Algorithmic Paradigms

- Incremental method
- Divide-and-conquer
- Randomization
- Space partitioning/bucketing (hashing)
- Data structure augmentation
- Space sweep
- Locus method
- Geometric transform (duality)

new

1

Design Methodology

1. Understand the geometry of the *general case*, ignoring "degenerate" cases. Identify useful primitives (both data and functionality)
2. Design algorithm for the general case
3. Extend the algorithm to handle degenerate cases
4. Provide a *robust* implementation of your algorithm, including required primitives and predicates

- A set of geometric objects is in ***general position*** (i.e., generic position) if it avoids troublesome or degenerate configurations, such as three collinear points, points with the same x-coordinate, four cocircular points, etc.

2

Incremental Approach

Given a set $G = \{g_1, \ldots, g_n\}$ of geometric objects, let $G_i = \{g_1, \ldots, g_i\}$ and let A_i denote the solution to instance G_i

1. Compute A_c, the solution to G_c, for small constant c
2. **for** $i \leftarrow c + 1$ **to** n **do**
   Compute A_i from g_i and A_{i-1}

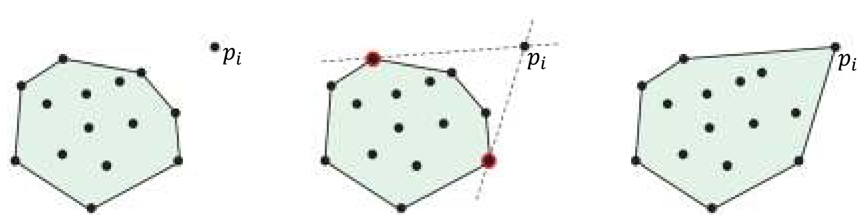
For the convex hull of a set P of points:

1. Let $H_3 \coloneqq \text{conv}(P_3)$, i.e., the triangle $p_1p_2p_3$
2. **for** $i \leftarrow 4$ **to** n **do**
   $H_i \leftarrow \text{conv}(H_{i-1} \cup \{p_i\})$

General position assumption: no three points are collinear

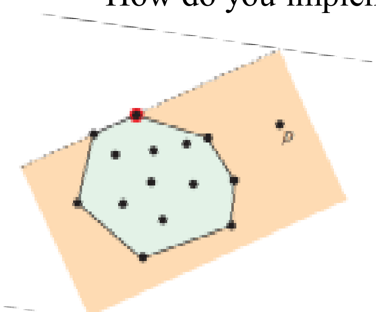
Polygon Tangents

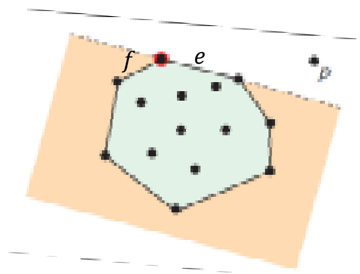
- Let P be a convex polygon and q a point on the boundary of P. A line ℓ through q ***supports*** P at q if all of P lies on the same side of P. Line ℓ is a ***tangent*** to P at q and q is a ***tangency point***
- When processing p_i, in an incremental step, our algorithm will need to find two tangency points in H_{i-1} which admit tangent lines through p_i



Polygon Tangents…

- Let P be a convex polygon given in CCW order and p a point exterior to P
- Each edge of P is either visible to p or invisible to p
- Let q be a vertex with incident edges e and f. Then q is a tangency point if exactly one of e and f is visible
  - How do you implement this efficiently and robustly?





Convex Hull: Algorithm 2

1) $H \leftarrow \text{conv}(p_1, p_2, p_3)$

2) **for** $i \leftarrow 4$ **to** n **do** {Assume $H = \langle q_1, \ldots, q_{h_{i-1}} \rangle$}

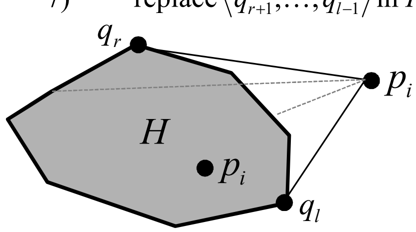
3) **if** $p_i \notin H$ **then**

4) **for** $j \leftarrow 1$ **to** n_{h-1} **do** {find tangency points}

5) **if** $\text{turn}(p_i, q_j, q_{j-1}) = \text{turn}(p_i, q_j, q_{j+1})$ **then**

6) q_j is a tangency point {q_l for right, q_r for left}

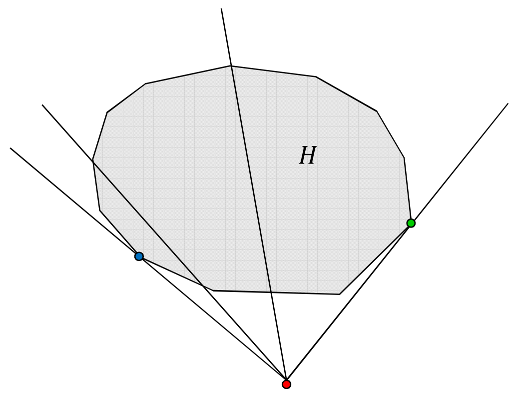
7) replace $\langle q_{r+1}, \ldots, q_{l-1} \rangle$ in H by p_i



Time: $\Theta(n^2)$

Exercise

- In Algorithm 2, find each tangency point in $O(\log m)$ time, where m is the size of H

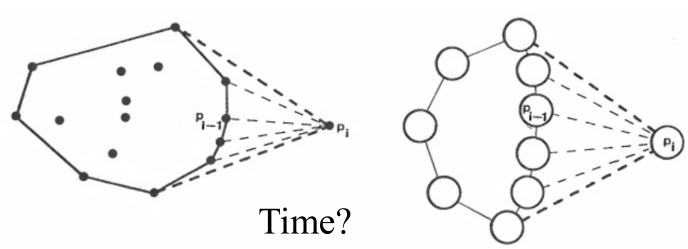


- *Hint*: use binary or exponential search!

7

Improving Algorithm 2

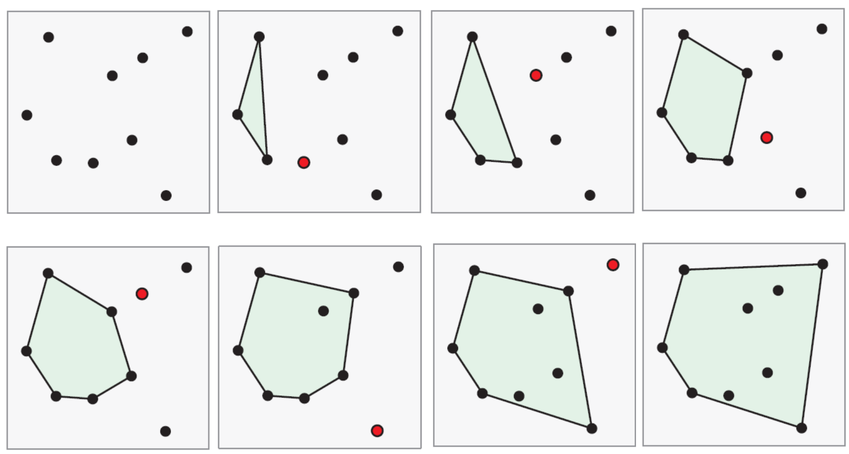
- Can simplify the code by making sure the new point p_i is always outside H_{i-1}. How?
  - Sort the input points by x-coordinate
- p_{i-1} is always visible from p_i
- Walk CCW (resp. CW) from p_{i-1} until right (resp. left) tangent is found, eliminate interior points



Time?

8

Example

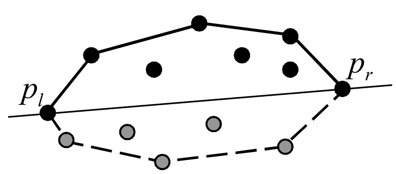


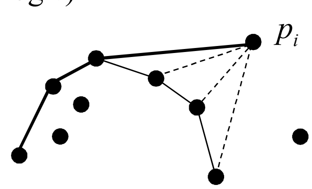
9

Graham Scan: Algorithm 3

Algorithm UpperHull(P)
1) Find the left and right extremes p_l and p_r
2) Find the points $P_U \subset P$ above support($p_l p_r$)
3) Sort P_U by x-coordinate resulting in $\langle p_1, \ldots, p_m \rangle$
4) $L = \langle p_1, p_2 \rangle$
5) **for** $i \leftarrow 3$ **to** m **do**
6) append p_i to L
7) **while** $|L| > 2$ **and** turn of last 3 points $\neq$ right **do**
8) delete the middle of last 3 points from L

Time: $O(n \log n)$

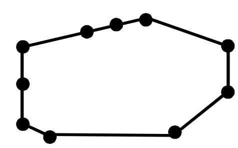




10

Convex Hull: Algorithm 3 Degeneracies and Robustness

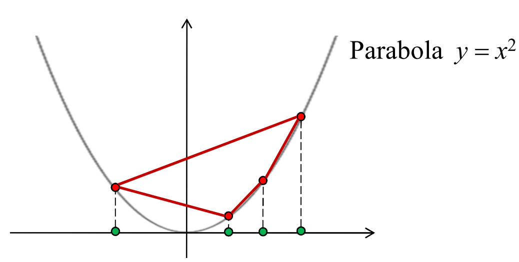
- Several points with the same x-coordinate?
  Sort lexicographically
- Three or more points lie on a straight line?
  Test returns "no turn"
- Rounding errors in floating point arithmetic?
  Algorithm computes a closed polygonal chain



