nothing;
- Else keep on popping/deleting from the queue based on the test, and update the hull by pushing/inserting v_i when tests flag convexity;
- Repeat until the last vertex is processed

Melkman's Algorithm: Snapshots







Final hull vertices in CCW

Courtesy: Joe Mitchell

Melkman's Algorithm

Let the simple chain be $C = (v_0, v_1, \ldots, v_{n-1})$, with vertices v_i and edges $v_i v_{i+1}$, etc.

(0) Initialize: If Left(v_0, v_1, v_2), then $D \leftarrow \langle v_2, v_0, v_1, v_2 \rangle$; else, $D \leftarrow \langle v_2, v_1, v_0, v_2 \rangle$.

$i \leftarrow 3$

(1) While (Left(d_{t-1}, d_t, v_i) and Left(d_b, d_{b+1}, v_i)) do

$i \leftarrow i + 1$

If $i = n$, Exit

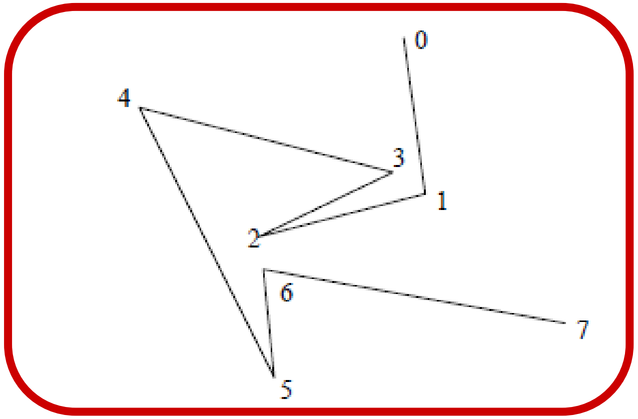
(2) Restore convexity:

Until Left(d_{t-1}, d_t, v_i), do pop d_t. Push v_i.

Until Left(v_i, d_b, d_{b+1}), do remove d_b. Insert v_i.

$i \leftarrow i + 1$; If $i = n$, Exit

Go to (1).



Assume all vertices are in general positions

On termination, the content of the deque read from L → R would provide the CCW-ordering of vertices on the CH; each vertex is processed $O(1)$ times. Hence, time complexity: $O(n)$

CS60064 Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya
Partha Bhowmick
18 February 2022
Lecture #19 & Lecture #20

Indian Institute of Technology Kharagpur
Computer Science and Engineering

Agenda Today

- Problem of the day: Maximum-Containment Problem or MaxRS (Maximum Range-Search Query)
- Convex hull: Chan's Method
- Bi-chromatic non-crossing matching using convex hull
- Proximity problem: Closest-pair of points

Problem of the Day: Maximum-Containment Problem

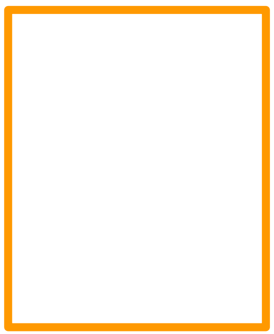
F: A rectangle comprising *n* random points in general positions

P: A smaller rectangle with given aspect ratio

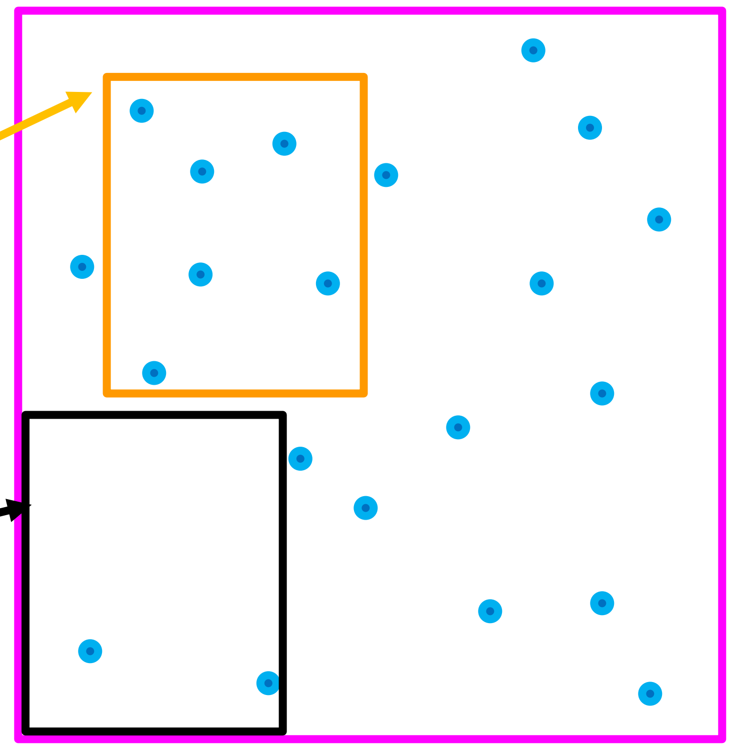
Max-containment: Find an isothetic position of *P* in *F* that encloses the maximum number of points (MaxRS)

Min-containment (MinRS): *P* should be fully contained in *F* and should enclose minimum number of points

