CMSC 754 - Computational Geometry
Lecture 1: Introduction

What is Computational Geometry?

- Subfield of algorithm theory involving
  discrete geometric structures
  - points, lines + line segments, polygons, spatial subdivisions

  in 2-dimensional
  3-dimensional
  low dimensional
  high dimensional

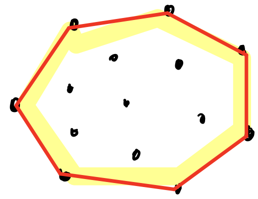
  more
  less

Features:

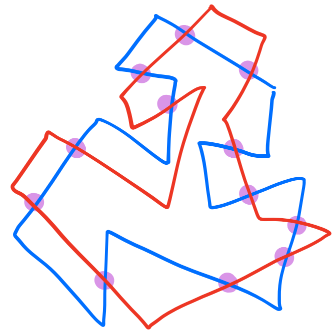
- Worst-case asymptotic complexity
  deterministic + randomized
- Rigorous - provably correct + efficient (in theory)
- Discrete inputs/outputs
- Combinatorial-based analysis
- "Simple" geometry - flat, Euclidean
- Low dimensionality

Topics:

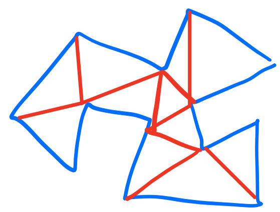
- Convex hulls

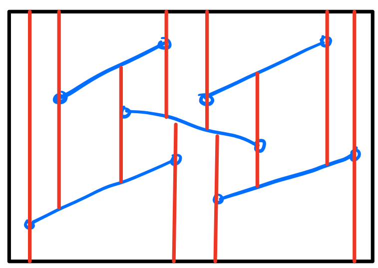


- Intersections



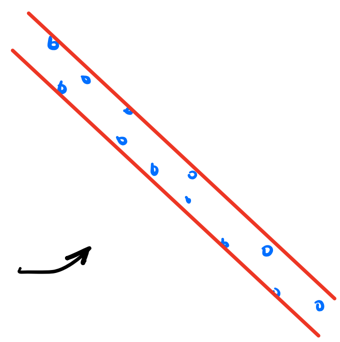
- Triangulations + spatial subdivisions



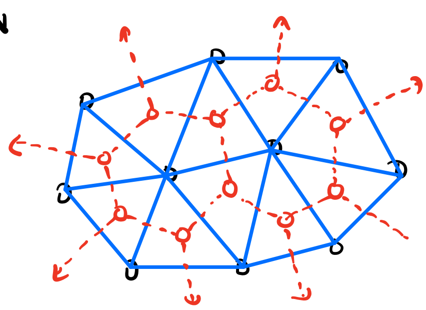


- Point location

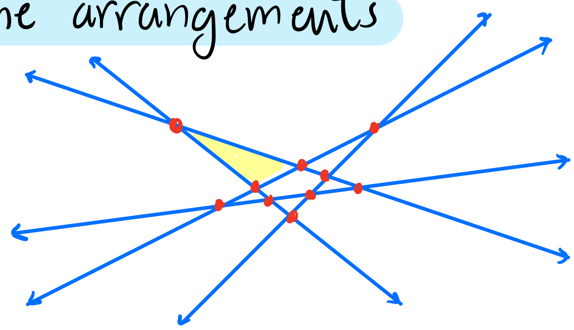
- Linear programming + duality



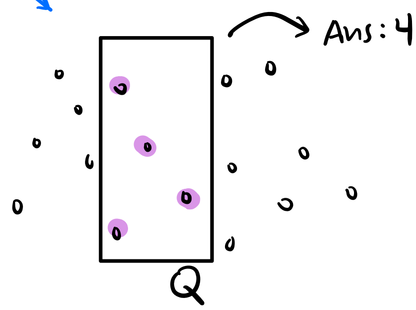
- Voronoi diagrams + Delaunay triangulations



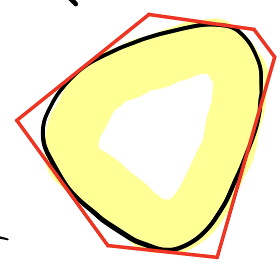
- Line/hyperplane arrangements



- Search + Data Structures



- Approximation
    - ε-nets
    - ε-kernels + coresets

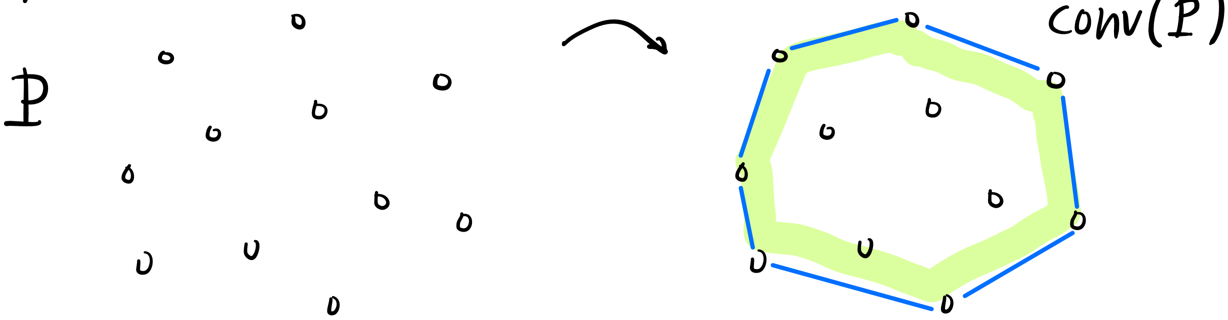


- More? High dimensional geometry
  Computational topology

CMSC 754 - Computational Geometry
Lecture 2: Convex Hulls in the Plane

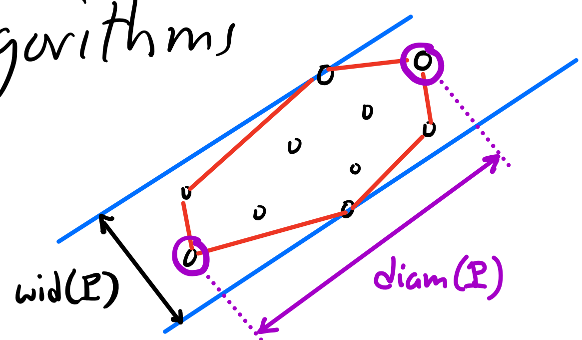
Convex Hull: (Intuitive definition)

Given a point set P in $\mathbb{R}^2$, imagine snapping a rubber band around the points



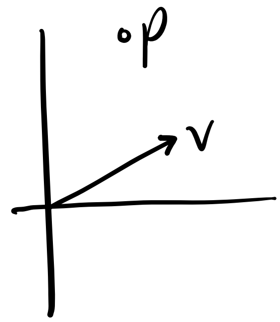
Uses:

- shape approximation (intersection test)
- first step in other algorithms
  - diameter
  - width



Basic Definitions:

$\mathbb{R}^d$ - Real d-dim space $p = (p_1, \ldots, p_d)$ $p_i \in \mathbb{R}$
- Refer to as
  points (p, q) - location
  or vectors (u, v, w) - displacement

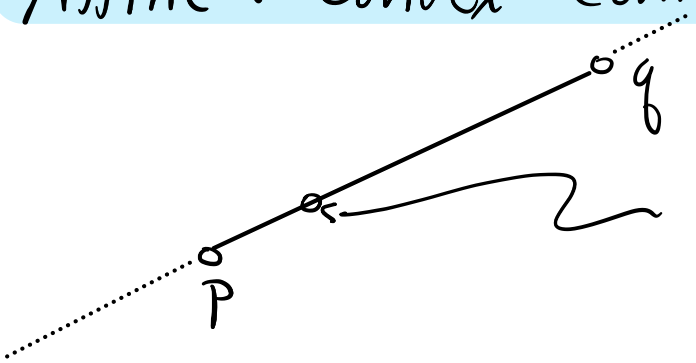


$\mathbb{R}$ - scalars $\alpha, \beta, \gamma, \ldots$

usual ops from linear algebra:

$u+v$, $u-v$	- vector addition
$\alpha \cdot u$	- scalar multiplication
$u \cdot v$	- dot product $= \sum_{i=1}^{d} u_i v_i$

Affine + Convex Combinations:



for $\alpha \in \mathbb{R}$
$(1-\alpha)p + \alpha q$

Generally given $p_1 \ldots p_k$:

Affine combination: $\sum_{i=1}^{k} \alpha_i p_i$ $\sum \alpha_i = 1$

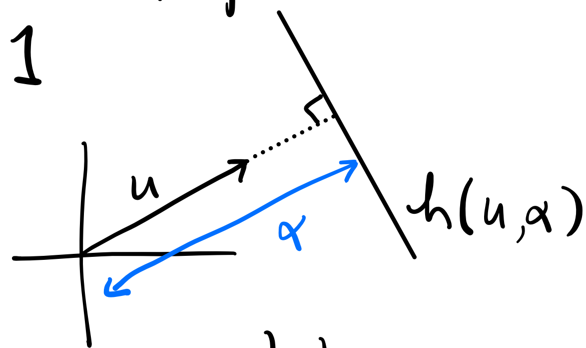
Convex combination: ... and $0 \leq \alpha_i \leq 1$

Lines, Hyperplanes, Halfspaces:

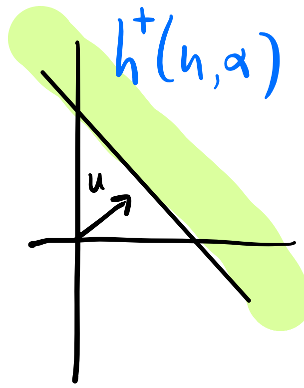
Given nonzero vector u + scalar α,

$h(u,\alpha) = \{ p \in \mathbb{R}^d \mid p \cdot u = \alpha \}$ is hyperplane

If $\|u\| = 1$



$h^+(u,\alpha) = \{ p \in \mathbb{R}^d \mid p \cdot u \geq \alpha \}$

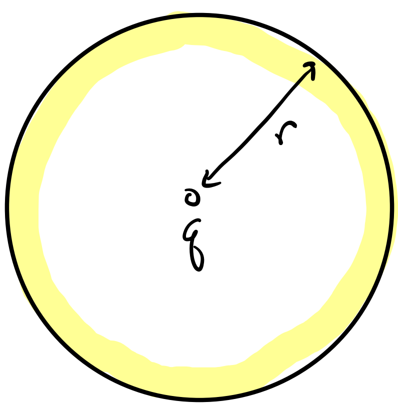


Euclidean Ball:

$$\mathrm{dist}(p,q) = \|p-q\| = \left(\sum_{i=1}^{d} (p_i - q_i)^2 \right)^{1/2}$$

$B(q,r) = \{ p \in \mathbb{R}^d \mid \|p-q\| \leq r \}$

(Euclidean) ball of radius r centered at q.



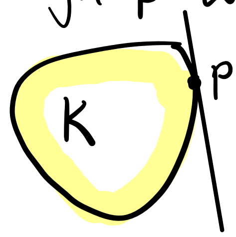
Convexity:

A set $K \subseteq \mathbb{R}^d$ is convex if $\forall p, q \in K$ the line segment $\overline{pq}$ (equiv. any conv. combination of $p + q$) lies within K

Support Hyperplane:

Boundary of K

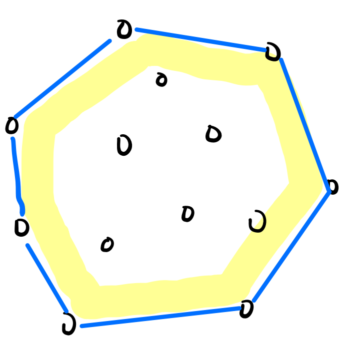
Given convex K and any point $p \in \partial K$, $\exists$ hyperplane passing through p with K lying all on one side.



Convex Hull:

Given a set P of points in $\mathbb{R}^d$, the convex hull, $\text{conv}(P)$, is the smallest convex set containing P.

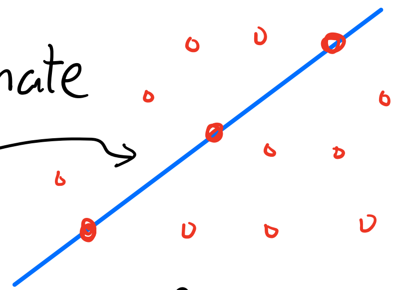
- The set of all convex combs in P
- The intersection of all halfspaces containing P



General Position:

Geometric algorithms are complicated by rare (?) degenerate cases:

- points having same coordinate
- ≥ 3 collinear points
- ≥ 4 cocircular points



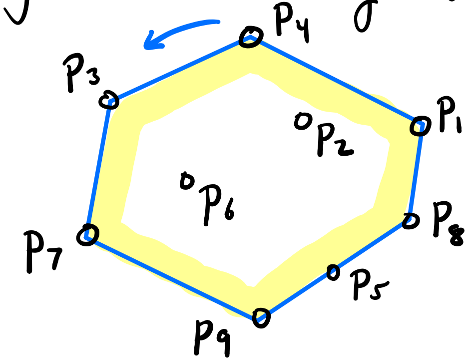
To simplify algorithm presentation we often assume these do not arise in the input.

Called **general-position assumption**

(Planar) Convex Hull Problem:

Given a set of n pts $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$ $(p_i = (x_i, y_i))$ compute $\mathrm{conv}(P)$.