11

Convex Hull: A Lower Bound

- Sorting requires $\Omega(n \log n)$ time
- Sorting can be done in $O(n + f(n))$ time where $f(n)$ is the time to compute convex hull
- Convex hull requires $\Omega(n \log n)$ time



12

Divide-and-Conquer (DAC)

Given a problem of size *n*:

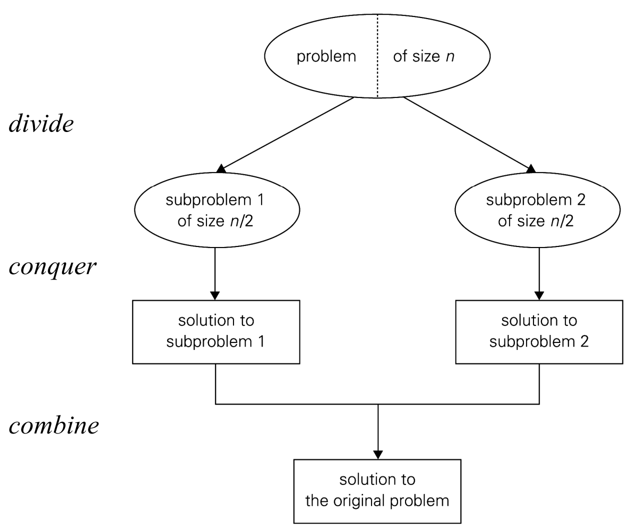
1) *Divide* the problem into *k* sub-problems of size *n/k* each
2) *Conquer* by solving each sub-problem independently
3) *Combine* the *k* solutions to sub-problems into a solution to the original problem

Time: $$T(n) = \sum_{i=1}^{k} T(n_i) + d(n) + c(n)$$

$$T(n) = k \cdot T(n/k) + d(n) + c(n)$$

13

DAC With 2 Sub-problems



$$T(n) = 2T(n/2) + d(n) + c(n)$$

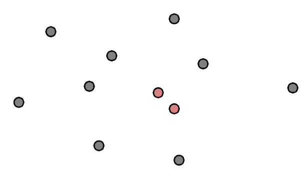
14

Example: Closest Pair

- Given a set $S = \{p_1, \ldots, p_n\}$ of points in the plane, find two closest, i.e., a and b such that

$$\text{dist}(p_a, p_b) \le \text{dist}(p_i, p_j), \forall 1 \le i, j \le n$$

- Brute force takes $\Theta(n^2)$ time
- Can we do better?



A Lower Bound

- *Element Uniqueness* : given a set $\{x_1, \ldots, x_n\}$ of numbers, determine if there are duplicates, i.e., find $i \neq j$ such that $x_i = x_j$
- *Element Uniqueness* requires $\Omega(n \log n)$ time
- Use same transformation as in convex hull

$$x_i = x_j \Leftrightarrow \text{dist}((x_i, x_i^2), (x_j, x_j^2)) = 0$$

- *Closest Pair* is at least as hard as *Element Uniqueness* $\Rightarrow T(n) = \Omega(n \log n)$

A Divide and Conquer Solution

To compute $CP(S)$:

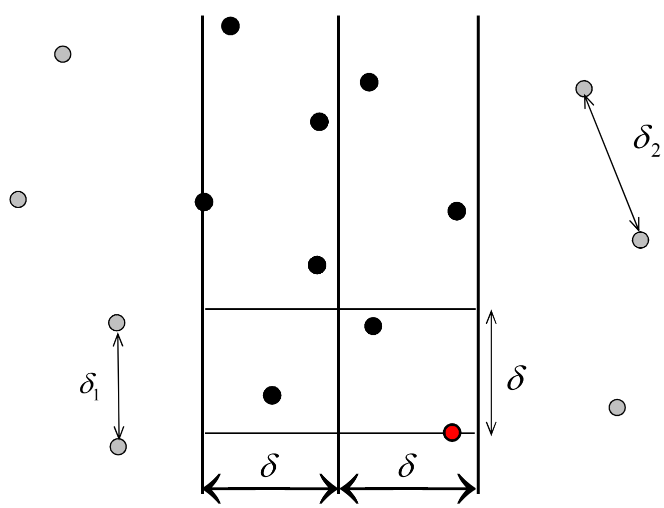
1. Sort S lexicographically, i.e., such that

$$x_i < x_{i+1} \vee x_i = x_{i+1} \wedge y_i \le y_{i+1}$$

2. Divide: $S_1 = \{p_1, \ldots, p_{n/2}\}$ and $S_2 = \{p_{n/2+1}, \ldots, p_n\}$
3. Conquer: let $\delta_1 = CP(S_1)$ and $\delta_2 = CP(S_2)$
4. Combine: how? is $\delta = \min(\delta_1, \delta_2)$ the answer?

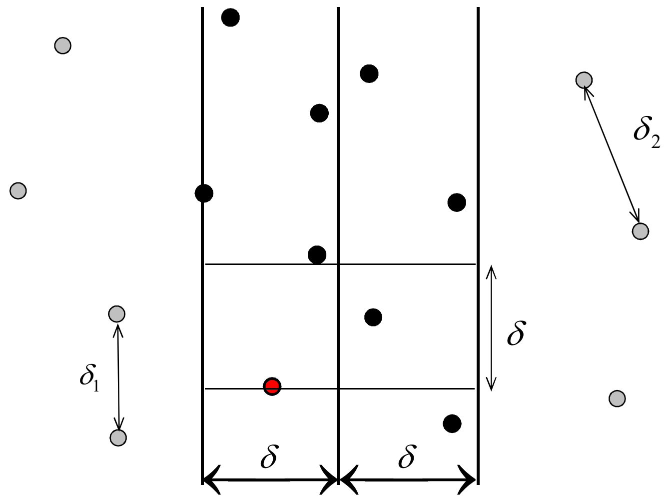
17

Closest Pair: Combine Step



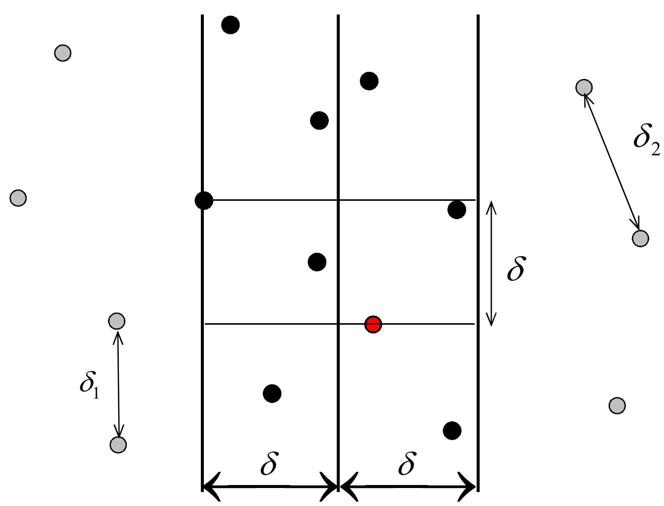
18

Closest Pair: Combine Step



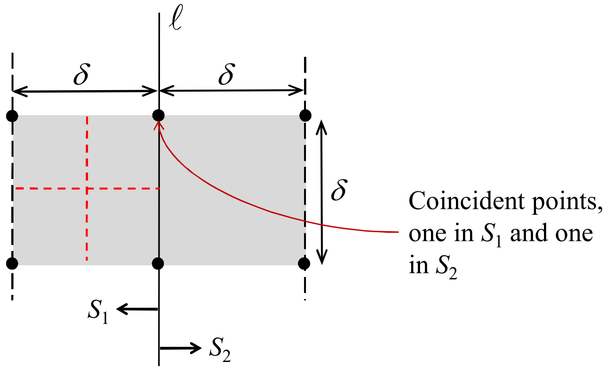
19

Closest Pair: Combine Step



20

How many points can you have in the box?



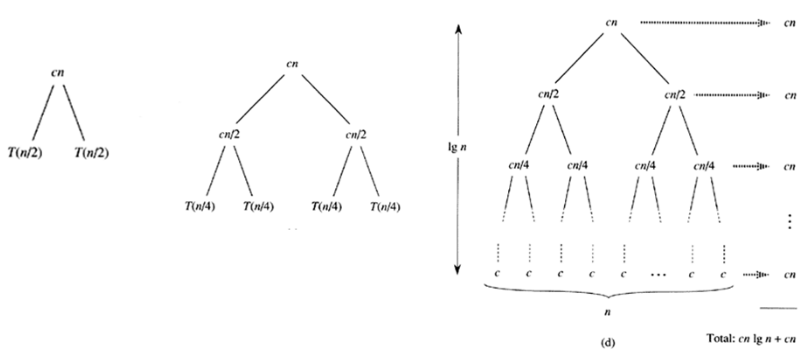
- Examine the points within δ of ℓ by ascending y-coordinate and consider for each those that are up to δ units above

21

Analysis

- Presorting by x- and y-coordinates: $O(n \log n)$
- Divide-and-conquer proper:

  $T(n) = 2T(n/2) + cn = \Theta(n \log n)$



22

3D Case

- Can the algorithm be generalized to higher dimensions?
- The combine step derives its efficiency from a *sparsity* condition:
  - A set S of points in R^d is *sparse* if there are positive δ and c such that every hypercube of side length δ contains at most c points
- Does sparsity hold in 3D? Does it help? Is it even needed?