P



F



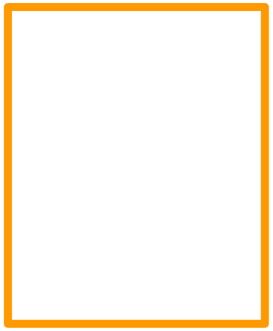
Maximum containment of *P*:
\# points = 6

Assume: only translation of *P* is allowed

Minimum containment of *P*:
\# points = 2

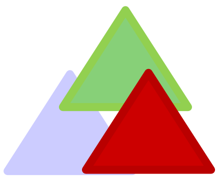
Maximum-Containment Problem (MaxRS)

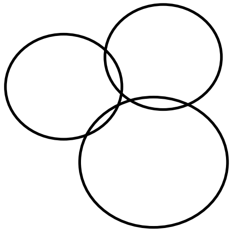
P



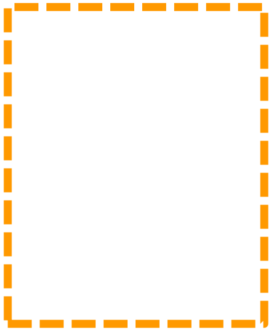
(Max-containment): find an isothetic position of *P* that encloses the maximum number of points in *F*

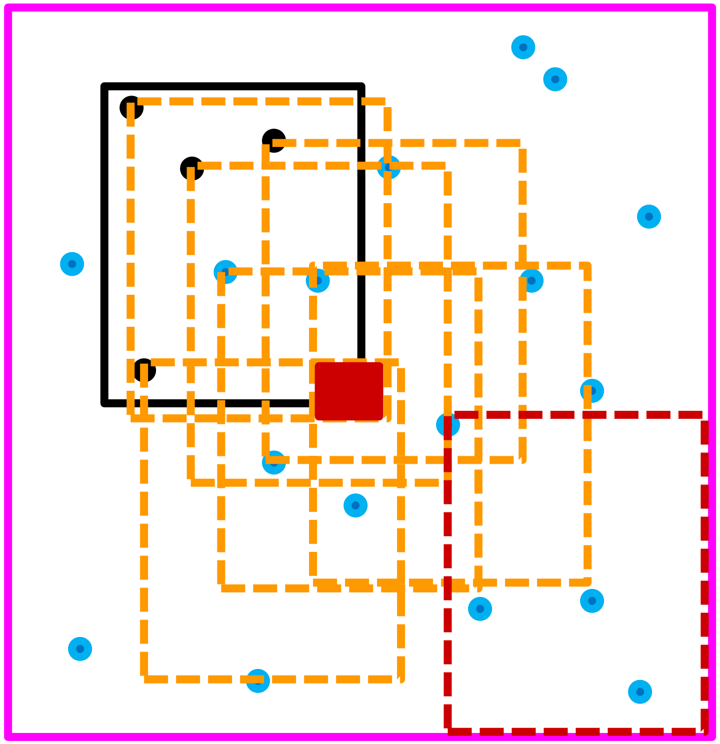
Naïve: $O(n^3)$ time





F

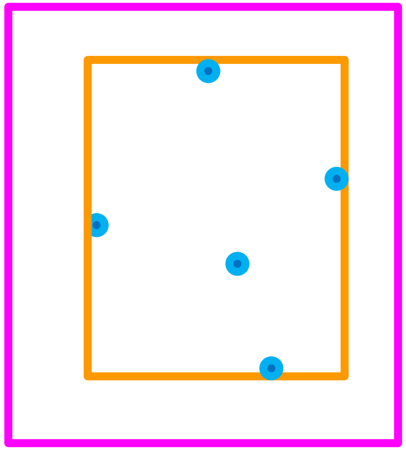




Helly property

How?

Use line-sweep algorithm and determine intersections of rectangles



Max-enclosure: find the maximum clique of the rectangle intersection graph $\Rightarrow$ $O(n \log n)$

Convex Hull: Summary So Far

- Brute force algorithm: $O(n^3)$
- Quick-Hull: $O(n^2)$
- Jarvis' march (gift wrapping): $O(nh)$
- Incremental insertion: $O(n \log n)$
- Divide-and-conquer: $O(n \log n)$
- Graham's scan: $O(n \log n)$
- Lower bound: $\Omega(n \log n)$
- CH for simple polygons/poly-chains: $O(n)$

Longstanding open question:
Is it possible to improve the time complexity of a convex-hull algorithm to $O(n \log h)$, where h: #hull vertices ?

Faster Algorithms (Output-Sensitive)

Kirkpatrick-Seidel (1986): an $O(n \log h)$ worst-case algorithm

n: # vertices; *h:* # vertices on the hull

Chan (1996)*: $O(n \log h)$ algorithm - combines two slower methods (Jarvis March and Graham Scan)

Note:
Jarvis' is good when $h \ll n$;
Graham's is good when $h \approx n$ (i.e., reduces backtracks)

*Timothy Chan, "Output-sensitive results on convex-hulls, extreme points, and related problems", *Discrete & Computational Geometry,* Vol. 16, pp. 369-387, 1996

Timothy Chan's Algorithm (*DCG*, 1996)

Objective: Can we implement CH for an n-set in $O(n \log h)$ time, where h denotes the number vertices on the hull?