Output: Cyclic ordering of vertices on the hull



Possible output: (indices)

$\langle 4, 3, 7, 9, 8, 1 \rangle$

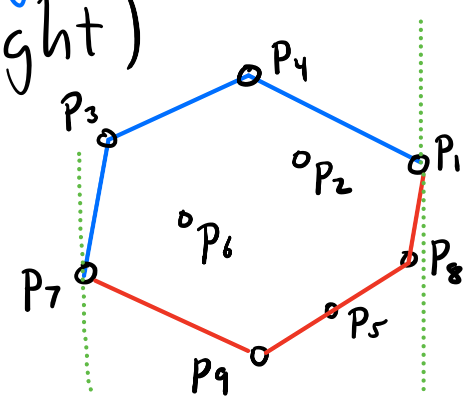
Note: p_5 not output

(Can assume this away by "general position")

Alternative output: (left to right)

Upper-hull + **Lower-hull**

$\langle 7, 3, 4, 1 \rangle + \langle 7, 9, 8, 1 \rangle$

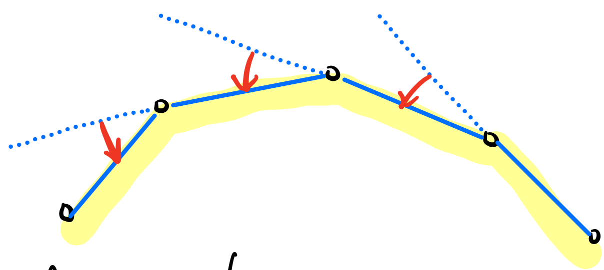


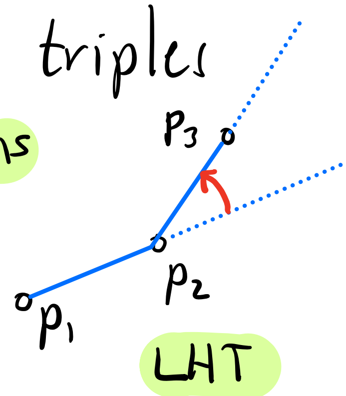
Graham's Scan: $O(n \log n)$ solution

- Compute upper + lower hulls separately
- Upper-hull:
  - Sort pts by x-coords
  - Add each to upper hull
  - Remove pts no longer on hull ← How?
- Lower-hull: (symmetrical)

Observations:

- The rightmost pt always on hull
- Reading right to left, consecutive triples on the hull form left-hand turns

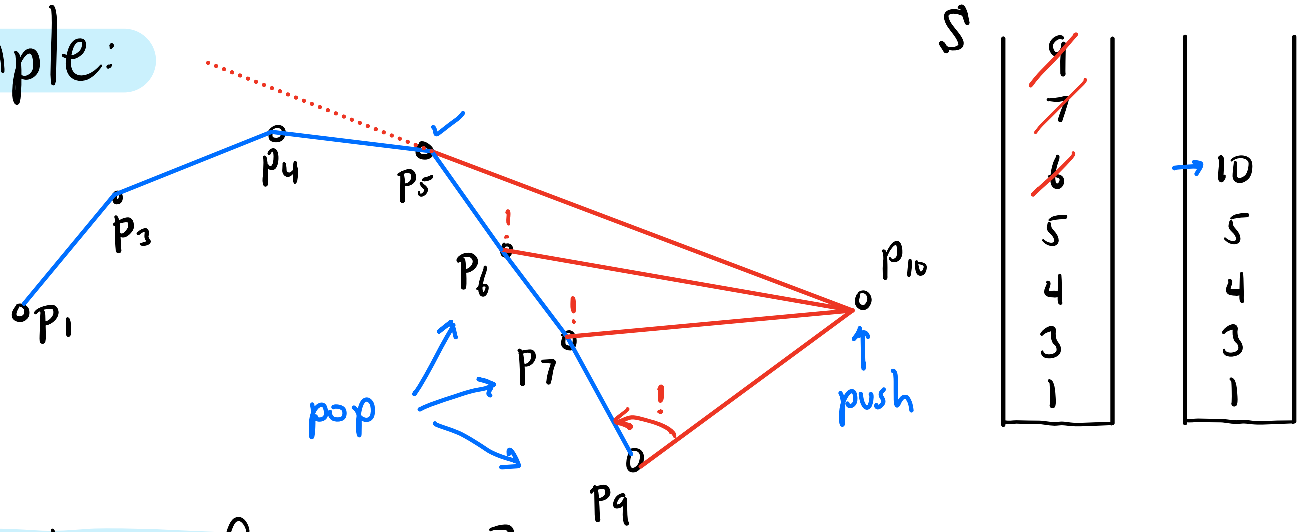




Incremental Approach:

- Store vertices (indices) of upper hull on stack
- For each new point p_i (left to right)
  - While $\langle p_i, S[top], S[top-1] \rangle$ do not form LHT — pop S
  - Push p_i

Example:



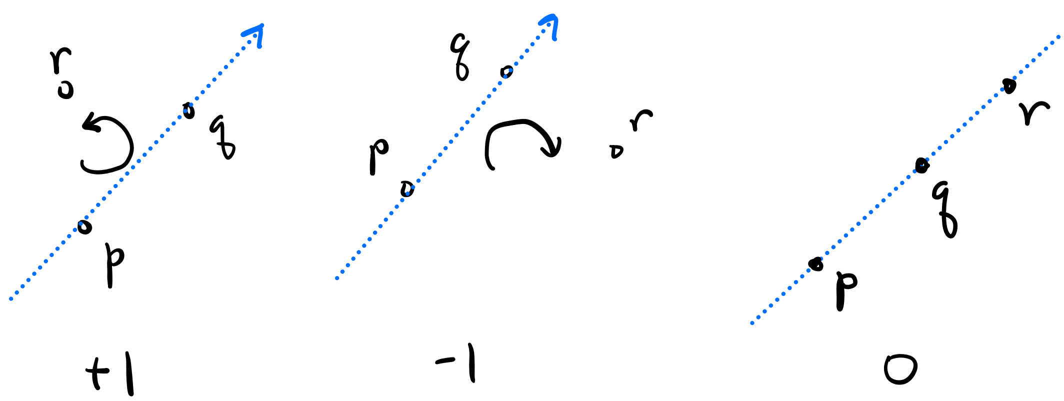
How to test for LHT?

Orientation test

Given a sequence $\langle p, q, r \rangle$ of 3 pts in $\mathbb{R}^2$

$$\text{orient}(p,q,r) = \text{sign}\left(\det\begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}\right)$$

is: +1 if they are oriented CCW (LHT)
-1 " " " " CW (RHT)
0 if they are collinear (or duplicates)



Graham's Scan: (Upper Hull only)

- Sort pts by increasing x-coords $\langle p_1 \dots p_n \rangle$
- Push p_1 + p_2 onto S
- for $i \leftarrow 3$ to n
  - while ($|S| \geq 2$ and $\text{orient}(p_i, S[t], S[t-1]) \leq 0$) pop S

    t = "top"
  - push p_i

Correctness:

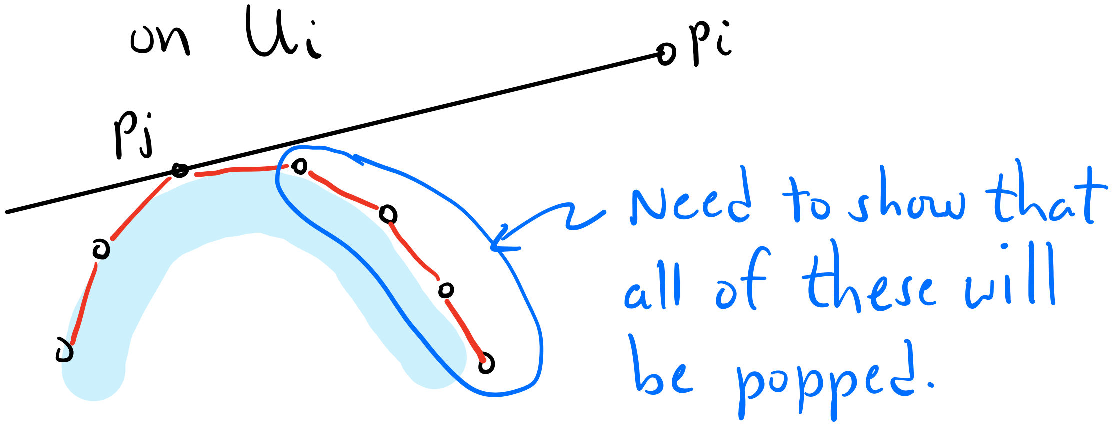
Lemma: After processing p_i, S contains upper hull of $\langle p \dots p_i \rangle$

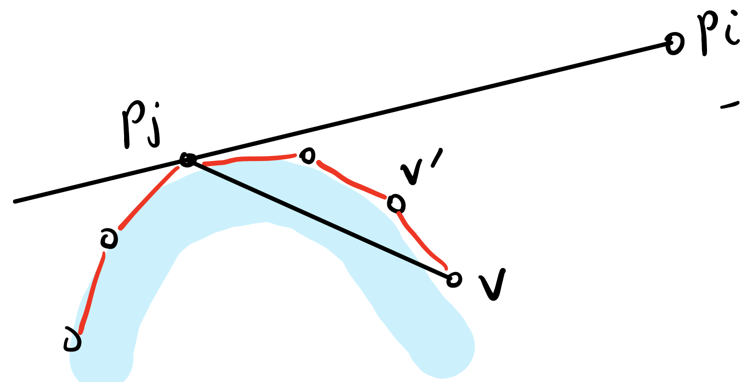
Proof: By induction on i.

Basis: $i = 1, 2$ – Trivial

Step: For $i \geq 3$, let U_i denote the vertices on upper hull up to p_i.

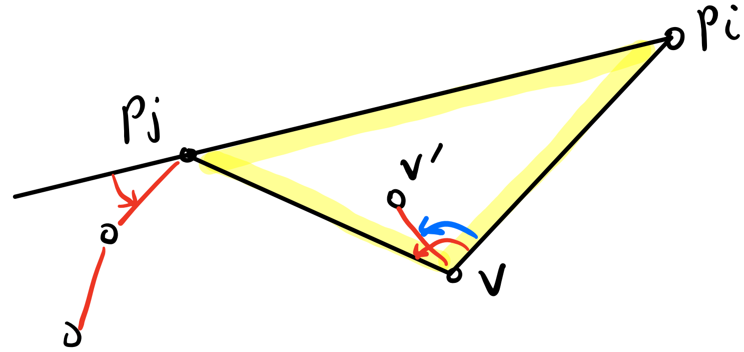
- By induction U_{i-1} is correct up to p_{i-1}
- We'll show its correct after adding p_i
- Let p_j denote vertex before p_i on U_i





- Let v be current vertex + let v' be predecessor

- By convexity:
  - all pts lie below $\overline{p_i p_j}$
  - all pts after p_j lie above $\overline{p_j v}$



$\Rightarrow$ v' lies in $\Delta p_j v p_i$

$\Rightarrow$ $\angle p_i v v' \leq \angle p_i v p_j \leq 2\pi$

$\Rightarrow$ orient$(p_i, v, v') \leq 0$

$\Rightarrow$ v is popped off stack

On arriving at p_j, orientation flips so popping stops at p_j + finally p_i pushed $\square$

Running time:

- $O(n \log n)$ to sort
- for $3 \le i \le n$, let d_i = num. of pops when inserting p_i
- Time for scan is $\sim$

$$\sum_{i=3}^{n} (d_i + 1) \le n + \sum_{i=3}^{n} d_i$$

for pops   for push of p_i

- Note that $\sum d_i \le n$ → Why?

- Total time: $O(n \log n + 2n) = O(n \log n)$

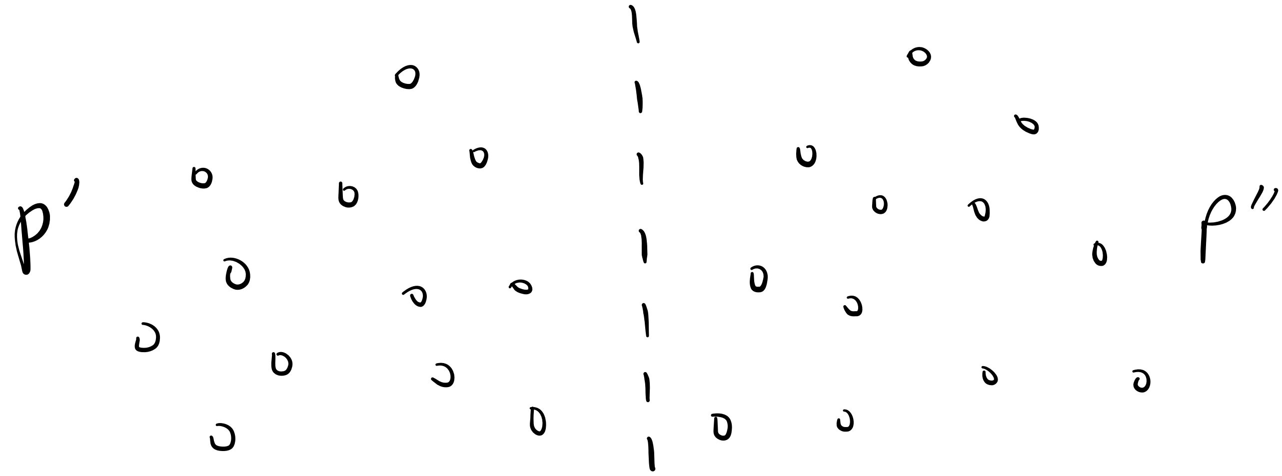
Divide + Conquer Algorithm:

Given point set $\underline{P}$:

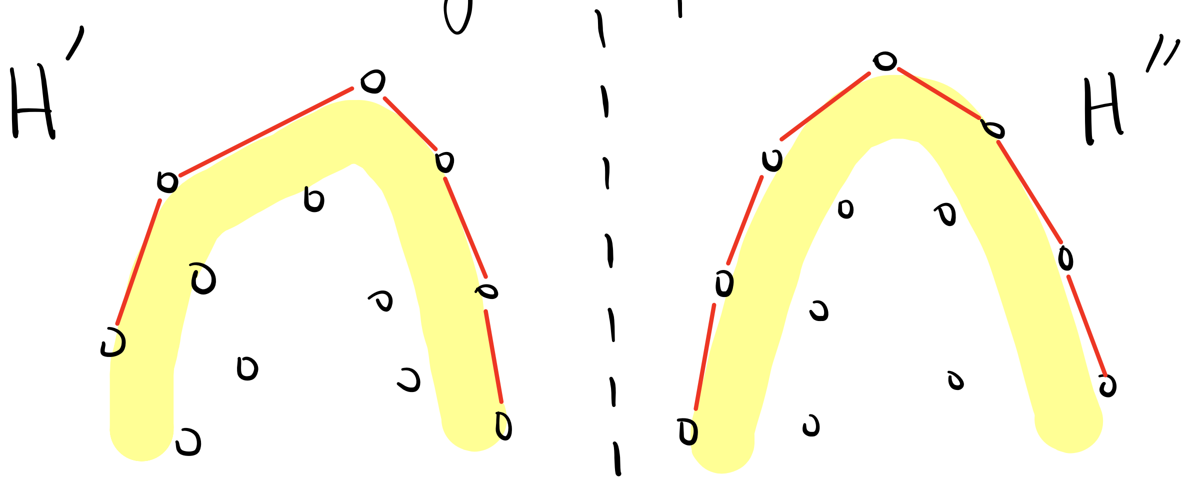
if $|P| \le 3$ then compute hull by brute force ($O(1)$)

else

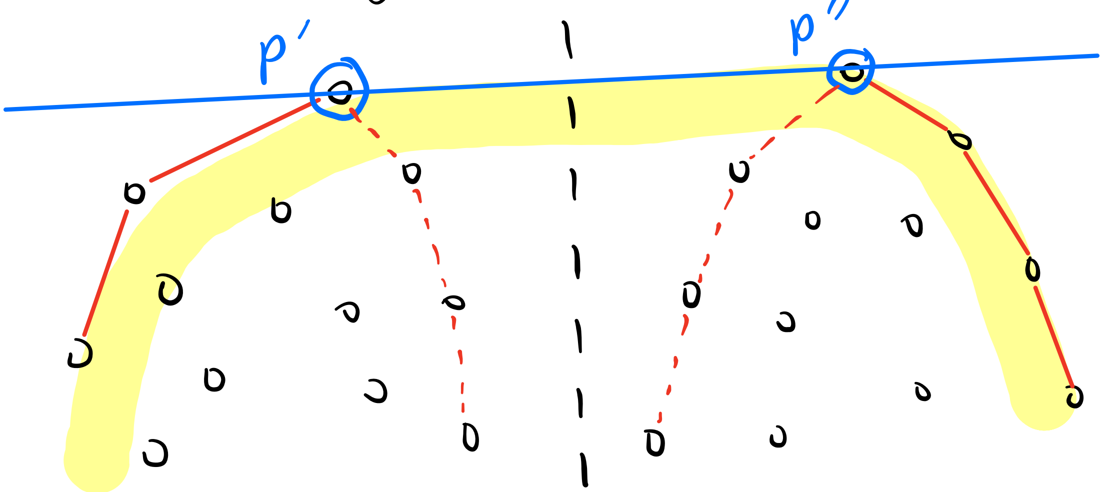
- Partition $\underline{P}$ by vertical line into P', P'' of sizes $\sim n/2$



- Recursively compute hull of each



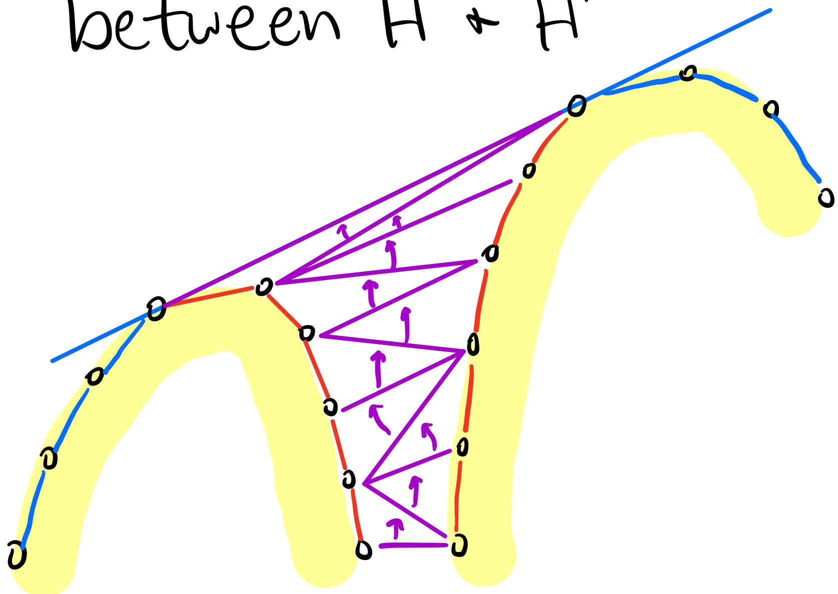
- Compute pts $p' \in H'$ + $p'' \in H''$ defining upper tangent



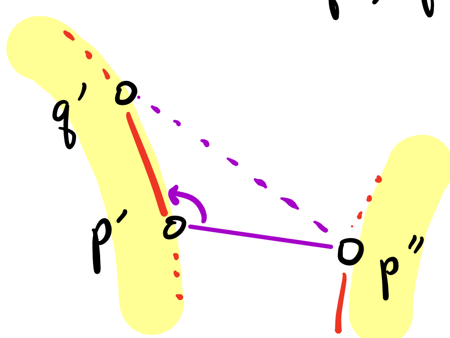
- Merge partial hulls together

How to compute upper tangent?

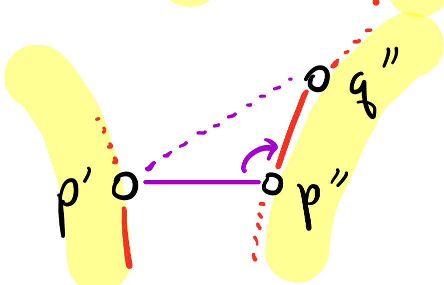
- Start with a chord joining closest points (w.r.t. x) of H' + H''
- "Walk" this chord up the ladder between H' + H''



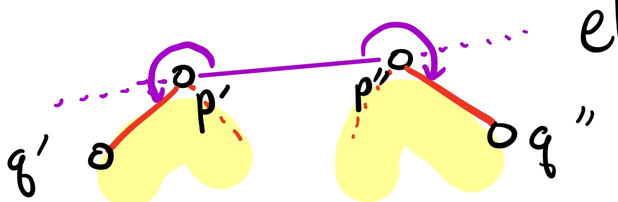
- How? Let p', p'' be current vertices
  Let q', q'' be vertices above



if $\text{orient}(p'', p', q') \le 0$
$p' \leftarrow q'$



if $\text{orient}(p', p'', q'') \ge 0$
$p'' \leftarrow q''$



else done!

Correctness? (Exercise)

Running time?

- $O(n)$ time to find upper tangent by walking
  - Each step takes $O(1)$ time
  - Eliminates one vertex
- Gives recurrence:

$$T(n) = \underbrace{2T(n/2)}_{\text{Recursively compute two hulls, each from } n/2 \text{ pts}} + \underbrace{n}_{\text{Split, tangent, merge}}$$

- Same as Mergesort. By Master Theorem

$$T(n) = O(n \log n)$$

Jarvis March: An $O(nh)$ algorithm

Idea: Compute any one vertex of hull $\rightarrow v_1$

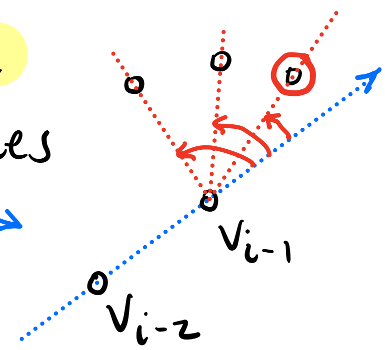
for $i = 2, 3, \ldots$

  compute next vertex v_i on hull

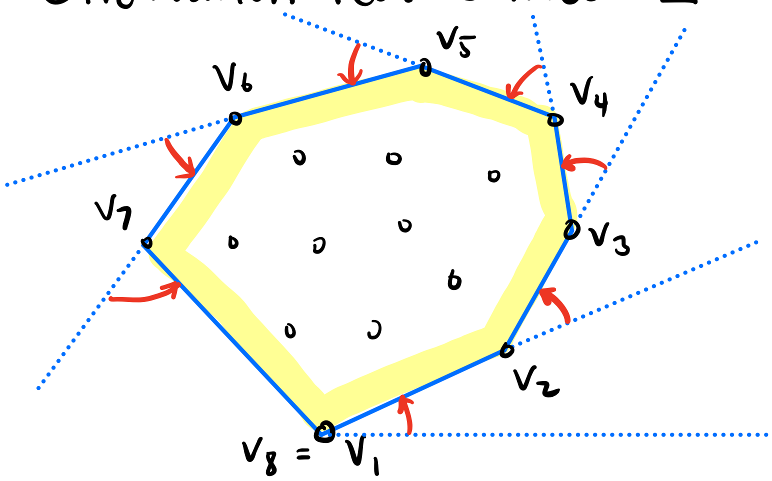
  if $(v_i == v_1)$ return $\langle v_1, \ldots, v_{i-1} \rangle$

v_1? Point of P with min y-coordinate

next vertex? The point of P that minimizes turn angle w.r.t. prior two vertices



[This doesn't require trig. Orientation test suffices]



Correctness: Easy

Running time: Compute v_1 - $O(n)$

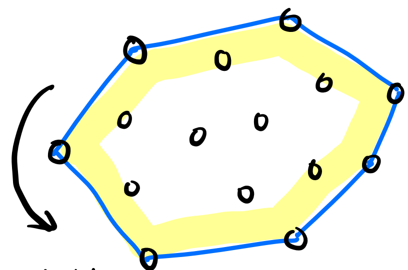
Compute v_i - $O(n)$ ← Repeat h times

Total: $O((h+1)n) = O(h \cdot n)$

CMSC 754 - Computational Geometry
Lecture 3: Convex Hulls - Chan's Algorithm

Recap:

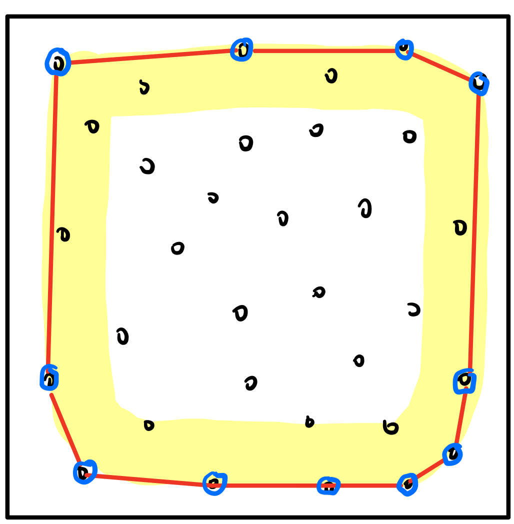
- Given a pt. set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$ compute conv(P) - smallest convex set containing P.
- Output: Cyclic sequence of hull vertices



- Algorithms: Graham's Scan $O(n \log n)$, Div & Conquer $O(n \log n)$, Jarvis March $O(n \cdot h)$

This Lecture:

- Can we beat $O(n \log n)$ time?
  - → We'll give an $\Omega(n \log h)$ lower bound
- Can we achieve $O(n \log h)$?
  - → Chan's algorithm



- Good when number of hull vertices h is very small

Theorem: Given n pts unif. distributed in a unit square $E[h] = \log n$

(Chan runs in $O(n \log \log n)$)

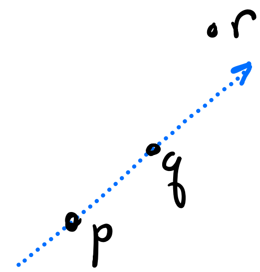
Lower bound for convex hulls:

Conv: Given a set P of n pts in $\mathbb{R}^2$, compute the vertices of $\mathrm{conv}(P)$ in cyclic order.

Def: An algorithm is comparison-based if its decisions are based on the sign of a fixed-degree polynomial function of inputs. (Algebraic decision tree model)

Almost all geometric primitives satisfy:

E.g. if($\langle p, q, r \rangle$ form a left-hand turn)



$$\equiv \text{if}\left(\det\begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix} > 0\right)$$

$$\equiv \text{if}\left(f(p_x, p_y, q_x, q_y, r_x, r_y) > 0\right)$$

where:

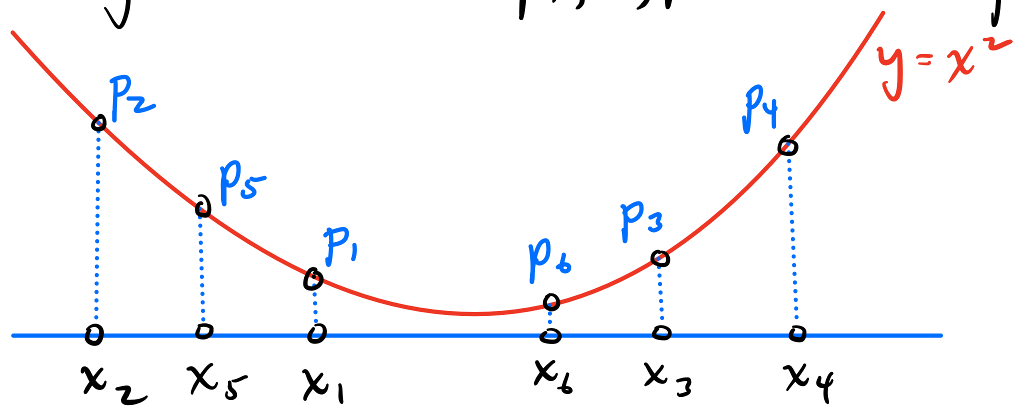
$$f(\dots) = (q_x r_y - q_y r_x) - (p_x r_y - p_y r_x) + (p_x q_y - p_y q_x)$$

A polynomial of degree 2

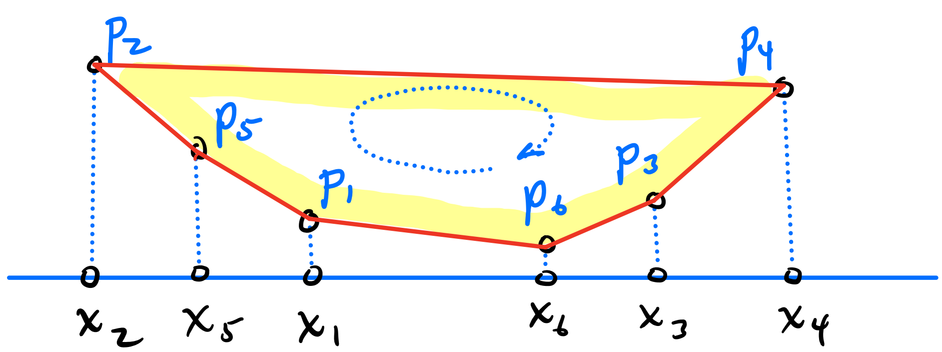
Theorem: Assuming a comparison-based algorithm, conv has a worst-case lower bound of $\Omega(n \log n)$

Proof: We will use the well-know fact that any comparison-based alg. for sorting reqs. $\Omega(n \log n)$ time in worst case.