23

DAC for Convex Hull

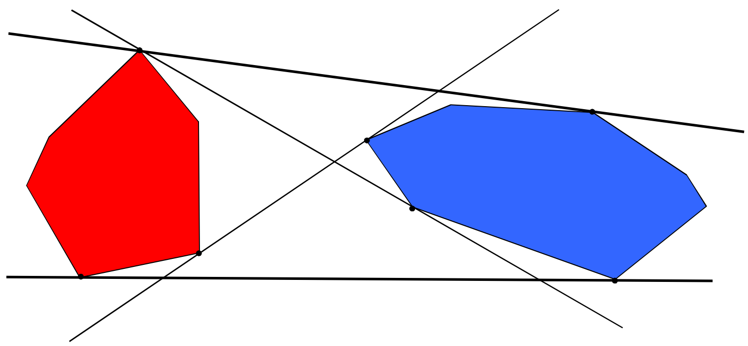
- Given a set $S = \{p_1,\ldots,p_n\}$ of points on the plane find conv(S)

1. Sort S lexicographically
2. Divide: $S_1=\{p_1,\ldots,p_{n/2}\}$ and $S_2=\{p_{n/2+1},\ldots,p_n\}$
3. Conquer: let $P_1=$ conv(S_1) and $P_2=$ conv(S_2)
4. Combine: $P =$ conv(P_1,P_2), but how?

24

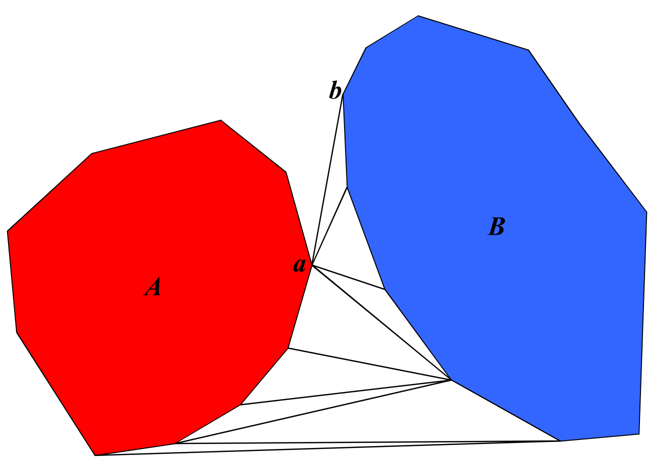
Convex Hull: Combine Step

- There are four tangents
- Need to find *upper* and *lower* tangents



25

Finding the Lower Tangent



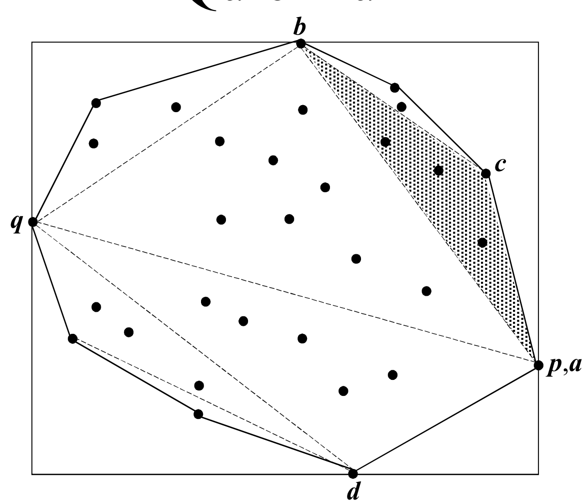
26

Finding the Lower Tangent

$a \leftarrow$ rightmost point of left polygon A
$b \leftarrow$ leftmost point of right polygon B
while $T = ab$ not lower tangent to A and B **do**
    while T not lower tangent to B **do**
        $b \leftarrow b + 1$ // move CCW
    while T not lower tangent to A **do**
        $a \leftarrow a - 1$ // move CW

27

Quickhull



- Start by finding two *extreme* points of S, e.g., $\boldsymbol{p}$ and $\boldsymbol{q}$
- Generate subproblems on each side of $\boldsymbol{pq}$

28

Quickhull

QuickHull(a, b, S)

1. if S is empty return $\langle\ \rangle$
2. $c \leftarrow$ point of S farthest from ab
3. $A \leftarrow$ points of S strictly to the right of ac
4. $B \leftarrow$ points of S strictly to the right of cb
5. **return** Quickhull(a, c, A) + $\langle c\rangle$ + Quickhull(c, b, B)

conv(S) = $\langle p\rangle$ + Quickhull(p, q, P) + $\langle q\rangle$ + Quickhull(q, p, Q)
where

P = points of S above pq, Q = points of S below pq, and
'+' means concatenation

29

Output-Sensitive Algorithms

- Running time is a function of both *input size* (n) and *output size* (h)

  Example: fixed distance neighbors ran in $O(n + h)$ time
- Potentially very fast convex hull algorithm when h is small
- Simplest approach is *gift wrapping*:
  - Let a string hang from the lowest input point and "wrap it around" the set
  - Finds the extreme points in order

30

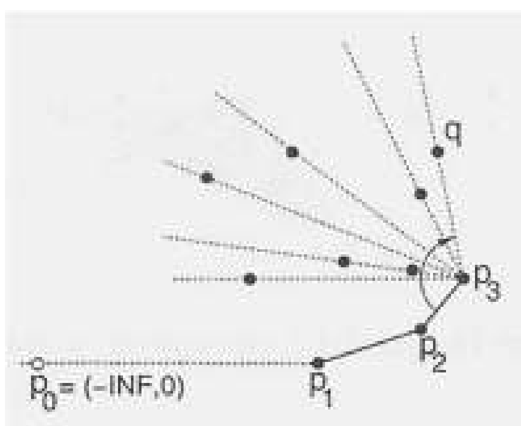
Jarvis March

- A greedy algorithm
- Find the next edge of conv(S) by brute force:

  If p_{k-1} and p_k are the last two vertices added to the hull, then p_{k+1} is the point $q \in S$ that maximizes the angle $\angle p_{k-1} p_k q$
- Takes linear time per vertex of conv(S)
- Initial edge is $p_0 p_1$ where $p_0 = (-\infty, y_1)$, and $p_1 = (x_1, y_1)$ is the lowest point of S
- Runs in O(nh) time, where h is the output size (good when h is small!)

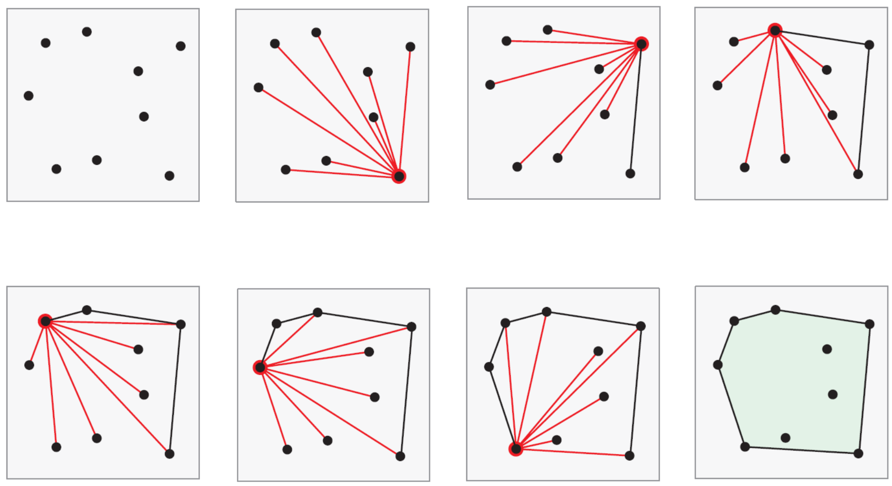
31

Jarvis March…



32

Example



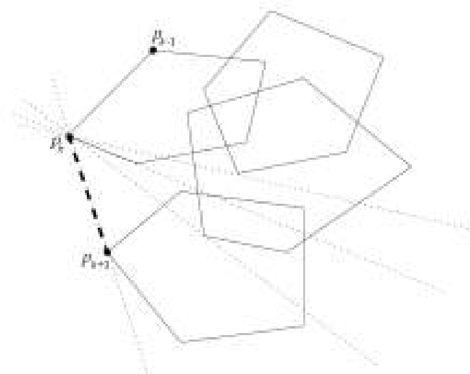
33

Chan's Algorithm (1996)

- Runs in $O(n \log h)$ time
- Combines two algorithms: Graham scan (or any other optimal algorithm) and Jarvis march
- Wrapping can be performed faster if we preprocess the points
  - Partition points into subsets and use the convex hull of each subset

34

Computing the Next Edge

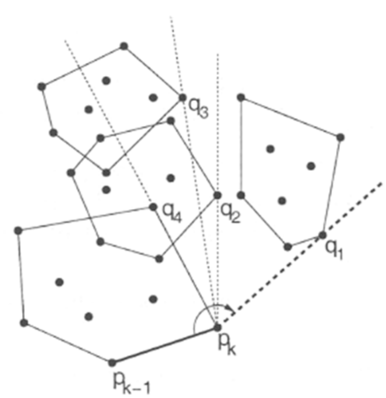


- The supporting line of each edge of conv(S) is tangent to the convex hull of one of the subsets
- There are fewer subsets than points!