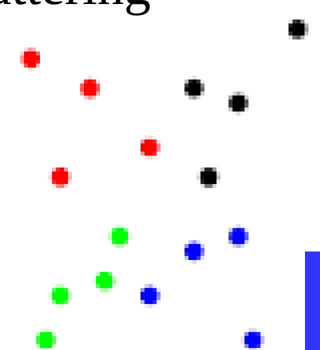
Note: Full sorting cannot be done!

Main Idea:

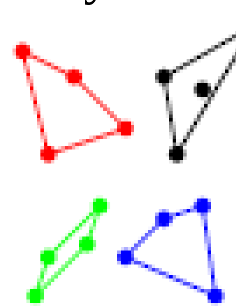
Integrate: Divide-and-Conquer (D&C), Graham Scan, and Jarvis March

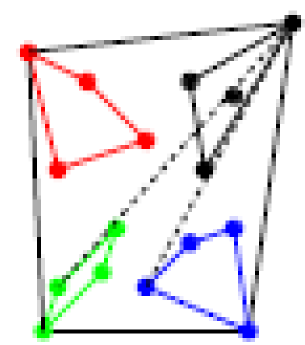
1. Suppose we shatter (D&C) the point-set into $r = \lceil n/m \rceil$ groups each having size "m", $m \approx h$
2. (convex hulls of these groups may not be *disjoint*)
3. Run Graham scan on each group $\Rightarrow$ "log h" factor
4. Wrap these groups using Jarvis March

random shattering



sub-hulls may be intersecting





Problem: we do not know h!

size of each group should not be two small or too large!

Observation: Interior points within the convex hull of each partition cannot appear on the final hull; only a subset of their *hull points* can! Two-fold reduction: Discard interior points; also the hull points that do not satisfy tangency will not appear in Jarvis-M

Timothy Chan's algorithm (1996): Key Ideas

Objective: Can we implement CH for an n-set in $O(n \log h)$ time, where h denotes the number vertices on the hull?

Basic Idea: Partition into smaller groups, and combine Graham scan and Jarvis March

Problem: we need to choose a suitable size for each group! Assume each group has size m; during execution of the algorithm, we will fix m iteratively. Note that $1 \leq m \leq n$; If m is too large or too small, we have no benefit;

Partition the n points into groups of size m; number of groups is $r = \lceil n/m \rceil$.
Compute hull of each group with Graham's scan.
Next, run Jarvis on the groups.

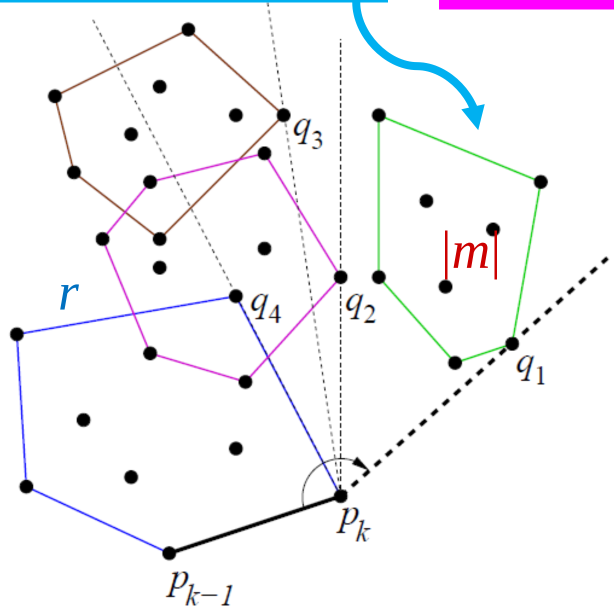
Chan's Algorithm

Partition points P into r-groups of size m, $r = \lceil n/m \rceil$

- Each group takes $O(m \log m)$ time - sort + Graham
- r-groups take $O(r\, m \log m) = O(n \log m) \longrightarrow (1)$

r groups, each of size m

A tangent to a convex m-gon from an external point can be found in $O(\log m)$ by binary search



- Cost of Jarvis on r convex hulls: Each step takes $O(r \log m)$ time; total $O(hr \log m) = ((hn/m) \log m)$ time. $\longrightarrow (2)$

h: number of points on CH(P)

- Thus, total complexity

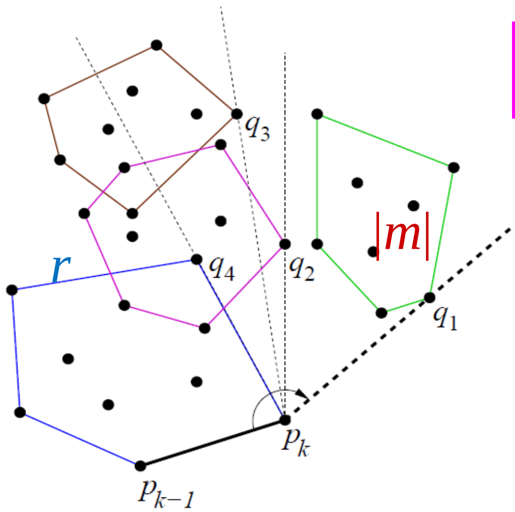
$$(1)\ \&\ (2) \Rightarrow O\left(\left(n + \frac{hn}{m}\right) \log m\right)$$

- If $m = h$, this gives $O(n \log h)$ bound.