We'll reduce sorting to conv. Given set $X = \{x_1, \dots, x_n\}$ to be sorted in $O(n)$ time we generate $P = \{p_1, \dots, p_n\}$ where $p_i = (x_i, x_i^2)$



If we compute conv(P), the vertices appear in sorted order of X, up to reversal and adjusting starting point ← $O(n)$ time



Letting $T(n)$ denote the time to compute $\text{conv}(P)$, up to constant factors, we can sort X in time

$$n + T(n) + n, \text{ which must be } \geq c \cdot n\log n$$

compute P from X — reorient output

$$\Rightarrow T(n) \geq c \cdot n\log n - 2n \Rightarrow T(n) = \Omega(n\log n)$$

$\square$

Obs: This exploits the fact that output is sorted cyclically. What if not?

Theorem: Assuming a comparison-based algorithm determining whether $\text{conv}(P)$ has h distinct vertices requires $\Omega(n \log h)$ time.

$\Rightarrow$ Just counting vertices reqs. log factor.

(See latex lecture notes for proof)

Output Sensitivity: Algorithm's running time depends on output size

$\rightarrow$ Is $O(n \log h)$ possible?

Yes!

Chan's Algorithm

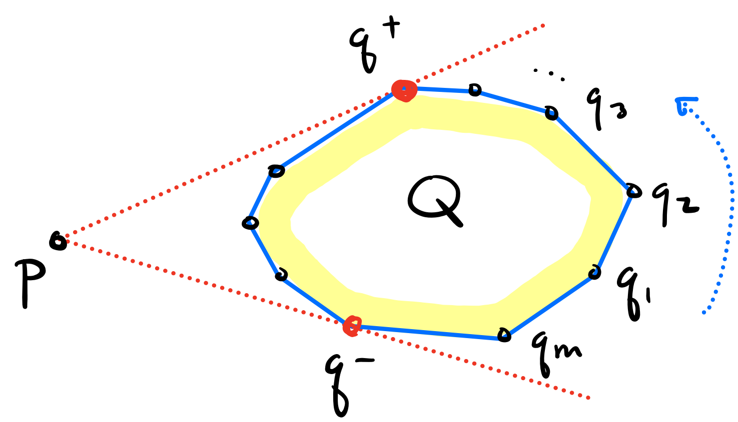
- combines - Graham scan $O(n\log n)$
  + Jarvis March $O(nh)$

Chan's Algorithm: An $O(n \log h)$ algorithm

- Optimal w.r.t. input size n + output size h
- Combines two slow algorithms (Graham + Jarvis) to make faster algorithm
- Chicken + Egg: Algorithm needs to know value of h - How is this possible?

Tangent Lemma:

Given a convex polygon Q given as a cyclic sequence of m vertices $\langle q_1, \ldots, q_m \rangle$ and $p \notin Q$, can compute tangent vertices q^- + q^+ w.r.t. p in time $O(\log m)$



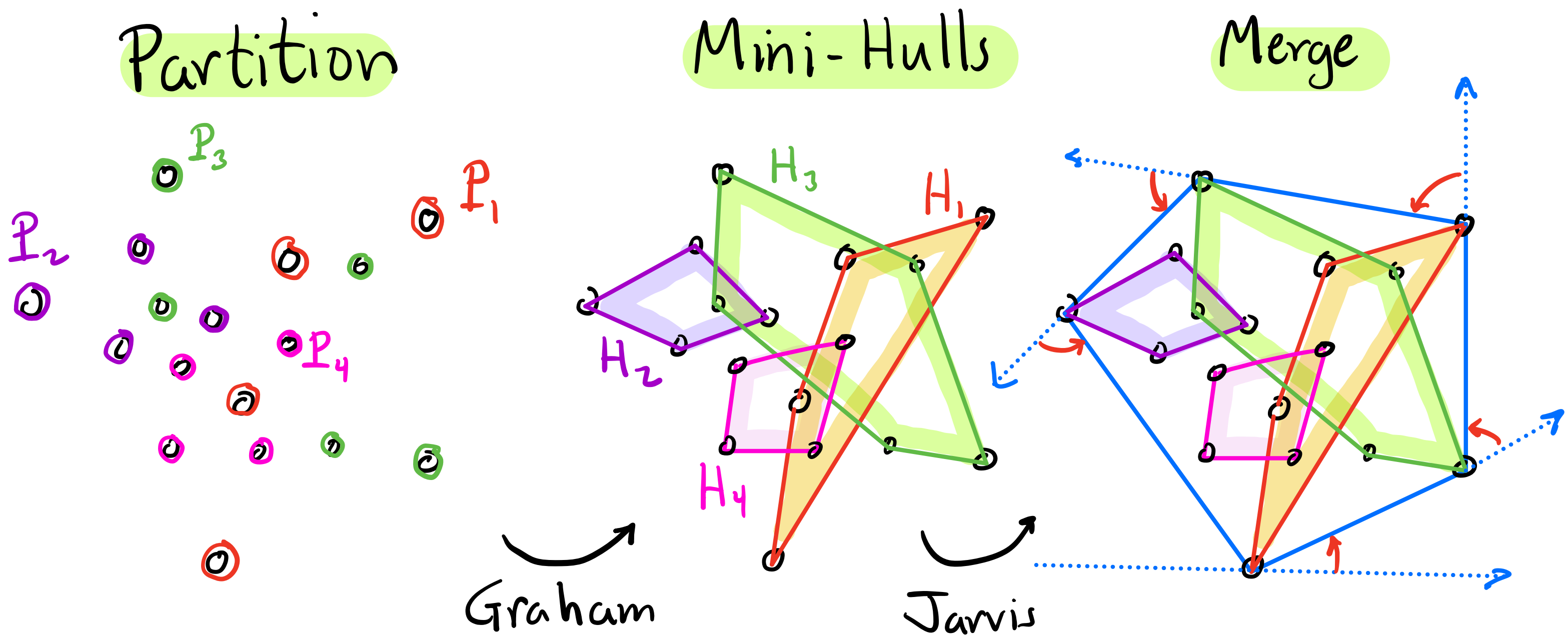
How? Exercise

Hint: Variant of binary search

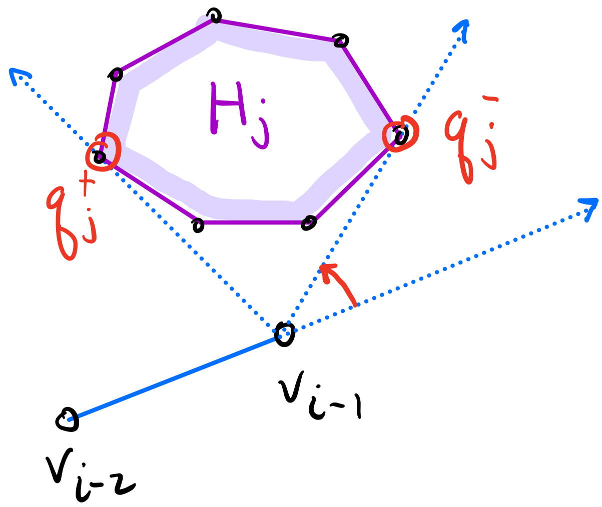
How to achieve $O(n \log h)$?

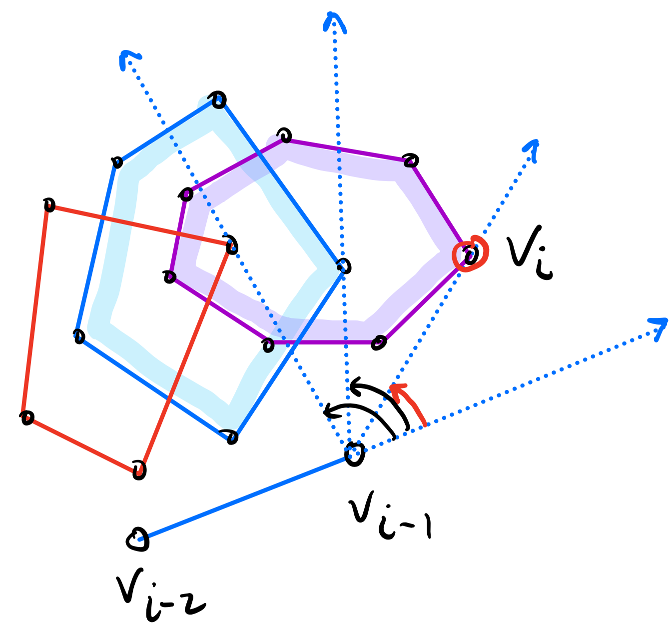
- Can't sort any set of size $\gg h$
- Guess the hull size – h^*
- Partition P into $\lceil n/h^* \rceil$ groups, each of size $\leq h^*$
  $\to P_1, \dots, P_k$, $k = O(n/h^*)$ $\longrightarrow O(n)$
- Run Graham on each group forming k "mini-hulls" $H_1, \dots, H_k$ $\longrightarrow O(k \cdot h^* \log h^*)$
  $= O(n \log h^*)$
  - If we guess right $(h^* = h) \to O(n \log h)$
- Run Jarvis, but treat each mini-hull as a "fat point"
  - use the utility function to compute turning angles

Example: Suppose $k = 5$



Merging Mini-hulls:





- By the Tangent Lemma, compute tangents q_j^- + q_j^+ for each H_j in time $O(\log h^*)$
- Compute all tangents in time $O(k \cdot \log h^*)$
- $v_i \leftarrow$ tangent with smallest turning angle
- Terminates after h iterations

$\Rightarrow$ Total merge time: $O(h \cdot k \cdot \log h^*)$

$\rightarrow$ If we guess right ($h^* = h$) then

$$O\left(h^*\left(\frac{n}{h^*}\right) \log h^*\right) = O(n \log h^*)$$
$$= O(n \log h)$$

Summary: If we guess correctly ($h^* = h$) this computes $\mathrm{conv}(P)$ in time $O(n \log h)$.

Conditional Hull (P, h^*):

Mini-hull Phase: $O(n \log h^*)$

Merge Phase: $O(n \frac{h}{h^*} \log h^*)$

If $h^* > h$ $\Rightarrow$ Mini-hull phase is too slow

Note: Can tolerate a polynomial error. E.g. if $h \le h^* \le h^2$

$$\Rightarrow O(n \log h^*) = O(n \log(h^2)) = O(2 \cdot n \log h) = O(n \log h) \text{ ok.}$$

If $h^* < h$ $\Rightarrow$ Merge phase too slow

- If Jarvis finds more than h^* hull pts - stop + return fail status

$\Rightarrow O(n \log h^*)$ time

Strategy:

Start small and increase until success

Arithmetic: $h^* = 3, 4, 5, \ldots$ way too slow $\rightarrow O(n \cdot h \cdot \log h)$

Exponential: $h^* = 4, 8, 16, \ldots, 2^i$ better $\rightarrow O(n \log^2 h)$

Double Exponential: $h^* = 4, 16, 256, \ldots 2^{2^i}$

best!

Note: $h_i^* = 2^{2^i}$ $h_i^* \leftarrow (h_{i-1}^*)^2$

Final Algorithm:

ChanHull(P):

```
h* = 2
repeat
    h* ← (h*)^2          ⤳ h*_i = 2^(2^i)
    (status, V) ← conditionalHull(P, h*)
until (status == success)
return V
```

Correctness: Already explained

Time:

- Running time per iteration $O(n \log h^*)$
- $h^*_i = 2^{2^i}$
- Stops when $h^* \geq h$

  $2^{2^i} \geq h \Rightarrow i = \lceil \lg \lg h \rceil$ iterations
- **Total time:** [up to constants]

$$\sum_{i=1}^{\lg \lg h} n \cdot \lg\left(2^{2^i}\right) = n \sum_{i=1}^{\lg \lg h} 2^i$$

$$\leq 2n \, 2^{\lg \lg h} \quad \text{[Geom series]}$$

$$= 2n \lg h$$

$$= O(n \lg h)$$

Lower Bound: (Optional)

Convex Hull Size Verification (CHSV):

Given a planar point set P of size n + int k, does $\text{conv}(P)$ have h vertices?

> **Thm:** CHSV requires $\Omega(n \log h)$ time to solve (worst case in the algebraic-tree decision model)

Take away - Just **counting** num. of hull vertices takes $\Omega(n \log h)$ time.