35

General Idea

- Choose parameter $1 \leq m \leq n$
- Partition S into $r = \lceil n / m \rceil$ subsets of size m each
- Compute the convex hull of each subset independently using an optimal algorithm
- Find the global convex hull by performing a Jarvis march on the polygons



36

Hull Check

HullCheck(S, m, H)

```
Partition S into disjoint subsets S_1,…,S_r of size m
for i ← 1 to r do
    Find conv(S_i) using Graham scan
Find the lowest point p_1 of S and let p_0 = (−∞,y_1)
for k ← 1 to H do
    for i ← 1 to r do
        find q_i in S_i that maximizes ∠p_{k−1}p_kq_i
    p_{k+1} ← q_j, where q_j maximizes ∠p_{k−1}p_kq_j
    if p_{k+1} = p_1 then return ⟨p_1,…,p_k⟩
return 'incomplete'
```

Runs in $O(n \log m + H(n/m)\log m) = O(n(1 + H/m)\log m)$

37

Properties of Hull Check

- Returns conv(S) when $H \geq h$
- Runs in $O(n(1 + H/m)\log m)$
- What happens if we choose $m = H$?

  Algorithm runs in $O(n \log H)$
- Need a scheme to "guess" h using few attempts

38

Full Algorithm

1. Set $t \leftarrow 1$
2. **repeat**
3. $\quad m \leftarrow H \leftarrow \min\{n,\ 2^{2^t}\}$
4. $\quad L \leftarrow$ HullCheck(S, m, H)
5. $\quad t \leftarrow t + 1$
6. **until** $L \neq$ '*incomplete*'
7. **return** L

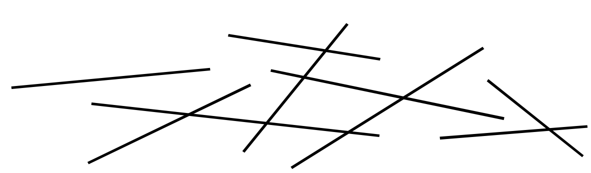
$$m = 4,\ 16, 256, \ldots, 2^{2^t} > h$$

$$\sum_{t=1}^{\log\log h} n \log H = \sum_{t=1}^{\log\log h} n 2^t = n \sum_{t=1}^{\log\log h} 2^t < n 2^{1+\log\log h} = 2n \log h$$

39

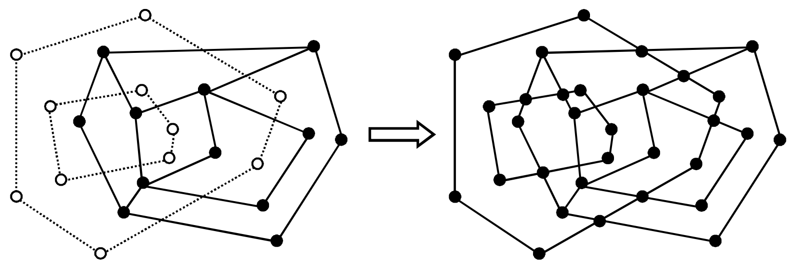
Problem: Segment Intersection

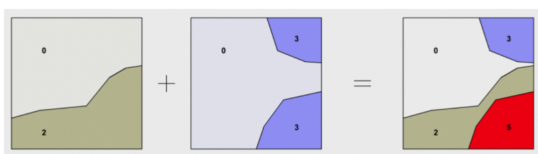
- Given a set S of closed segments report all pairs of segments that intersect
- Many applications: architectural databases, map overlay in GIS

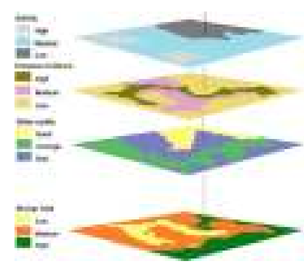


40

Map Overlaying







41

Motivation: Line Sweep

- Given n integers, in no particular order, how do you find out if they are all distinct?
- One possibility is two sort and then “sweep” the real line from smallest to largest, stopping at each integer and asking if it is equal to its predecessor
- The stopping points are the only points of interest or *events* on the real line, i.e., the places where you can gather useful information
- The sorting step collects the events in an *schedule*

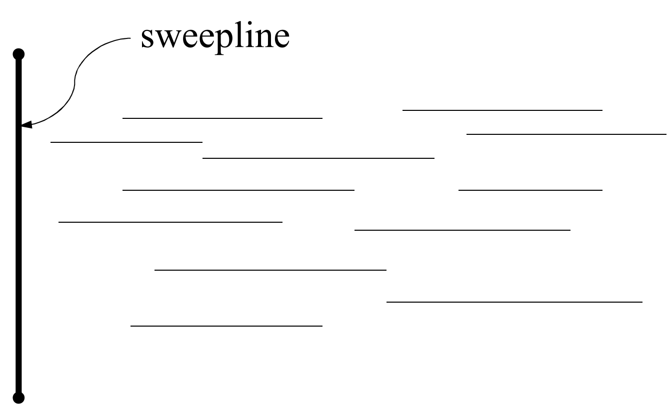
42

Plane Sweep

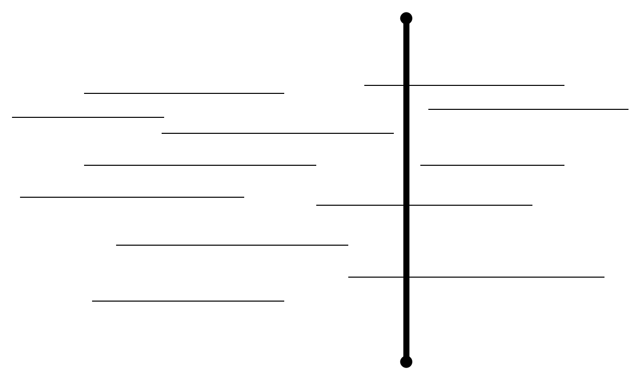
- Sweep a line ℓ over the plane keeping track of objects from S intersected by ℓ as it moves
- The *active* objects are stored in a *status* data structure
- The status changes at locations called *events*
- Two data structures:
  – *Schedule*: ordered sequence of events
  – *Status*: subset of S currently intersected by ℓ

43

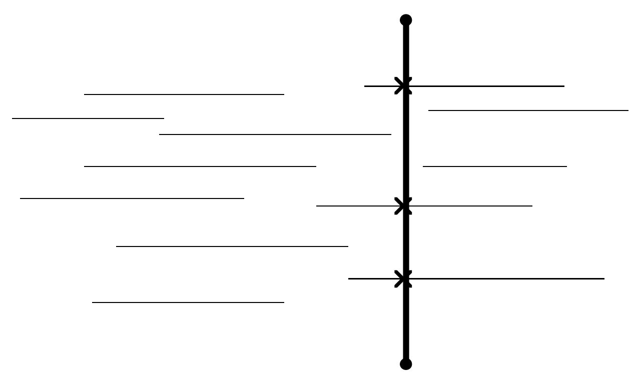
Sweepline Approach



Sweepline Approach

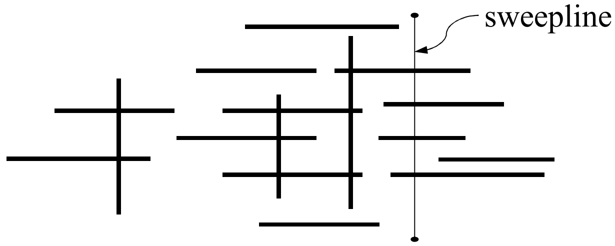


Sweepline Approach



Orthogonal Segment Intersection

- All segments are horizontal or vertical
- Status *T* stores *active* horizontal segments
- Status changes at segment endpoints only
- Vertical segments used for reporting



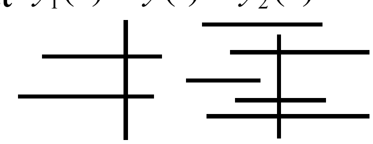
47

Updating the Status

Types of events:

- Left endpoint of horizontal segment ***s***: store ***s*** into *T* using $y(s)$
- Right endpoint of horizontal segment ***s***: delete ***s*** from *T*
- Vertical segment ***s***: report all horizontal segments ***t*** such that $y_1(s) \le y(t) \le y_2(s)$