Find CH of each group

Unfortunately we do not know h and hence, m. Ideas?

First Part of Chan's algorithm: Partial_Hull (*P*, *m*)



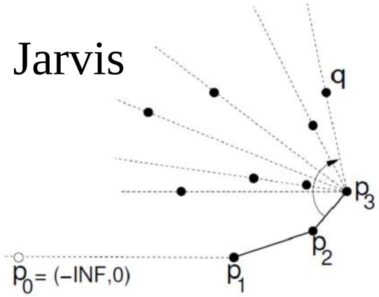
As if *m* is known via Little Birdie!

Partition P into r groups of m each.

Compute Hull(P_i) using Graham scan, $i = 1, 2, \ldots, r$.

$p_0 = (-\infty, 0)$; p_1 bottom point of P.

For $k = 1$ to m do



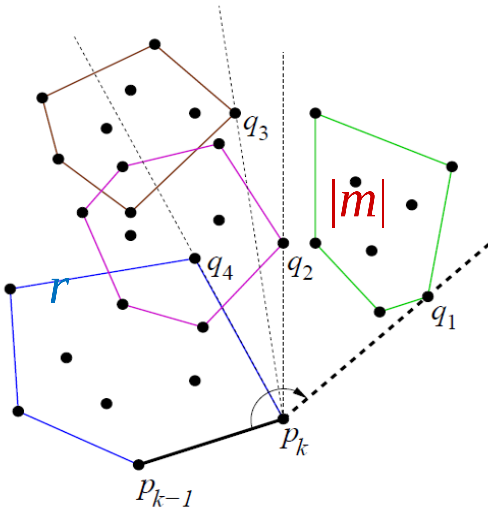
Run Jarvis March on *r* hulls: for *i* = 1 to *r*, do

Draw tangent from p_k to *r*-hulls and choose the one that first hits the sweep-line when rotated CCW with pivot at p_k;

If $p_{k+1} = p_1$, then return the convex hull $\{p_1, p_2, \ldots, p_k\}$ else *m* is too small, "try again" [for success, $m \geq h$]

Fact: $m = 1 \Rightarrow$ the problem reduces to full Jarvis March
$m = n \Rightarrow$ the problem reduces to full Graham scan

Finishing Chan's algorithm: Analysis



How to choose m? use doubly-exponential search

Hull(P)

Partial_Hull(P, m)

- **for** $t = 1, 2, \ldots$ **do**

  1. Let $m = \min(2^{2^t}, n)$.
  2. Run Chan with m, output to L.
  3. If $L \neq$ "try again" then return L.

Referring to Exp. (2): Jarvis-M is being wrapped m times $\Rightarrow$ thus for each value of t, the time including Graham's and Jarvis' becomes $O(n \log m)$

Iteration t takes time $O(n \log 2^{2^t}) = O(n2^t)$.

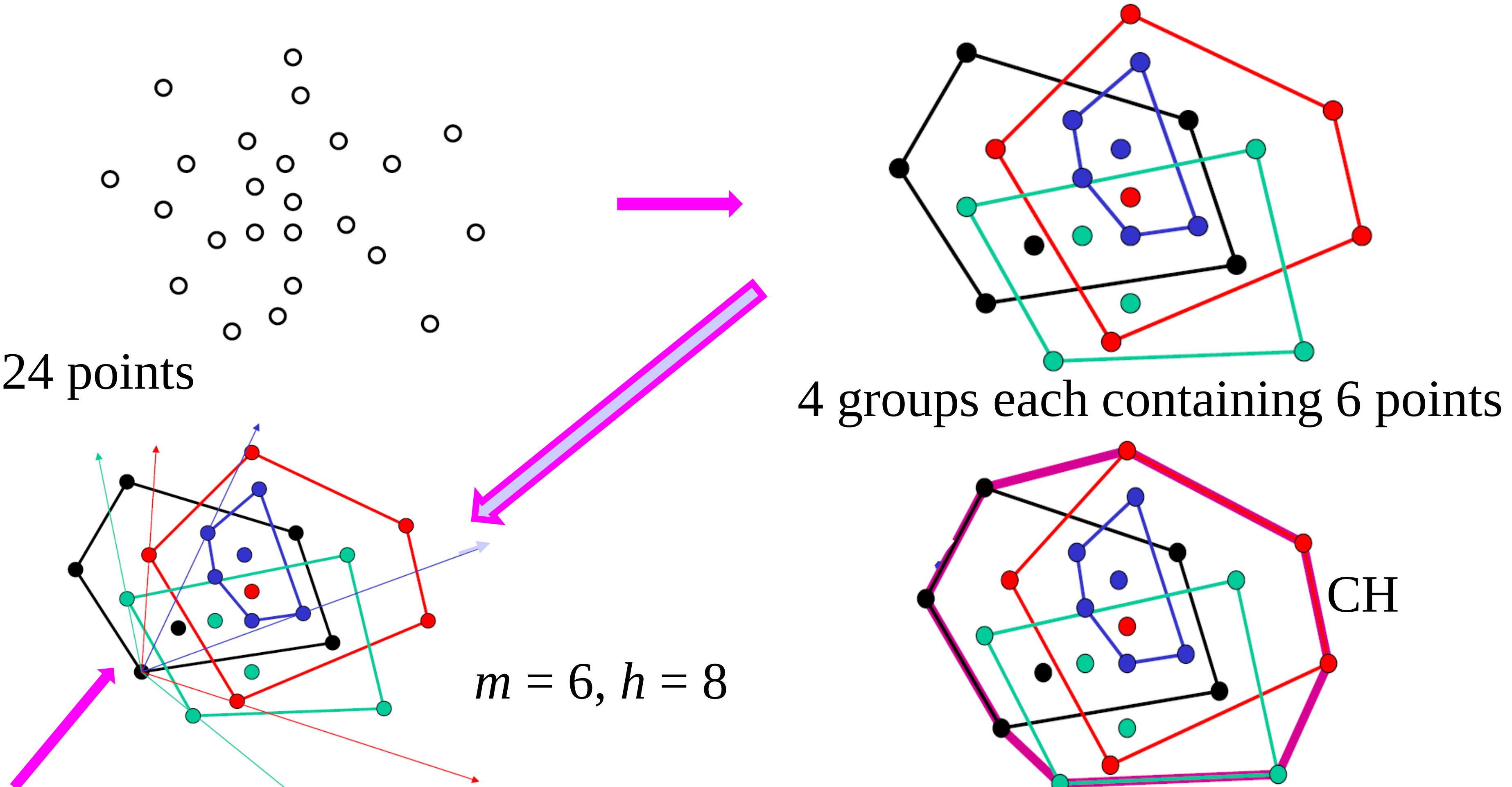
Max value of $t = \lceil \lg \lg h \rceil$, since we succeed as soon as $2^{2^t} \geq h$. [Note that for success, $m \geq h$]