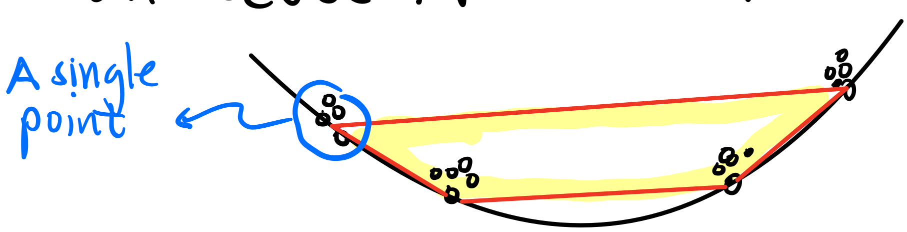
Proof (sketch):

Multiset Verification Problem (MSV):

Given a set S of n real numbers and integer k, does $|S| = k$?

Known: MSV has lower bound of $\Omega(n \log k)$

Can reduce MSV to CHSV in linear time

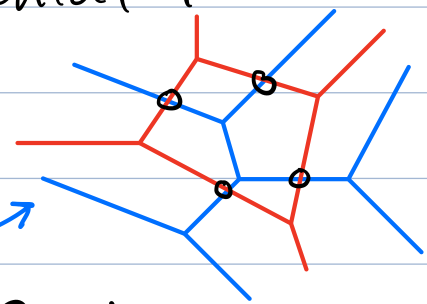


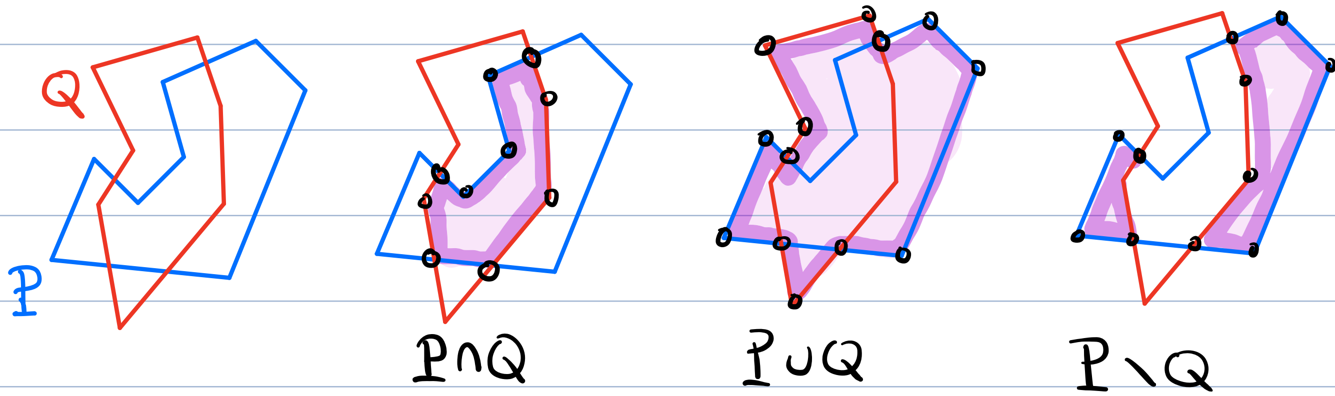
Map $z_i \to (z_i, z_i^2)$

CMSC 754 - Computational Geometry
Lecture 4: Line Segment Intersection

Computing intersections is fundamental to geometric computation

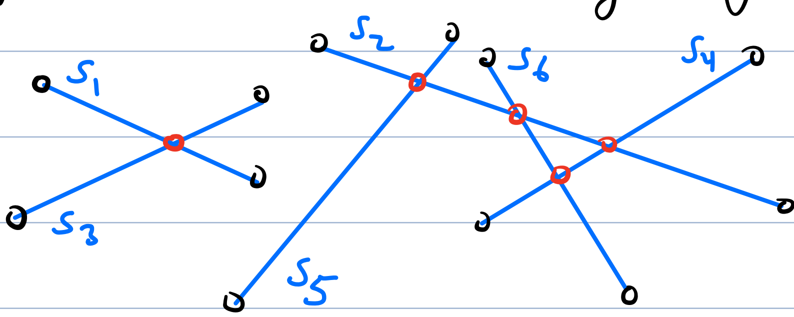
- collision detection
- subdivision overlay
- boolean operations – $\cap, \cup, \ldots$





Line Segment Intersection:

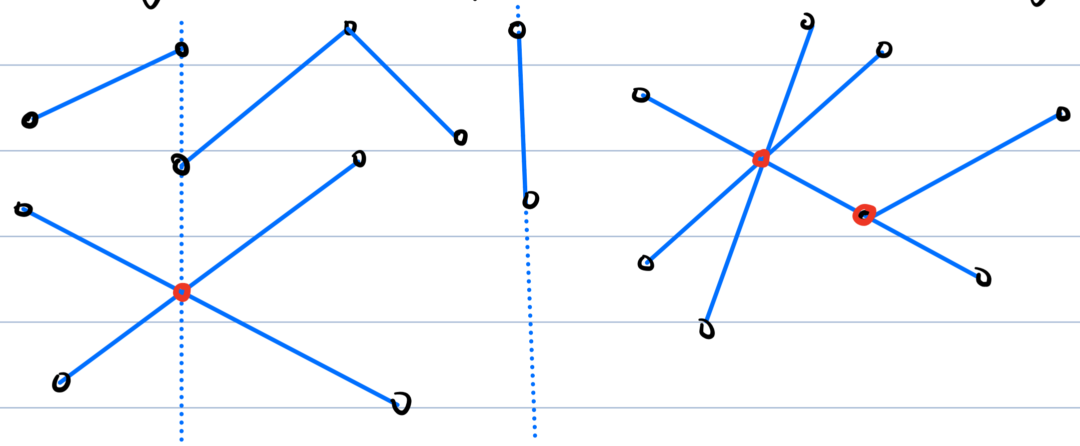
Given a set $S = \{s_1, \ldots, s_n\}$ of line segments in $\mathbb{R}^2$ (where $s_i = \overline{p_i q_i}$), report all pairs of intersecting segments.



(s_1, s_3)
(s_2, s_5)
(s_2, s_6)
$\vdots$

General Position Assumptions:

- No duplicate x-coords (for both endpoints + intersections)
- No segment endpt on another segment



Output Sensitivity:

Input size: n (2n endpts, 4n coords)

Output size: m

$$0 \le m \le \binom{n}{2} = O(n^2)$$

Best possible: $O(m + n\log n)$

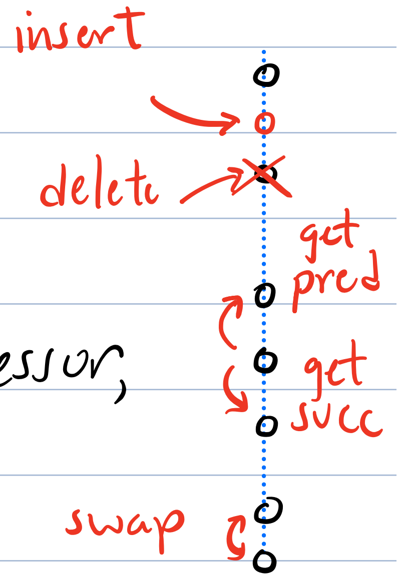
Follows from a lower bound on element uniqueness

This lecture: $O((n+m)\log n)$

Plane sweep

Utility Data Structures:

Ordered Dictionary: Supports:



insert, delete, find,
get-predecessor, get-successor,
swap adjacent

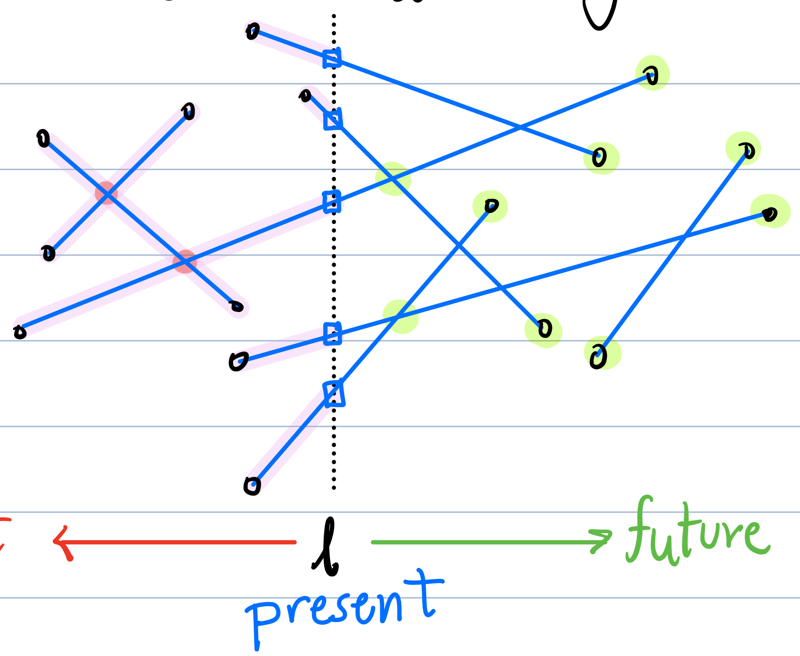
all in $O(\log n)$ time + $O(n)$ space

Priority Queue: Stores object σ + priority x

ref $\leftarrow$ enqueue(σ, x)
$\sigma \leftarrow$ extract_min() - removes obj w. min priority
delete(ref)

Sweep-Line Algorithm:

Sweep a vertical line ℓ from left to right + update solution as we go.

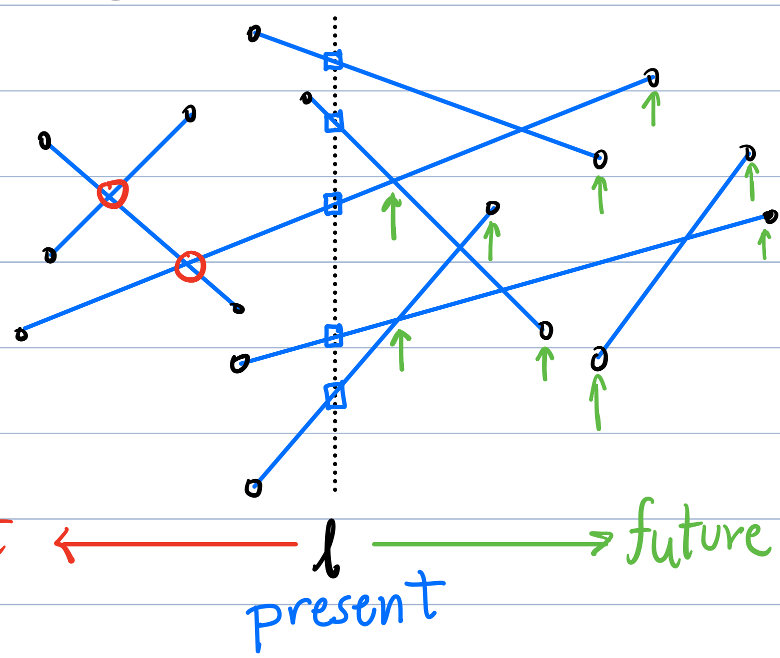


What we store: (Generic Plane Sweep)

(Past) Partial solution to left of ℓ

(Present) Current status along ℓ

(Future) (Known) Events to right of ℓ



What we store: (For segment intersection)

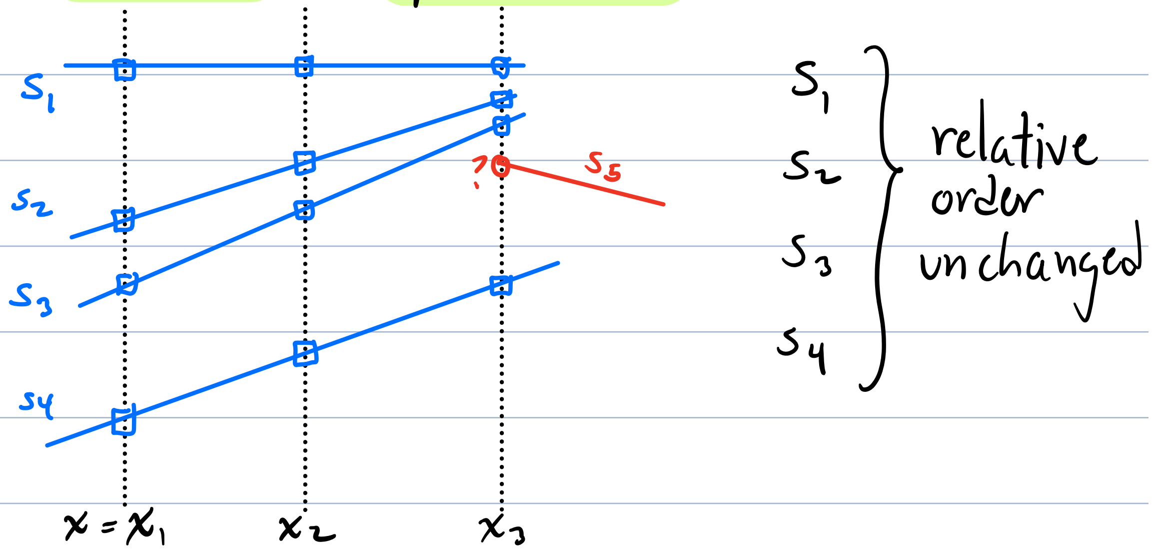
Past: List of intersecting pairs so far

Present: Ordered dictionary (top to bottom, say) of segments intersecting ℓ
— sweep-line status

Future: Priority queue with future events:
- segment endpts to right of ℓ
- "imminent" intersections right of ℓ

Sweep-Line Status:

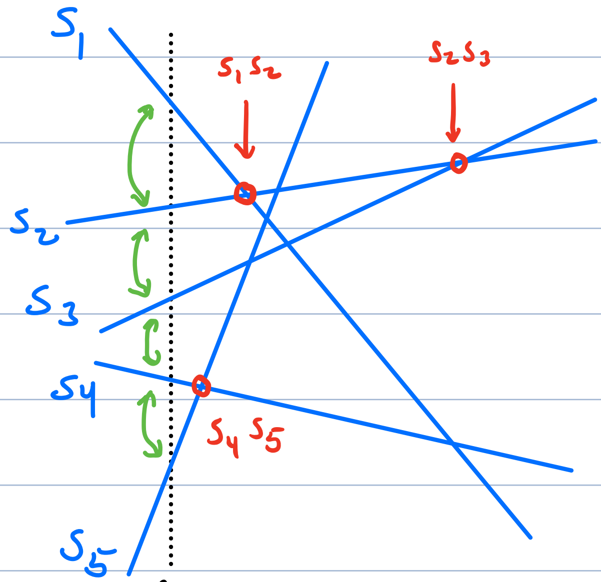
- As ℓ moves, all y-coordinates on sweep line change
- Much too slow to update all



- Dynamic comparator: Rather than storing y coords in dictionary, store line equation: $y = ax + b$
  - As x changes, reevaluate to compare y based on current x value

Future Events: (Stored in priority queue)

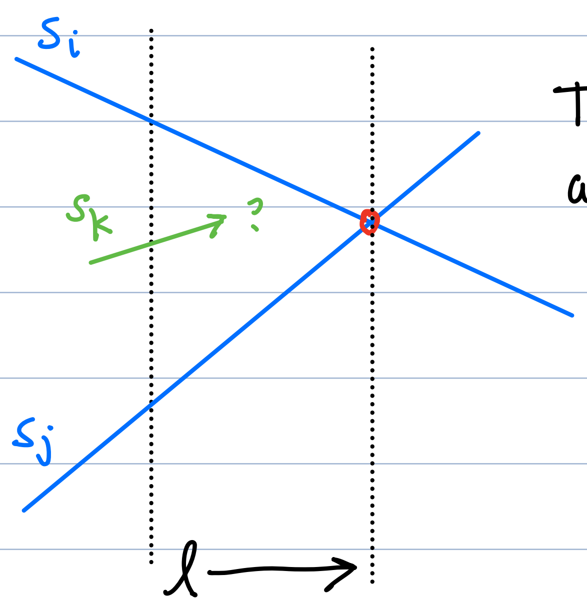
- All segment endpts to right of sweep line
- "Imminent" intersections: Intersections between pairs of lines that are consecutive on sweep line



Why? - Consecutive pairs are easy to detect + update
- At most $n-1 = O(n)$ intersection events in priority queue (+ $\leq 2n$ end pt events)

Lemma: If the next event is an intersection, these segments will be consecutive on the current sweep line.

Proof:
- Suppose not
- $s_i s_j$ is next event, but not consecutive

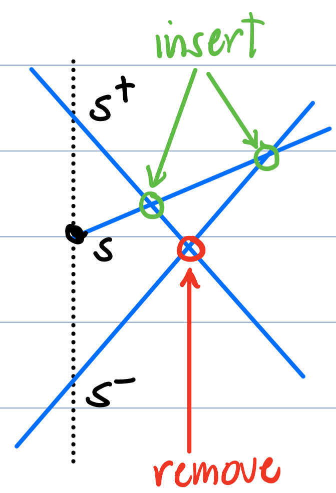


There must be an event involving s_k first

Final Sweep-Line Algorithm: $S = \{s_1, \dots, s_n\}$ $s_i = \overline{p_i q_i}$

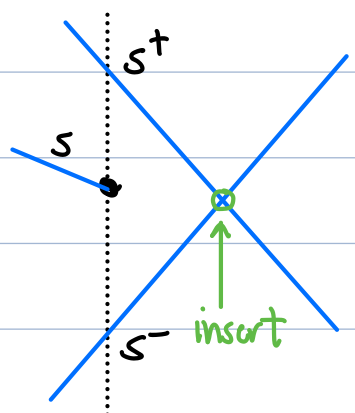
- Insert all seg. endpts into priority queue (sorted by x-coord)
- while(queue is non-empty) {
    - extract next event (min x)
    - cases:

Segment s left endpt:



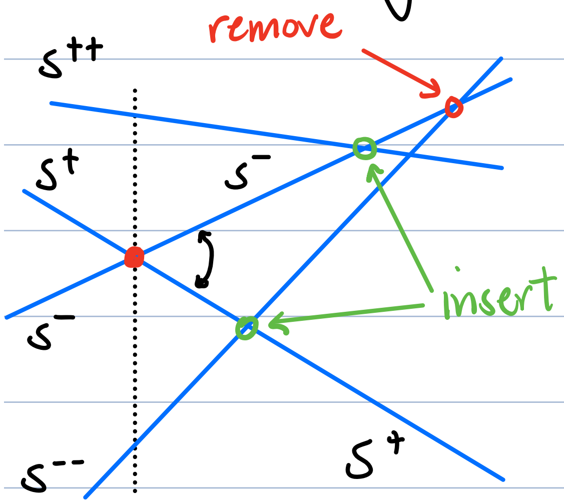
- Insert segment into sweep line (dictionary) based on y-coord
- Let s^+ & s^- be segs just above and below
- If s^+s^- has intersection event, remove from priority queue
- Add to priority queue, intersection events for ss^+ & ss^- (if appropriate)

Segment s right endpt:



- Let s^+ & s^- be segments above & below
- Add to priority queue, intersect event for s^+s^- (if appropriate)

Segment s^+s^- intersection:



- Let s^{++} + s^{--} be segs above and below intersection
- Remove intersection events s^+s^{++} + s^-s^{--} (if exist)
- Swap s^+ + s^- on sweep line
- Add to prior. queue, intersect events for s^+s^{--} + s^-s^{++} (if appropriate)

Correctness: Easy, but be sure not to forget anything

Running Time: n = num. of segs. m = num. of intersects

Total events: $2n + m = O(n+m)$

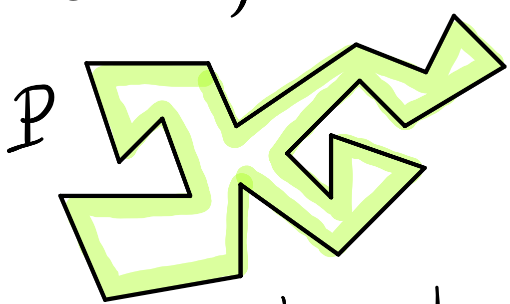
Time per event: Extract min
$O(1)$ dictionary ops
$O(1)$ queue ops
} $O(\log n)$ total

Total time: $O((n+m)\log n)$

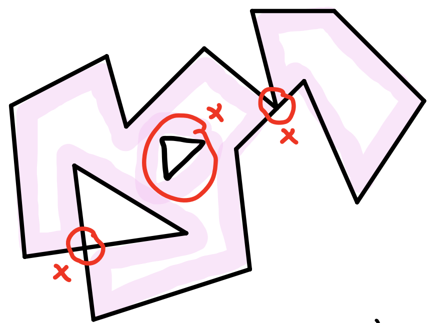
Space: $O(n)$ for data structures
$O(m)$ for output

CMSC 754 - Computational Geometry
Lecture 5: Polygon Triangulation

Polygon Triangulation: Given a **simple polygon** P (that is, a simple, closed polygonal chain)...

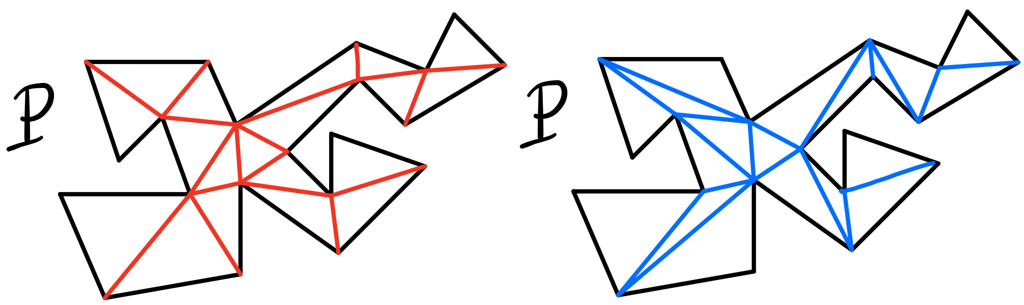


simple polygon



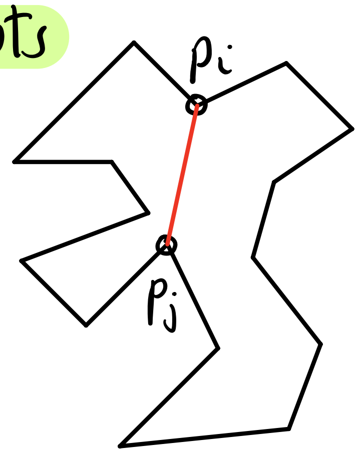
not simple

subdivide the interior of P into triangles (vertices drawn from P's vertices)



Notes: - P given as a **cyclic seq. of pts**

- Vertices p_i + p_j are **visible** if open segment $\overline{p_i p_j} \subseteq \operatorname{int}(P)$
- If p_i + p_j visible, segment $\overline{p_i p_j}$ called a **diagonal**



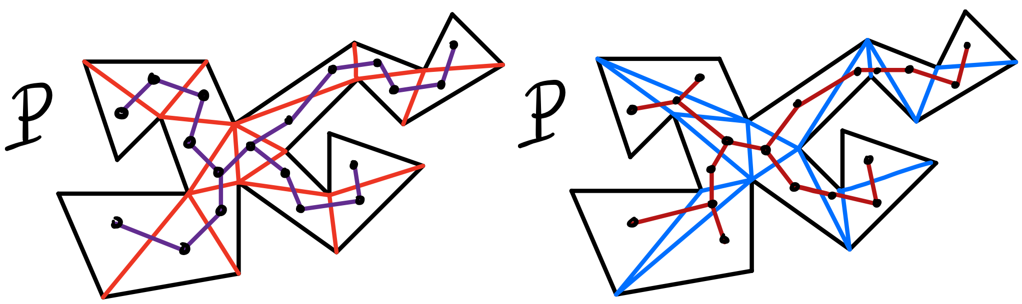
Lemma: Given any n-vertex simple polygon ($n \geq 3$)

- A triangulation exists
- Any triangulation has $n-3$ diagonals
- Any triangulation has $n-2$ triangles

Dual Graph: A triangulation defines a graph:

Vertices ← triangles

Edges ← adjacent (share common edge)



The dual graph of a polygon triangulation is connected + acyclic ⇒ tree

History of Polygon Triangulation:

$O(n^2)$ - Easy (find a diagonal + recurse)

$O(n \log n)$ - We'll present this

$O(n)$ - Chazelle 1991 (very complicated!)

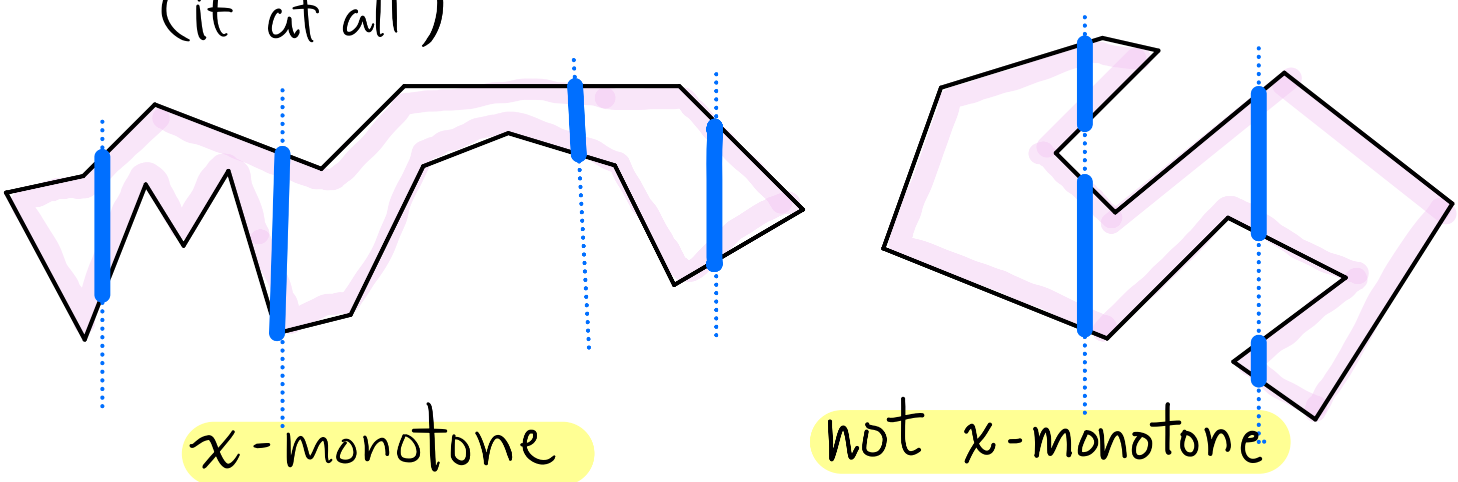
Two steps:

① Decompose the polygon into (simpler) polygons - monotone polygons - $O(n \log n)$

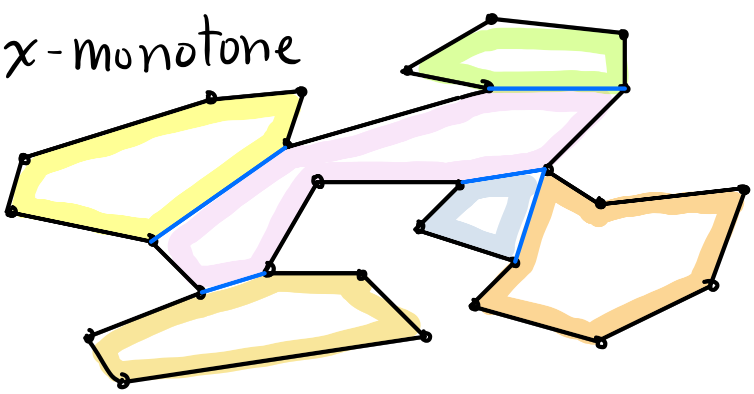
② Triangulate each monotone polygon - $O(n)$