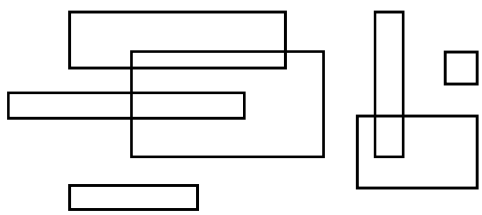
Time: $O(n \log n + k)$



48

Variant: Rectangle Intersection

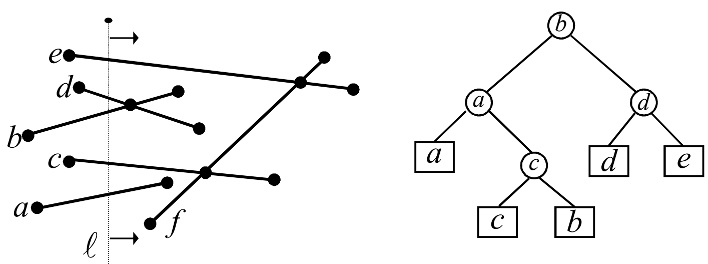
- Report all pairs of rectangles that intersect
- Events: left or right sides of rectangles
- Status stores (y-span) of active rectangles
- Problem reduced to interval intersection search



49

Variant: Segment Intersection

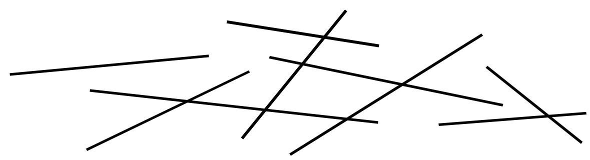
- Report all pairs of segments that intersect
- Status stores active segments sorted by y-coordinate
- Events: left or right endpoints of segments plus all intersection points



50

Segments in General Position

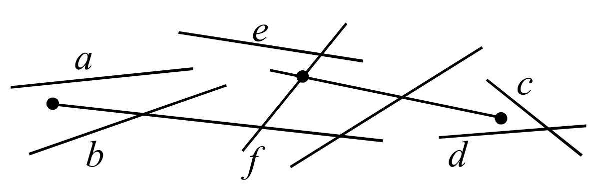
- A set of segments is in general position if:
  - No two endpoints with same x-coordinate
  - Any two segments intersect in at most one point
  - No three segments intersect in a common point
- Intersecting segments change relative order at the intersection point
- *Key insight*: Intersecting segments must be neighbors in the status prior to the intersection point



51

Event Handling (General Position)

- Left endpoint of s: test s for intersection against its two new neighbors
- Right endpoint of s: delete s and test for intersection the former two neighbors of s
- Intersection point: “swap” intersecting segments and test each against former neighbor of other
- Time: $O((n+k)\log n)$



52

Segment Intersection Algorithm

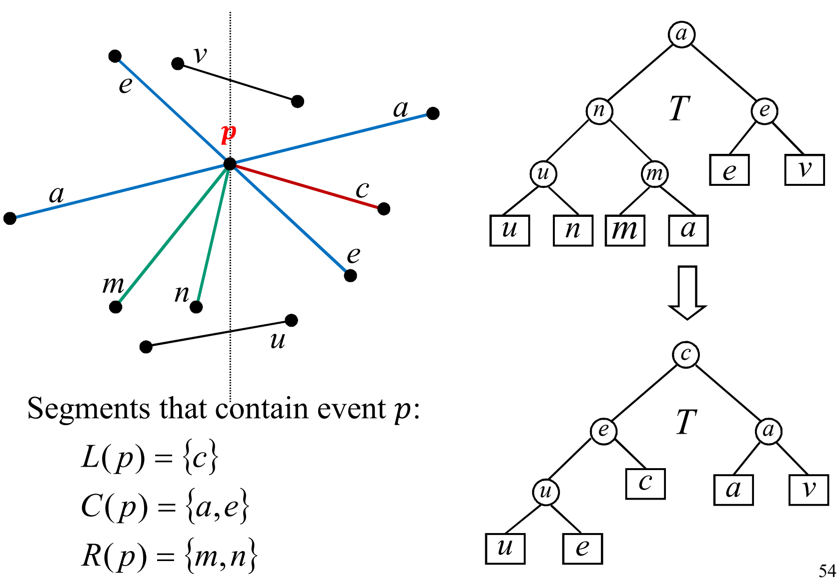
Input: A set S of segments in the plane

Output: all intersection points with segments involved

1. Create event queue Q with segment endpoints (left endpoints include a list of corresponding segments)
2. Initialize an empty status structure T
3. **while** $Q \neq \emptyset$ **do**
4. Extract from Q the next event p
5. Handle event point p

53

Handling Degenerate Cases



Segments that contain event p:

$L(p) = \{c\}$

$C(p) = \{a, e\}$

$R(p) = \{m, n\}$

54

Handling an Event p

1. Search T for set $S(p)$ of segments containing p
2. Partition $S(p)=C(p)\cup R(p)$
3. **if** $|C(p) \cup R(p) \cup L(p)| > 1$ **then**
   report p as well as $C(p) \cup R(p) \cup L(p)$
4. delete $R(p) \cup C(p)$ and insert $L(p) \cup C(p)$ in T
5. **if** $C(p) \cup L(p) = \emptyset$ **then**
   find event of new neighbors s and q of p
   else
   find events of new neighbors of $L(p) \cup C(p)$

- How about vertical segments?

55

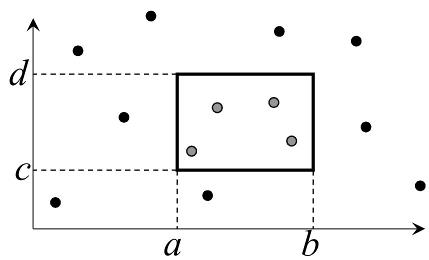
Locus Method

- Pre-compute the answer to *all* possible queries and store results to facilitate look ups
- Each query is mapped to a point in a query space
- The query space is partitioned into regions (loci) within which the answer does not vary.
- Store partition in a data structure $D(S)$

56

Example: Range Searching

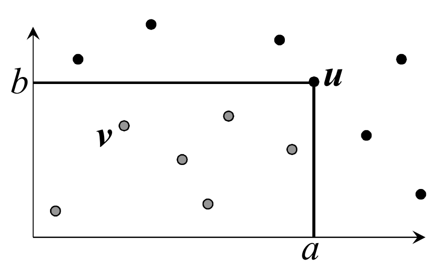
Given a set of points $S = \{p_1, \ldots, p_n\}$ on the plane determine how many lie inside a given upright rectangle ***Q*(*a,b,c,d*)** (a rectangle with sides parallel to the coordinate axes)



57

A Simpler Variant: Dominance

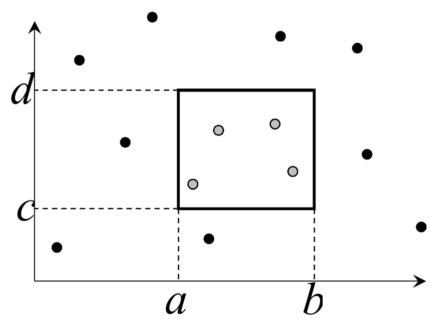
- A point ***u*** dominates ***v*** iff $u_i \geq v_i, 1 \leq i \leq d$
- Report $N(a,b)$, the number of points in S dominated by (a, b)



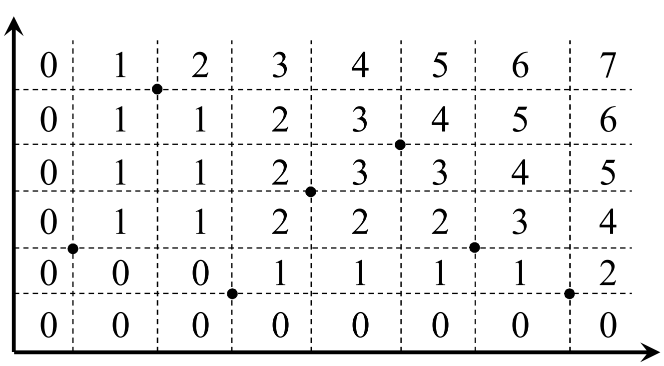
58

Range Searching

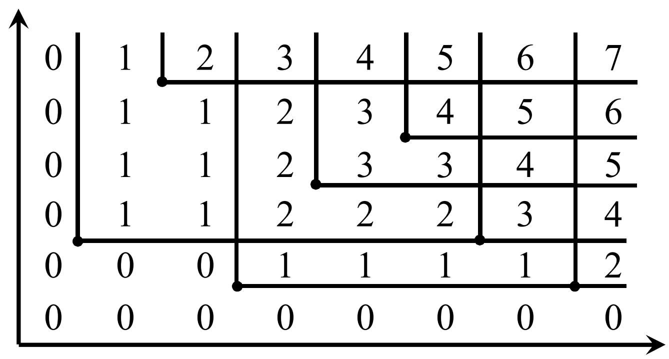
$$Q(a,b,c,d) = N(b,d) - N(a,d) - N(b,c) + N(a,c)$$