Hence, the running time (ignoring constant factors) = $\sum_{t=1}^{\lg \lg h} n2^t = n \sum_{t=1}^{\lg \lg h} 2^t \leq n2^{1+\lg \lg h} = 2n \lg h$

$\Rightarrow O(n \log h)$ □

Ref: David Mount's Lecture Notes

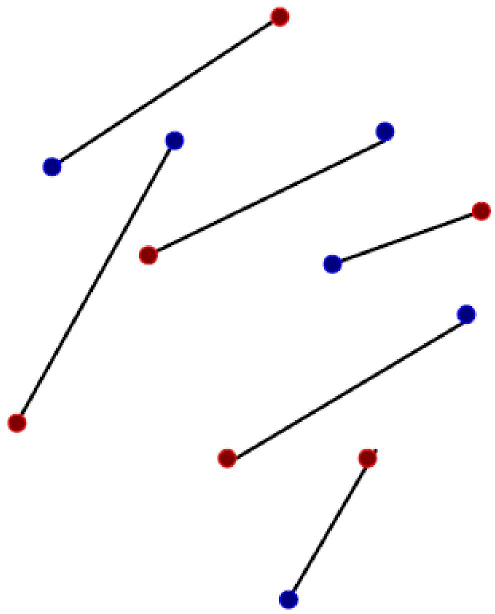
Chan's Algorithm: Example

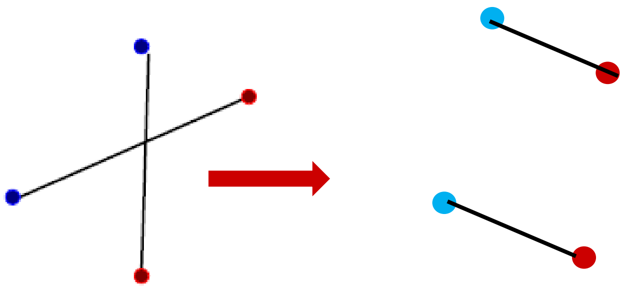


One iteration of gift wrapping: find tangents to convex hulls and take the one with the least slope; since $m = 6$, continue wrapping six times; however, Jarvis-M will not take you to the start vertex in six steps. More wraps are needed. Instead, m should be incremented, Graham's and Jarvis' be redone so as to construct the hull in $m \geq h$ steps and to achieve the intended complexity results.

Non-Crossing Matching

Given *n* red and *n* blue points in the plane (no three collinear), compute a red-blue non-crossing matching