Output: Graph structure, called a doubly-connected edge list (DCEL)

Def: A polygon is x-monotone if any vertical intersects the polygon in a single segment (if at all)



Monotone Decomposition - Add (non-intersecting) diagonals so that connected components are all x-monotone

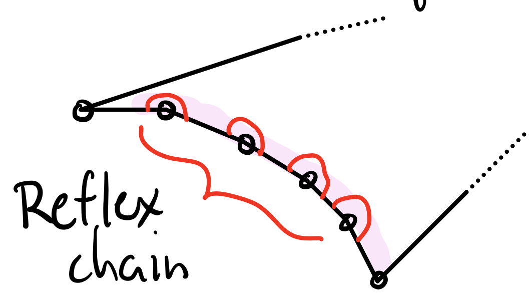


Triangulating a Monotone Polygon:

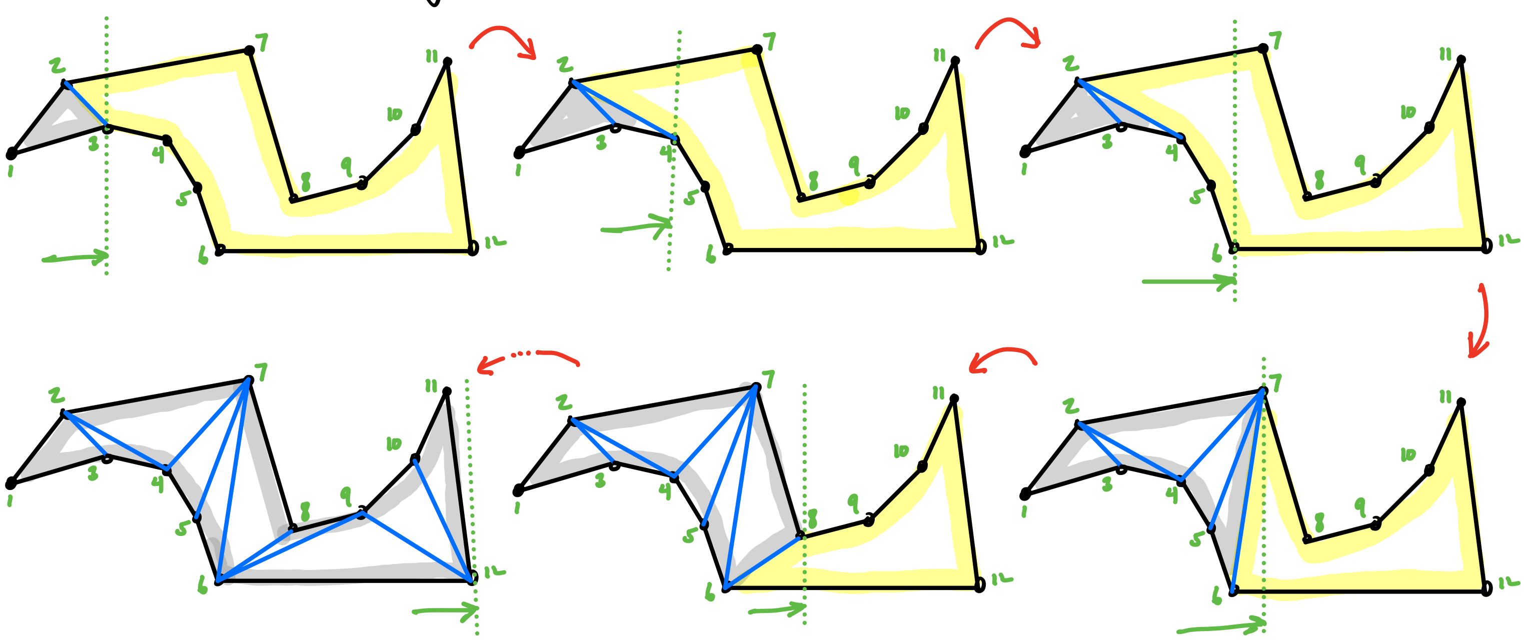
General position: No duplicate x-coords (no vertical edges)

Reflex Vertex: Internal angle $\geq \pi$

Reflex Chain: Sequence of reflex vertices

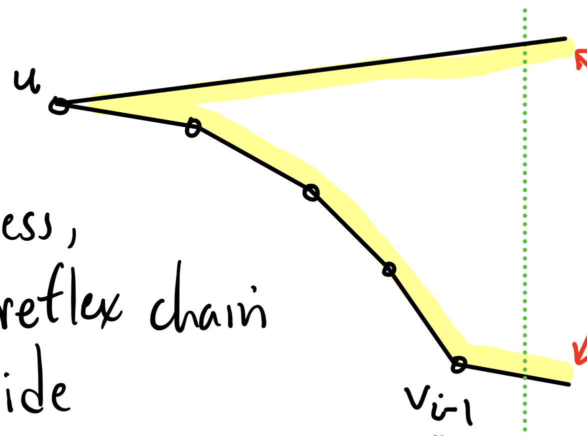


General approach: Sweep from left to right + triangulate as much as we can behind us.



What's the loop invariant?

Lemma: For $i \geq 2$, let v_i be the next vertex to process. The untriangulated region to left of v_i consists of two x-monotone chains starting from a common vertex u. One chain is a single edge, and the other is a reflex chain (of one or more edges).



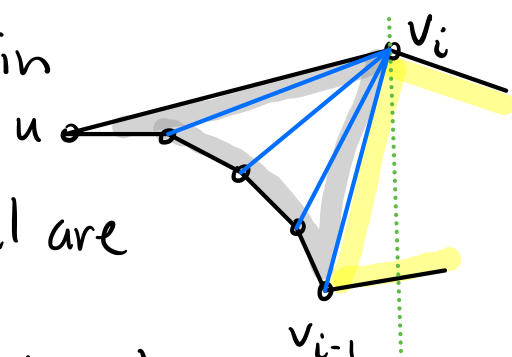
For concreteness, let's assume reflex chain is on lower side

v_i may be on either side

v_{i-1} may or may not be reflex

Case 1: (v_i lies on upper chain)

- add diagonals between v_i and all vertices of the chain



[By monotonicity, all are visible to v_i]

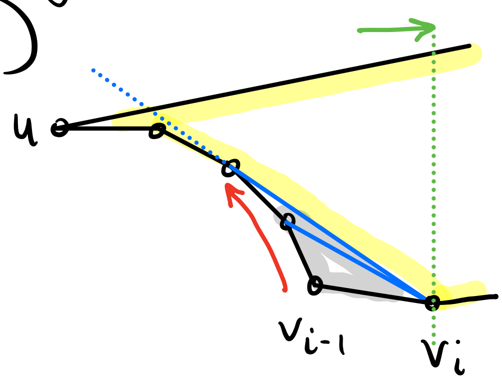
Now $u = v_{i-1}$. Reflex chain has just one edge.

Case 2: (v_i lies on lower chain)

2a: (v_{i-1} is non-reflex)

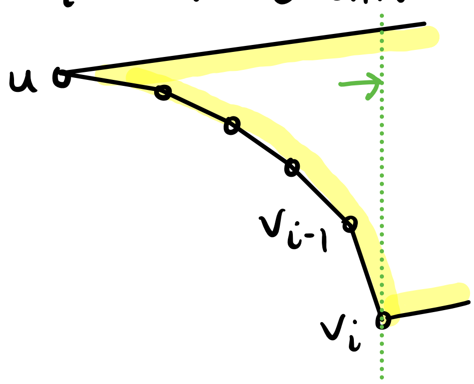
- connect v_i to all visible vertices on chain until hitting point of tangency. (Similar to Graham's scan)

[May go all the way back to u]



2b: (v_{i-1} is reflex)

- Add v_i to the chain



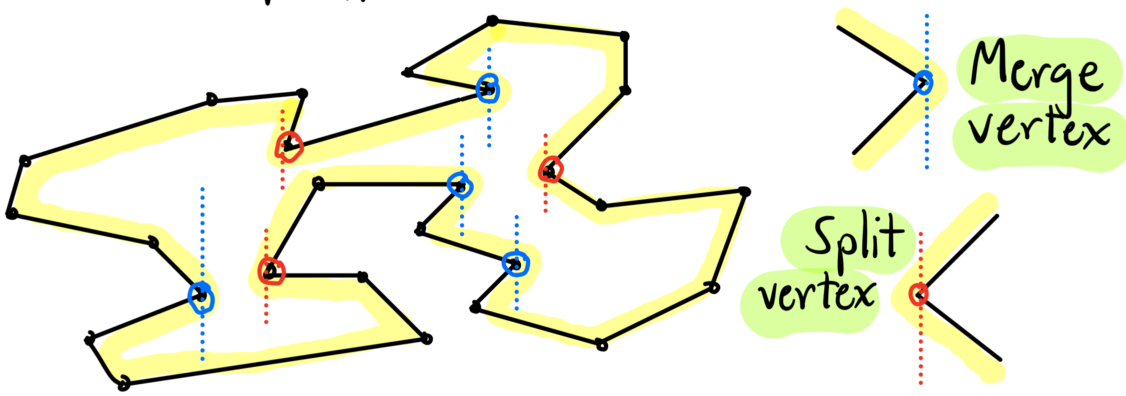
Correctness: Invariant holds after each iteration

Running time: $O(n)$ [As in Graham, once a vertex is removed from the chain, it never reappears)

Monotone Subdivision:

Recall: Add diagonals to create x-monotone

Where? Scan reflex vertex: Reflex vertex where both edges on same side of vertical line.



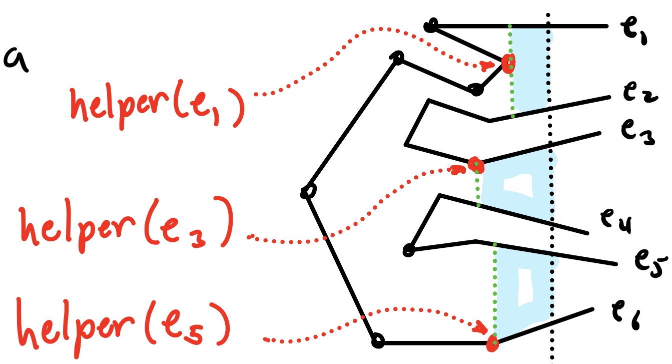
Add a diagonal to right side of each merge
" " " " left " " " split

Plane-sweep Approach:

Need auxilliary info to help with diagonals

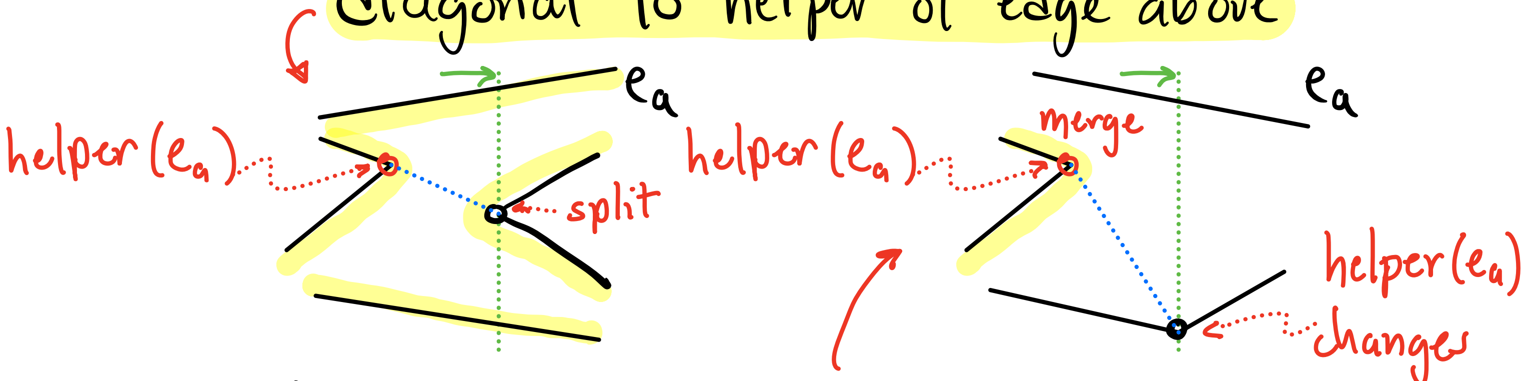
For each edge e_a of sweep line with int($\mathcal{P}$) below:

helper(e_a) = rightmost vertically visible vertex on or below e_a to left of sweep line



Why is the helper helpful?

- when we see a split vertex, we add diagonal to helper of edge above



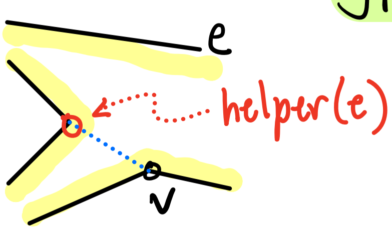
- When we see a merge vertex, it is the helper of edge above + we connect it to next vertex where helper(e_a) changes

Events: Polygon vertices (sorted by x)

Sweep-line status: Edges intersecting the sweep line (ordered dictionary)

Event processing: There are many cases!