A Locus Approach to Dominance



A Locus Approach to Dominance



61

Range Searching: Complexity

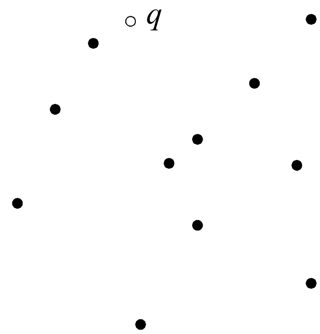
Method	**T(n)**	**M(n)**	**P(n)**
One shot	$O(n)$	$O(n)$	-
Locus	$O(\log n)$	$O(n^2)$	$O(n^2)$
Range tree	$O(\log^2 n)$	$O(n \log n)$	$O(n \log n)$

62

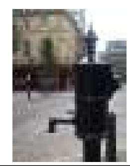
Example: Nearest Neighbor

Given a set P of n points in the plane:

- Store P in a data structure $D(P)$
- Given a query point q, use $D(P)$ to find the point in P that is closest to q

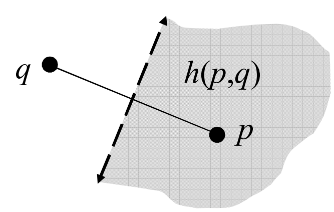


Example: London Cholera outbreak (1854). Given a patient q, which water pump is closest to q's home



Notation

- For points p and q the perpendicular bisector of segment pq splits the plane into two half-planes. The open halfplane that contains p is denoted by $h(p,q)$
- The points in $h(p,q)$ are strictly closer to p than to q



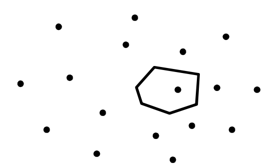
64

Voronoi Diagram

- Vor(P) is a subdivision of the plane into n cells $V(p_1),\ldots,V(p_n)$, one for each site in P
- The Voronoi cell $V(p_i)$ is the locus of points closer to p_i than to any other site in P:

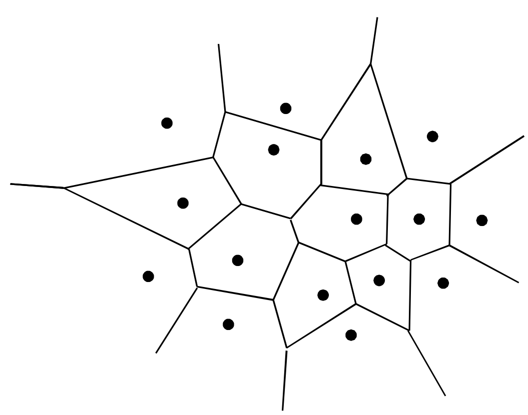
$$q \in V(p_i) \Leftrightarrow \text{dist}(q, p_i) < \text{dist}(q, p_j), \forall p_j \neq p_i$$

- $V(p_i) = \bigcap_{i \neq j} h(p_i, p_j)$



65

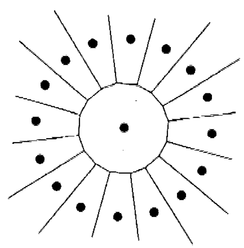
Voronoi Diagram: Example

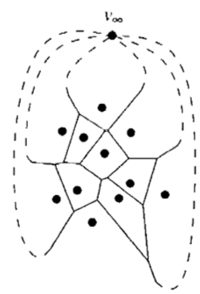


66

Voronoi Diagram: Complexity

Theorem. The number of vertices in the Voronoi diagram of n points in the plane is at most $2n-5$ and the number of edges is at most $3n-6$.





67

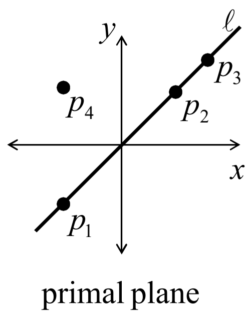
Geometric Transform (Duality)

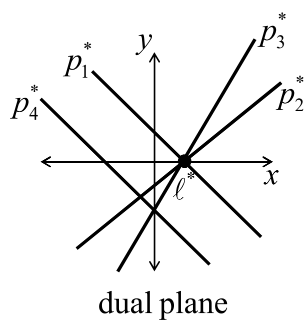
- Lines and points may each be specified by two numbers: m and b or x and y, respectively
  - What if we transform one to the other while preserving some spatial relations in the process?
- Consider the mapping $\ell\colon y = mx - b \leftrightarrow p\colon (m, b)$
- Denote by $D(\ell) = \ell^\star = p$ and $D(p) = p^\star = \ell$
- One-to-one mapping between all non-vertical lines and all points in the plane

68

Duality Properties

- Involution: $p^{\star\star} = (p^\star)^\star = p$ and $\ell^{\star\star} = \ell$
- Incidence preserving: $p \in \ell$ iff $\ell^\star \in p^\star$
- Order preserving: p is above ℓ iff $\ell^\star$ is above $p^\star$

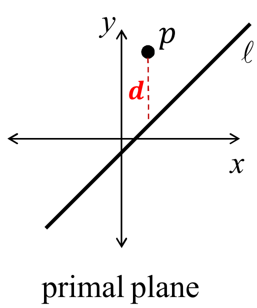


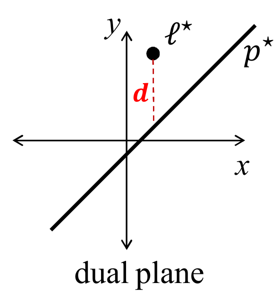


69

Duality Properties

- Distance preserving: the vertical distance between p and ℓ is the same as that between $\ell^\star$ and $p^\star$



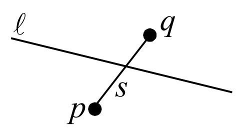


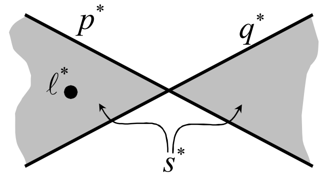
Exercise. What can you say about the distance between parallel lines $\ell: y = ax + b$ and $g: y = ax + c$?

70

Segment Duals

- The dual of a segment s from p to q is the union of the duals of points on s
- The dual of s is a *double wedge*, bounded by the duals of p and q



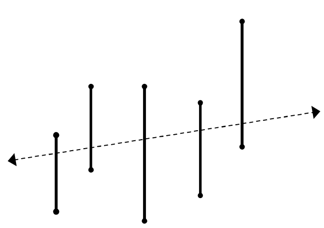


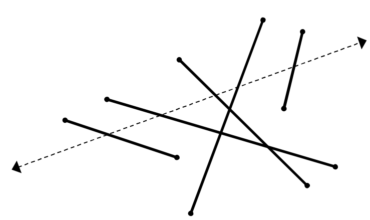
71

Example: Line Stabbing

Given a set of segments $S = \{s_1, \ldots, s_n\}$:

- Find a transversal (a line ℓ that stabs all segments of S)
- Construct a representation of all transversals of S
- Determine if a query line ℓ is a transversal





72

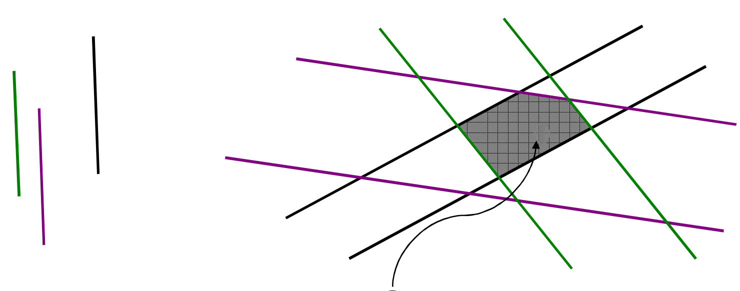
Stabbing Regions

- $R(S) = \bigcap_{s \in S} D(s)$ is the stabbing region of S
- A line ℓ is a transversal of S iff $D(\ell) \in R(S)$

73

Vertical Segments

- For a set S of vertical segments, $R(S)$ is a convex polygon with at most $2n$ edges.

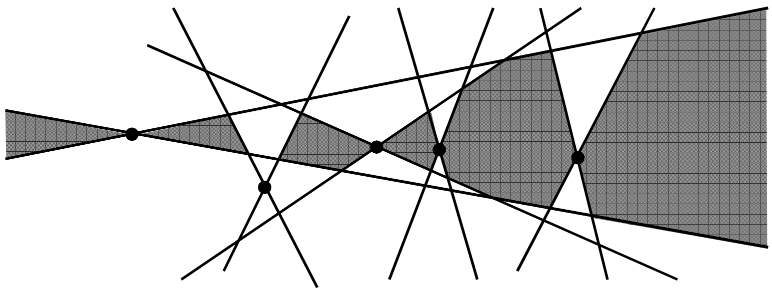


$R(S) = \bigcap_{s \in S} D(s)$

74

Arbitrary Segments

- For arbitrary segments, $R(S)$ is the union of at most $n+1$ convex polygons such that any two of them intersect in at most one point and there is a vertical line that separates them.