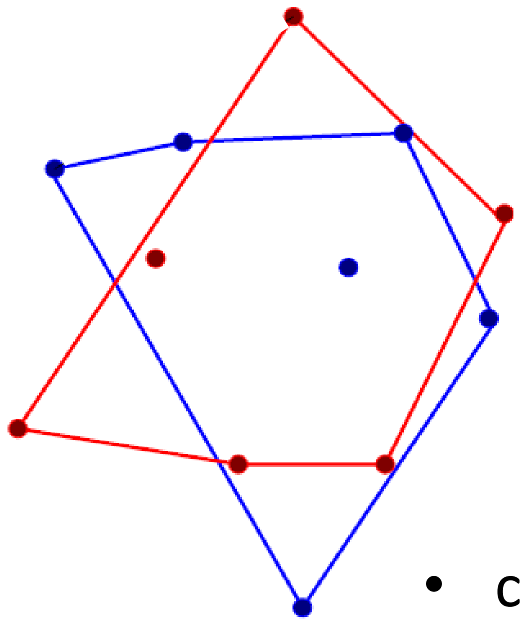


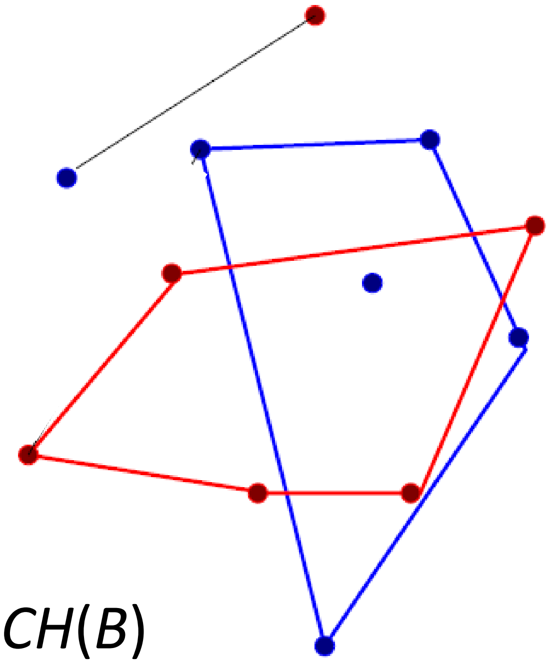
reduces matching cost

A non-crossing matching always exists;
Matching with minimum total length must be non-crossing

Non-Crossing Matching

Given n red and n blue points in the plane (no three collinear), compute a red-blue non-crossing matching



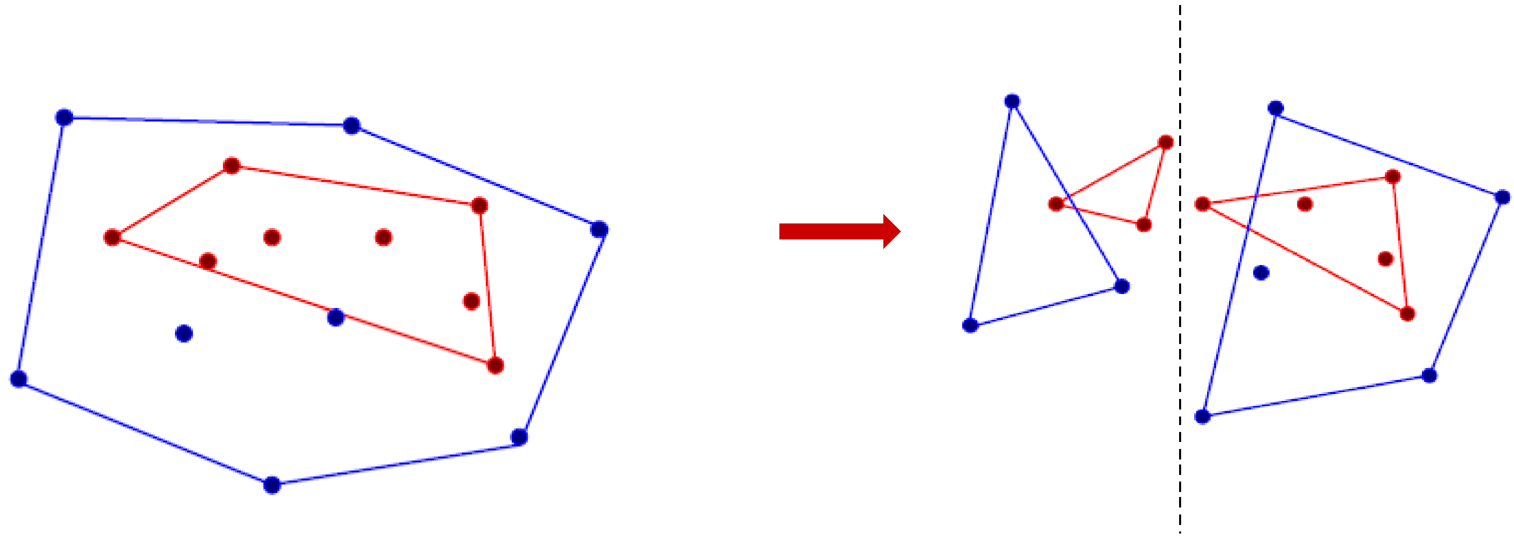


- compute $CH(R)$ and $CH(B)$
- find a common tangent, say, rb
- output (r, b) as a matching edge
- remove points r, b
- update convex hulls and iterate

Courtesy:
Subhash Suri, UCSB

Non-Crossing Matching

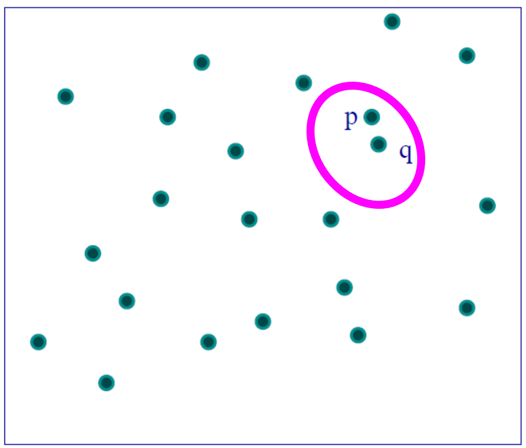
What if Red, Blue CHs are nested? No tangents can be drawn!