75

Combinatorial Analysis

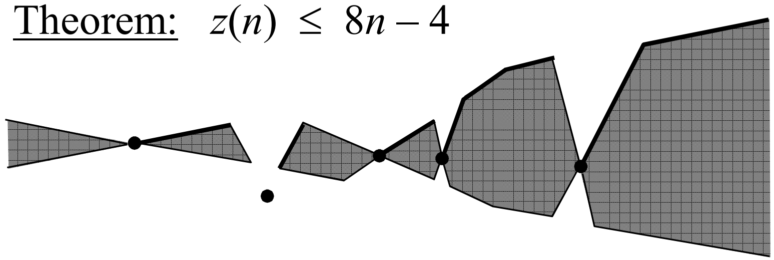
What is the combinatorial complexity of $R(S)$?

Notation:

$z(S)$ = number of edges in boundary of $R(S)$

$z(n) = \max\{z(T) \mid T \text{ a set of } n \text{ segments}\}$

Theorem: $z(n) \le 8n - 4$



76

Construction of $R(S)$: General Case

- A divide-and-conquer solution:

if $|S| = 1$ **then**
  $R(S) = D(s)$, where $S = \{s\}$
else
  $S_1 = \{s_1, \ldots, s_{\lfloor n/2 \rfloor}\}$ and $S_2 = \{s_{\lfloor n/2+1 \rfloor}, \ldots, s_n\}$
  Construct $R(S_1)$ and $R(S_2)$ recursively
  Construct $R(S) = R(S_1) \cap R(S_2)$

Time : $T(n) = 2T(n/2) + O(n) = O(n \log n)$

77

Building $R(S)$: Vertical Segments

- An incremental solution:

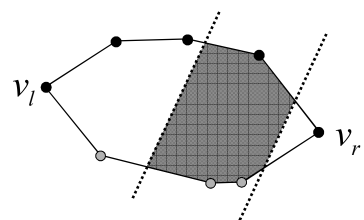
Preprocess : so that no two segments lie on a common vertical line
Initial Step : compute initial quad
  $R(S_2) \leftarrow D(S_1) \cap D(S_2)$
Iteration : Update the stabbing region
  for $i \leftarrow 3$ **to** n **do**
    $R(S_i) \leftarrow R(S_{i-1}) \cap D(s_i)$

<u>Theorem:</u> It takes O(n) time to compute $R(S)$ of a presorted list of vertical segments.

78

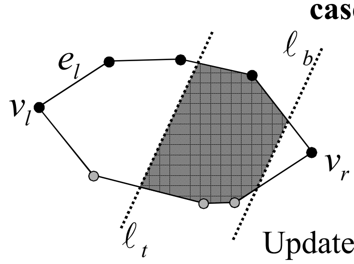
Updating the Upper Chain of $R(S)$

- Use a dequeue Q to store the edges of upper chain of $R(S_{i-1})$
- Each node of Q is contained in the line dual to the lower endpoint of a segment $s_j, j \le i-1$
- the slope of the lines of $D(s_i)$ is greater than the slopes of lines containing edges of $R(S_{i-1})$



Adding a Vertical Segment

Let $s_i = [p_b, p_t]$ and set $\ell_b = D(p_t)$, $\ell_t = D(p_b)$
Case 1 : v_l is below ℓ_b or v_r is below ℓ_t
  return $R(S) \leftarrow \emptyset$
Case 2 : $D(s_i) \cap R(S_{i-1}) \neq \emptyset$
  Update the left end of Q
    case 2.1 : $v_l \in D(s_i)$
      left end does not need to be updated
    case 2.2 : v_l lies above line ℓ_t
      $e \leftarrow e_l, e' \leftarrow \text{succ}(e_l)$
      while $\text{line}(e) \cap \text{line}(e')$ is above/on ℓ_t
        remove e, $e \leftarrow e', e' \leftarrow \text{succ}(e)$
      create new node e'' with $\text{succ}(e'') = e$,
        $\text{pred}(e) = e''$, $\text{line}(e'') = \ell_t$
  Update the right end symmetrically



Randomized Algorithms

- Use randomness as an algorithm design tool
  - Controlled randomness $\Rightarrow$ fast *expected* behavior
- Traditional probabilistic analysis
  - Makes assumption about distribution of inputs

  Example: What is the expected running time of quicksort if all n! permutations of the input are equally likely?
- New approach
  - If you don't know input distribution then force one that makes analysis possible $\Rightarrow$ average behavior independent of input
  - Analysis with respect to random choices made by algorithm for a fixed input, *not with respect to possible inputs*
  - No bad inputs, results apply to all inputs

Types of Randomized Algorithms

- Montecarlo: correctness is random
  - Probably correct, provably fast

  Example: randomized primality test
- Las Vegas: performance is random
  - Probably fast, provably correct

  Example: randomized quicksort
- Transformations
  - Las Vegas B to Montecarlo B'
    - Stop B if it is taking too long. Since B runs fast with high probability then B' is correct with high probability
  - Montecarlo A to Las Vegas A'
    - Run A until a correct answer is found

Indicator Variables

- An *indicator variable* is a random variable with sample space {0,1}

Notation. For event A, define

$$I_A = I(A) = \begin{cases} 1, & \text{if } A \text{ occurs} \\ 0, & \text{if } A \text{ does not} \end{cases}$$

- What is the expected value of an indicator variable?

$E(I_A) = 1 \cdot \Pr(A) + 0 \cdot \Pr(\neg A) = \Pr(A)$

Example

Algorithm Max(A, n)
Input. An array of integers
Output. The largest value in A

1. Randomly permute A
2. max $\leftarrow -\infty$
3. **for** $i \leftarrow 1$ **to** n **do**
4. **if** $A[i] >$ max
5. **then** max $\leftarrow A[i]$
6. **return** max

- How many times X is line 5 executed?
- X is a random variable!
- $X_i = I$(line 5 is executed in i-th iteration)
- $X = \sum_{i=1}^{n} X_i$

$E(X_i)$ = Pr(line 5 is executed in i-th iteration) = ?

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} \frac{1}{i} = H_i < 1 + \ln n$$

Incremental Construction

- The most common design technique for Las Vegas geometric algorithms
- Add the n input objects in random order, assessing the effect of each object on the solution so far
- Resulting algorithm is usually simple to program and matches complexity of optimal deterministic algorithm

85

Closest Pair

Input: set S of points in 2D

Output: distance between two closest points in S

- Deterministic algorithms take $\Theta(n \log n)$ (lower bound from *Element Uniqueness)*
- Can do better with randomized algorithm
  - Incremental, with $O(n)$ expected time
  - $d(p,q)$: Euclidean distance between p and q
  - δ^*: actual CP distance
  - δ: upper bound estimate on CP distance

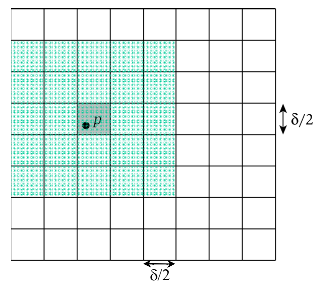
86

General Idea

- Randomly permute the points: $p_1, p_2, \ldots, p_n$
  (Let δ_i := closest pair distance for $p_1, p_2, \ldots, p_i$)
- Initially, $\delta := d(p_1, p_2) = \delta_2$
- For $i = 3$ to n, update $\delta := \delta_i$

Key Problem: How do we know if p_i updates δ?

- Keep a partition of the plane into $\delta/2 \times \delta/2$ cells
  - Neighborhood of p is 5×5 sub-grid centered at p's cell



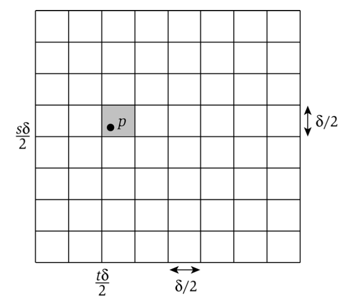
Properties

- No two points lie in same cell
  - If points p and q lie in the same cell then $d(p,q) < \delta$
- If $d(p,q) < \delta$ then q is in p's neighborhood
- Enough to check neighborhood of p_i
  1. Neighborhood not empy $\Rightarrow$ for each point p_j present ($j<i$) check $d(p_i, p_j)$, update δ, and regrid with new δ if necessary
  2. Neighborhood empty $\Rightarrow \delta_i = \delta_{i-1}$

88

Managing the Grid

- Operations: create, insert, lookup
- Use a "dictionary", e.g., a hash table H
- H stores non-empty grid cells
- Cells have integer coordinates (s,t)



$$S_{st} = \{(x, y) : s\delta/2 \le x < (s+1)\delta/2;\ t\delta/2 \le y < (t+1)\delta/2\}$$

$$p = (x, y) \to \underbrace{\left(\left\lfloor \frac{x}{\delta/2} \right\rfloor, \left\lfloor \frac{y}{\delta/2} \right\rfloor\right)}_{\text{key for } (x,y)}$$