- Split by a vertical line, creating two smaller, intersecting or disjoint hull problems; select tangents and iterate as before
- [Hershberger-Suri ’92] gives optimal $O(n \log n)$ solution

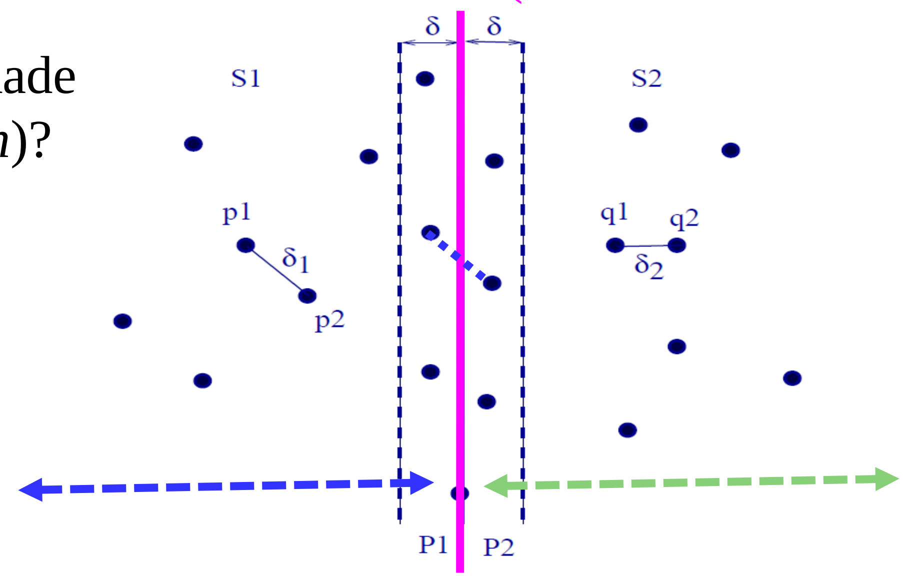
Proximity Problem: Closest Pair of Points

Given n points on the plane determine the closest pair



Can it be made in $O(n \log n)$?

Naïve: $O(n^2)$ time



1D problem is easy; sort and check

2D problem: use Divide-and-Conquer

- We partition S into S_1, S_2 by vertical line ℓ defined by median x-coordinate in S.
- Recursively compute closest pair distances δ_1 and δ_2. Set $\delta = \min(\delta_1, \delta_2)$.
- Now compute the closest pair with one point each in S_1 and S_2.