89

A Las Vegas Algorithm

```
Order the points in a random sequence p_1, p_2, ..., p_n
Let δ denote the minimum distance found so far
Initialize δ = d(p_1, p_2)
Invoke MakeDictionary for storing subsquares of side length δ/2
For i = 1, 2, ..., n:
  Determine the subsquare S_st containing p_i
  Look up the 25 subsquares close to p_i
  Compute the distance from p_i to any points found in these subsquares
  If there is a point p_j (j < i) such that δ' = d(p_j, p_i) < δ then
    Delete the current dictionary
    Invoke MakeDictionary for storing subsquares of side length δ'/2
    For each of the points p_1, p_2, ..., p_i:
      Determine the subsquare of side length δ'/2 that contains it
      Insert this subsquare into the new dictionary
    Endfor
  Else
    Insert p_i into the current dictionary
  Endif
Endfor
```

90

Analysis

- What is the cost of the i-th iteration?
  - $O(1)$ if δ does not change
  - $O(i)$ if δ changes

$$X_i = I\{p_i \text{ changes } \delta\} = \begin{cases} 0 & \text{if } \delta \text{ does not change} \\ 1 & \text{if } \delta \text{ changes} \end{cases}$$

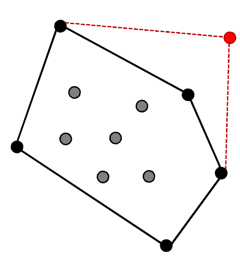
$$E[X_i] = \Pr\{p_i \text{ changes } \delta\} \le 2/i$$

$$E[T(n)] = n + \sum_{i=1}^{n} E(X_i)O(i) = O(n)$$

91

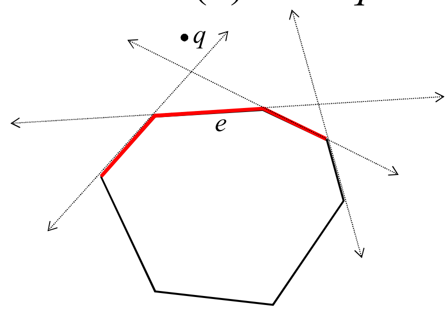
A Randomized Convex Hull Algorithm

- Will describe a Las Vegas Algorithm
  - Provably correct, probably fast
- *Idea*: randomized incremental algorithm
- The expected running time depends on the random order of insertion but is independent of the input data
- The method generalizes to higher dimensions



92

- Let T be a set of points and q a point *outside* conv(T). An edge e of conv(T) is *visible* from q if the supporting line of e separates conv(T) from q



Note: The edges of conv(T) visible from q

- Are not part of conv($T \cup \{q\}$)
- Form a contiguous chain

A Randomized Algorithm

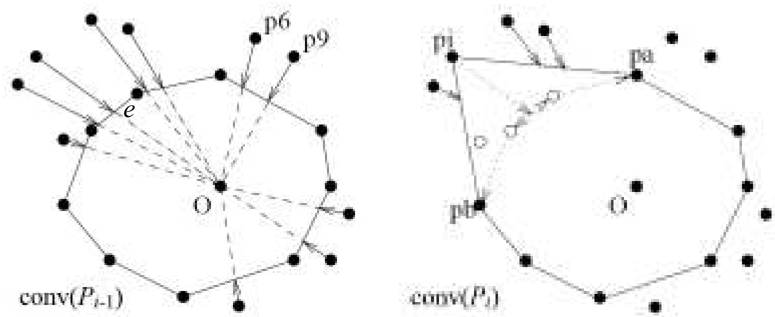
- Randomly permute the input points $P = \langle p_1, p_2, \ldots, p_n \rangle$ and let $P_i = \langle p_1, p_2, \ldots, p_i \rangle$

Algorithm ConvexHull(P, n)
Input. An array P of points in the plane
Output. The convex hull of P

1. Randomly permute P
2. conv(P_3) $\leftarrow \Delta(p_1, p_2, p_3)$
3. **for** $i \leftarrow 4$ **to** n **do**
4. **if** p_i not in conv(P_{i-1}) **then**
5. $e \leftarrow$ any edge of conv(P_{i-1}) visible from p_i
6. $C \leftarrow$ chain $\langle q_j, \ldots, q_k \rangle$ of edges of conv(P_{i-1}) visible from p_i
7. conv(P_i) is obtained by replacing C with chain $\langle q_j, p_i, q_k \rangle$

- Ignoring the cost of lines 4 and 5, algorithm runs in $O(n)$ time!

- A *conflict edge* for p, denoted e_p, is any edge visible from p. If p is interior then it has no conflict edge.
- Every point p outside the convex hull keeps a conflict edge e_p. Conversely, every edge e keeps the list L_e of points that list e as their conflict edge.
  - How do we compute it and maintain conflict information?



95

Updating Conflict Information

- Whenever an edge e is deleted we need to update the conflict information for all points p such that $e_p = e$.
- Each affected point either becomes an interior point or its conflict edge changes.
- What should we do in each case?
  - Interior points can be deleted now or later
  - Updating e_p takes constant time per exterior point as only 2 candidate edges need to be considered $\Rightarrow$ cost proportional to number of points updated.

96

Analysis

- There are three tasks the algorithm performs in each iteration
  1. *Creation* of two new edges
  2. *Destruction* of a variable number of old edges
  3. *Reclassification* of points whose conflict edge was removed
- How much time do you spend in each?
  – Creation and destruction of edges takes $O(n)$ time
  – Reclassification time is a random variable whose expected value is computed by *backward analysis*

97

Backwards Analysis

- Pretend to run the algorithm backwards
  – running time same as running forward
  – Easier to estimate probability of reclassification
- What is the probability that e_p changes while "deconstructing" $\mathrm{conv}(P)$, i.e., while computing $\mathrm{conv}(P_{i-1})$ from $\mathrm{conv}(P_i)$?
  – If a segment is removed from $conv(P_i)$ we must update all the pointers in its conflict list
  – An edge is removed only iff one of its endpoints is p_i
  – Since each of the remaining i points is equally likely to be chosen for removal, each edge of $conv(P_i)$ is removed with probability $2/i$

98

Backwards Analysis…

- Let X_i = # of pointers updated in the ith iteration, then:

$$E[X_i] = \sum_{e \in \mathrm{conv}(P_i)} \left((\text{size of } e'\text{s conflict list}) \cdot (\Pr(e \text{ is removed})\right)$$

$$= \Pr(e \text{ is removed}) \cdot \left(\sum_{e \in \mathrm{conv}(P_i)} \text{size of } e'\text{s conflict list} \right)$$

$$= 2/i \cdot O(n) = O(n/i)$$

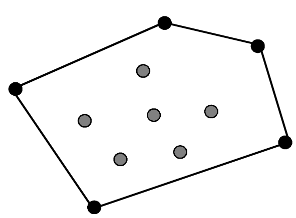
- Total expected time, adding across all iterations is

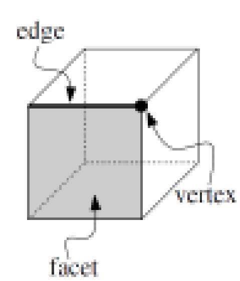
$$E[\#\text{pointer updates}] = \sum_{i=4}^{n} O(n/i) = O(n \log n)$$

99

Convex Hulls in $\mathbb{R}^d$

- The *convex hull* of a set of points P in $\mathbb{R}^d$, denoted conv(P), is the intersection of all convex sets containing P
- Goal: design an efficient algorithm that computes conv(P) from P for $d \geq 2$
  - The vertices of conv(P) are the *extreme points* of P.
  - Output is a complete specification of the boundary, including all i-dimensional faces (i = 0,…d – 1) , plus their adjacencies

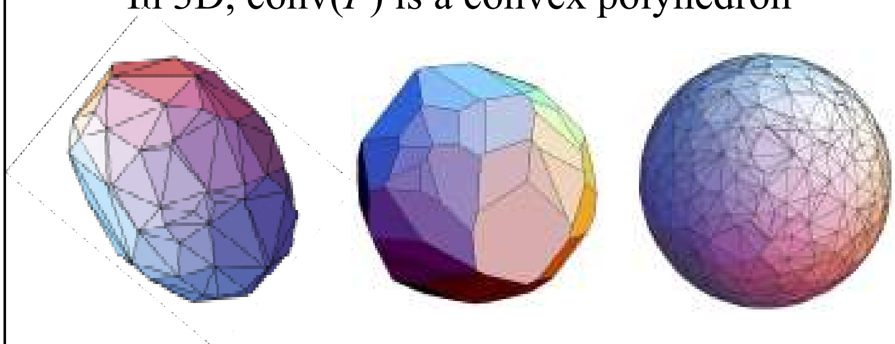




100

A Preview of 3D Convex Hull

- In 3D, conv(P) is a convex polyhedron



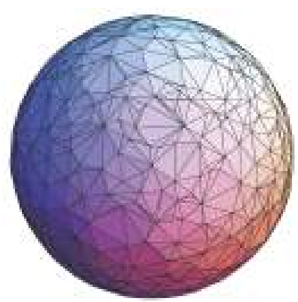
- It consists of *vertices*, *edges*, *facets*, and their incidences

101

Issues

- If $|P| = n$, how big is conv(P)?
  - How many vertices, edges, faces

  Example: $n = 758$, $e = 2268$, $f = 1512$



- What is a good data structure to store conv(P)?
  - Space should be linear in $|\text{conv}(P)| = n + e + f$
  - Retrieve all adjacencies efficiently
- Does the randomized incremental approach generalize?

102