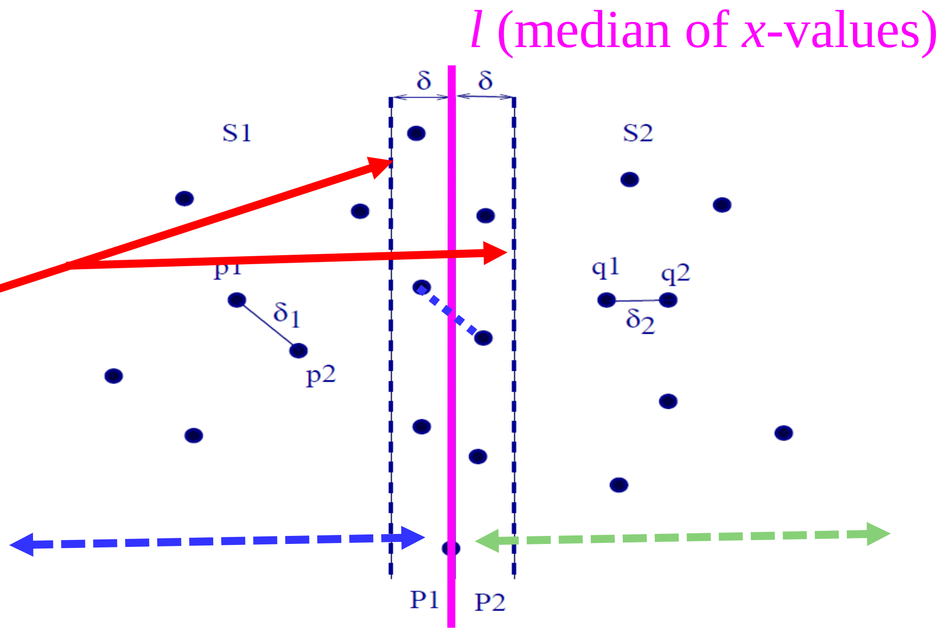
Courtesy:
Subhash Suri, UCSB

Closest Pair of Points

Given *n* points on the plane determine the closest pair

$$\delta = \min(\delta_1, \delta_2)$$

interval {(*l* - δ), (*l* + δ)}



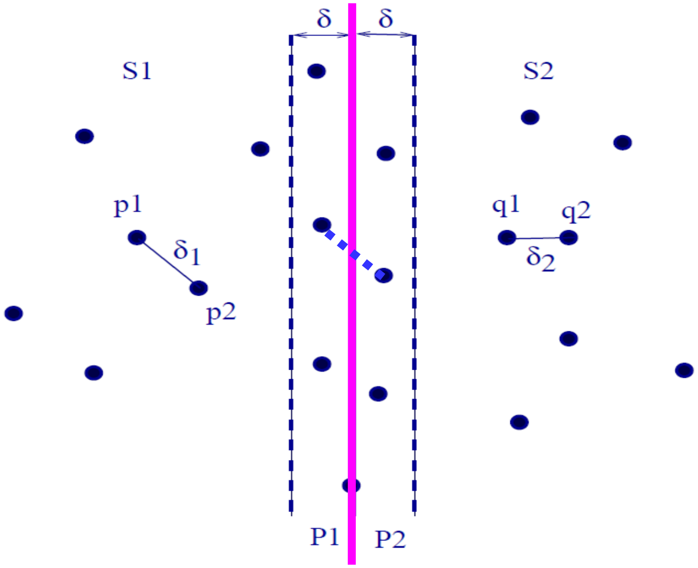
In each candidate pair (p, q), where $p \in S_1$ and $q \in S_2$, the points p, q must both lie within δ of ℓ.

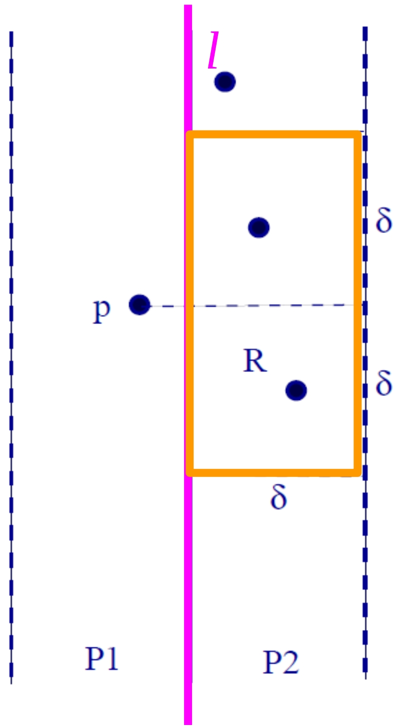
Is it **possible that all $n/2$ points of S_1 (and S_2) lie within δ of ℓ** ? Complexity still remains $O(n^2)$!

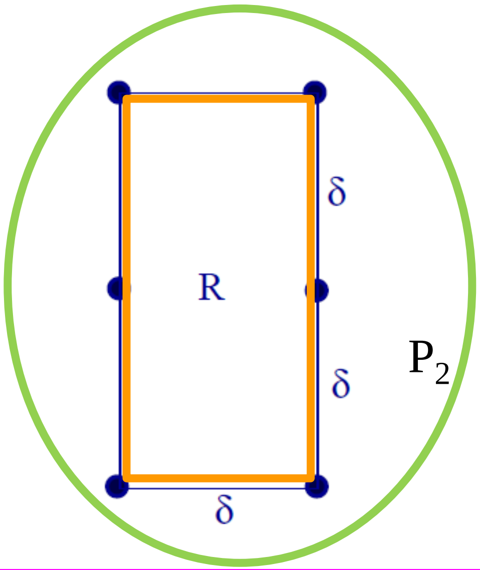
Closest Pair of Points

R can contain at most six points!
$\Rightarrow$ # Overall tests $\leq 6 \times n/2$ distance comparisons

$$\delta = \min(\delta_1, \delta_2)$$







Use some properties from discrete geometry: packing

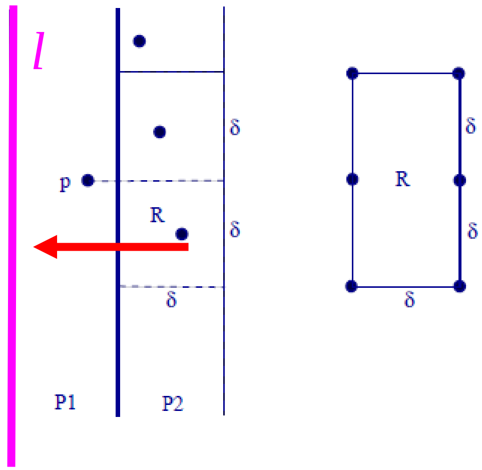
Consider a point $p \in S_1$. All points of S_2 within distance δ of p must lie in a $\delta \times 2\delta$ rectangle R.

How many points can be inside R if each pair is at least δ apart?

Courtesy: Subhash Suri

Closest-Pair Problem: Analysis

- In order to determine at most 6 potential mates of p, project p and all points of P_2 onto line ℓ.



Ref: Preparata and Shamos, Computational Geometry

Analogous to merge sort

- Pick out points whose projection is within δ of p; at most six.
- We can do this for all p, by walking sorted lists of P_1 and P_2, in total $O(n)$ time.
- The sorted lists for P_1, P_2 can be obtained from pre-sorting of S_1, S_2.
- Final recurrence is $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

Announcement of Online Test - 01

Friday, 25 February, 2022; 11:05 am - 12:50 pm

Coverage: Polygons, point-inclusion queries, orientation test, robustness issues, polygonization, diagonals, triangulation, convex partition, art-gallery problems, DCEL, intersections, map overlay, convex hull, related algorithms, data structures, and complexities (material covered until 18 February 2022);

Questions will be made available through Moodle and answers should also be submitted via Moodle. The submission server will open at 10:55 am and close at 1:05 pm, 25 February 2021.

Credit: 25%