Utility:

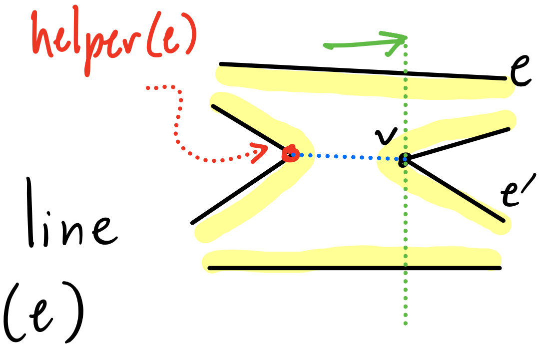


fix-up(v,e):

```
if (helper(e) is a merge vertex)
    add diagonal v to helper(e)
```

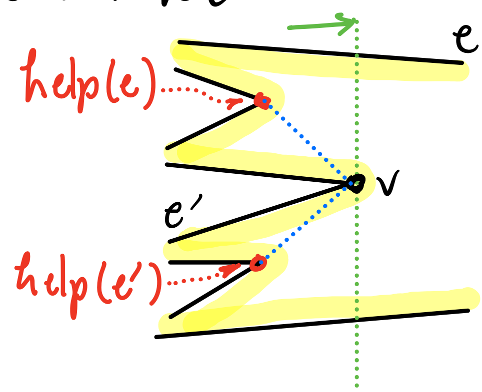
Split Vertex (v):

- e ← edge above v in sweep line
- add diagonal v to helper(e)
- insert edges incident to v into sweep line
- letting e′ be lower, set helper(e′) ← v



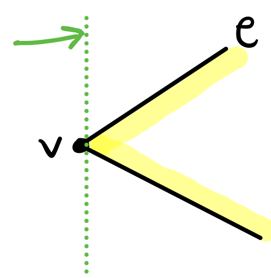
Merge Vertex (v):

- Consider two edges incident to v + let e′ be lower one
- Delete both from sweep line
- Let e be edge above v
- fix-up(v,e) + fix-up(v,e′)



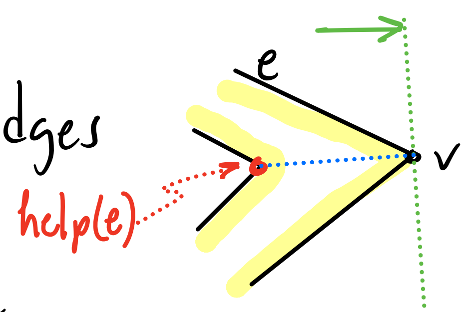
Start vertex (v):

- Insert v's incident edges into sweep line
- Letting e be upper edge, helper(e) ← v



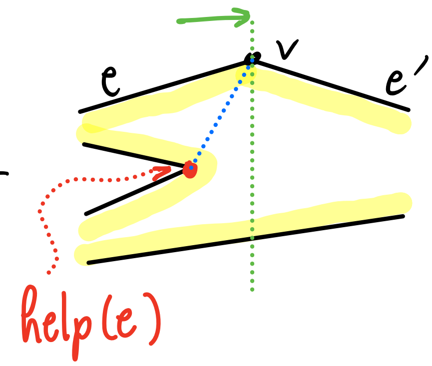
End vertex (v):

- Consider the two incident edges + let e be upper edge
- Delete both from sweep line
- fix-up(v,e)



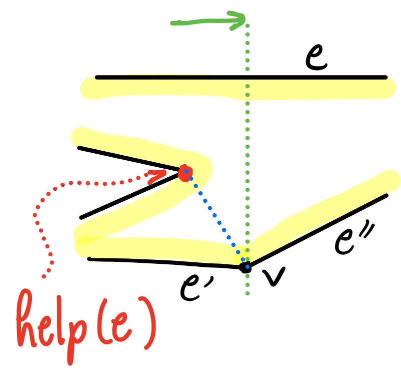
Upper-chain vertex (v):

- Let e be edge to left, e′ to right
- fix-up (v, e)
- Replace e with e′ in sweep line
- helper(e′) ← v



Lower-chain vertex (v):

- Let e be edge above
- fix-up (v, e)
- Let e′ be edge to left, e″ to right
- Replace e′ with e″ in sweep line



CMSC 754 - Computational Geometry
Lecture 6: Halfplane Intersection + Duality

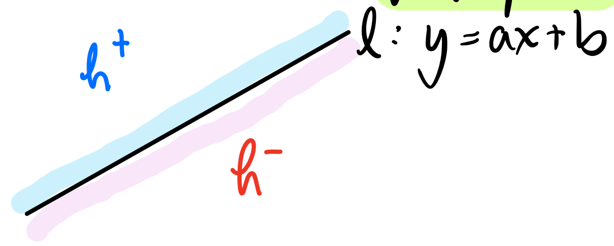
Halfplane Intersection:

Recall, each line in plane defines two **halfspaces**

$\ell: y = ax + b$

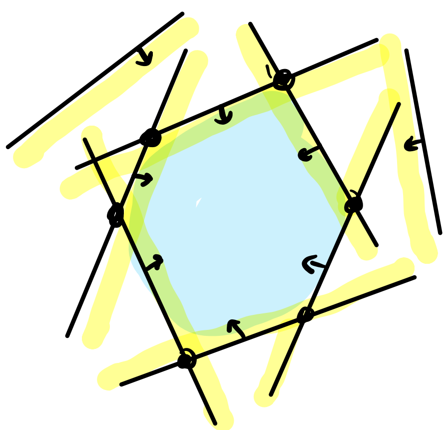
$h^+: y \geq ax + b$

$h^-: y \leq ax + b$

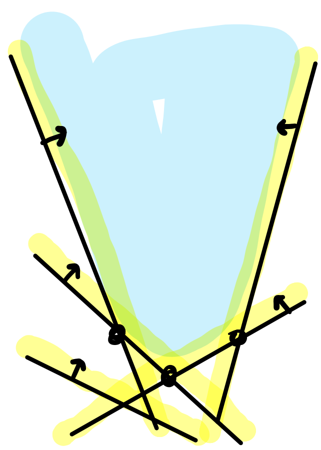


A halfspace is an **(unbounded) convex set**

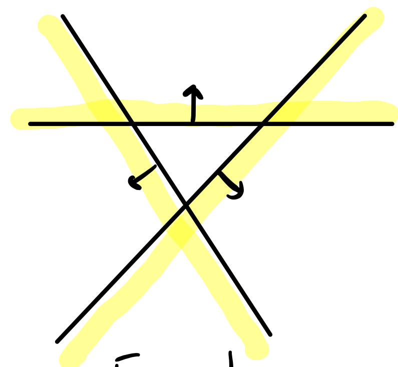
Given a set of halfspaces: $H = \{h_1, \dots, h_n\}$
their **intersection $\bigcap_{i=1}^{n} h_i$** is a (possibly unbounded / possibly empty) **convex polygon**



Bounded



Unbounded



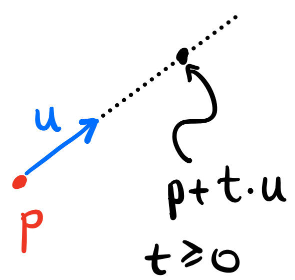
Empty

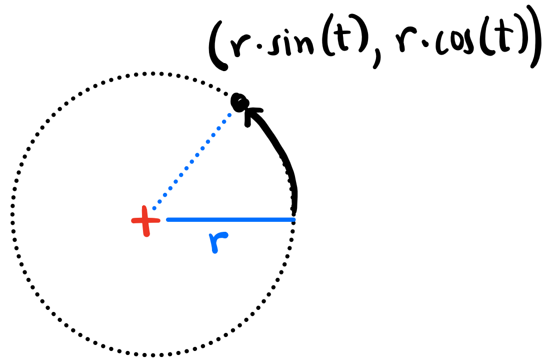
Representing lines (and more):

	$\mathbb{R}^2$ (Line)	$\mathbb{R}^d$ (Hyperplane)
Explicit: $y = f(x)$	$y = ax + b$	$x_d = \sum_{i=1}^{d-1} a_i x_i + b$
Implicit: $f(x,y) = 0$	$f(x,y) = ax + by + c$	$f(x_1, \dots, x_d) = \sum_{i=1}^{d} a_i x_i + b$

Parametric:

$(x(t), y(t))$



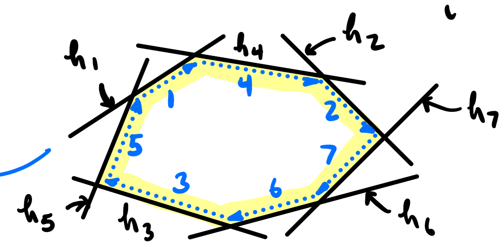


Halfplane Intersection:

Given halfplanes $H = \{h_1, \dots, h_n\}$ construct $\mathcal{H} = \bigcap_i h_i$

Output: Sequence of edges

$\langle 5, 1, 4, 2, 7, 6, 3 \rangle$



Divide and Conquer Algorithm: $O(n \log n)$

Intersect (H) {

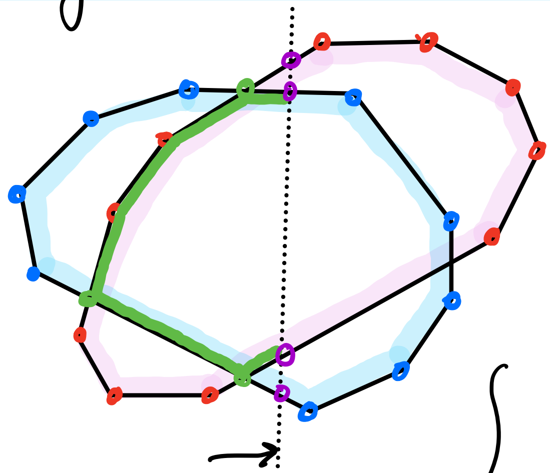
- if($|H| = 1$) return h_1 [single halfspace]
- else

  partition $H \to H_1, H_2$ $\quad |H_i| \le n/2$

  $I_1 \leftarrow$ Intersect (H_1); $I_2 \leftarrow$ Intersect (H_2)

  return merge (I_1, I_2) ← How?

How to merge? Plane sweep



- At most 4 segments hit sweep line
- $\leq n_1 + n_2$ end pt events $n_i = |H_i|$
- $\leq 2(n_1+n_2)$ intersection events
- Boundaries are already sorted

$O(n_1+n_2)$ time

Overall Running Time:

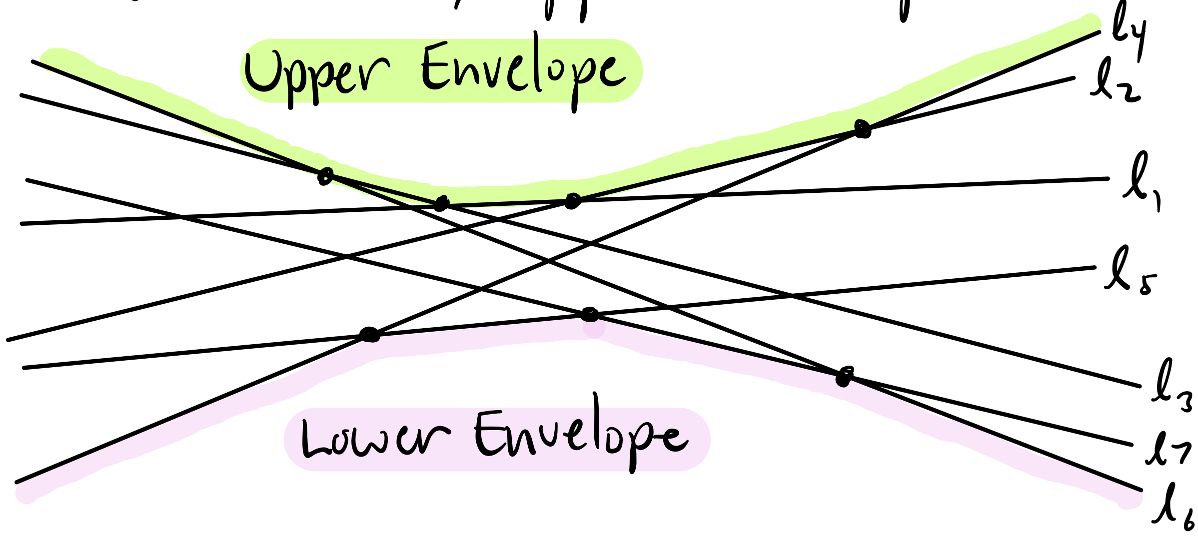
$$T(n) = 2T(n/2) + n$$

2 recursive calls on $n/2$ halfspaces

merge in linear time

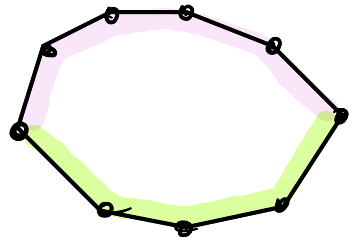
$$= O(n \log n)$$ [see, e.g., CLRS]

Special Case: Lower/Upper Envelopes



Envelopes of lines ~ Hull of points

Related?

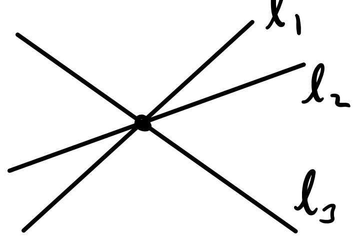


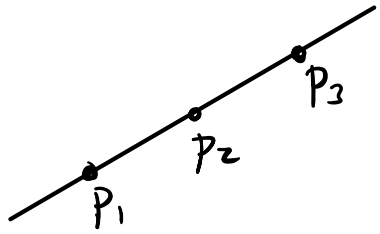
Point-Line Duality

Lines in $\mathbb{R}^2$ are a lot like points:

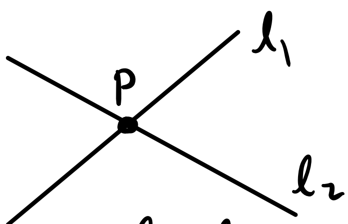
2 degrees of freedom: $y = \underline{a}x + \underline{b}$ $\quad p: (\underline{a}, \underline{b})$

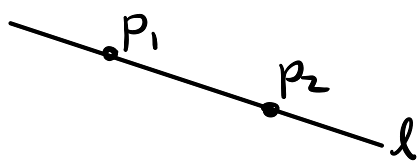
degeneracy:





incidence:





$p = l_1 \wedge l_2$

Two lines meet at a point

$l = p_1 \vee p_2$

Two points join to form a line

Dual Operator:

Given point $p = (a, b)$ $\quad a, b \in \mathbb{R}$

line $l: y = cx - d$ $\quad c, d \in \mathbb{R}$

Dual p^* is the line $y = a \cdot x - b$

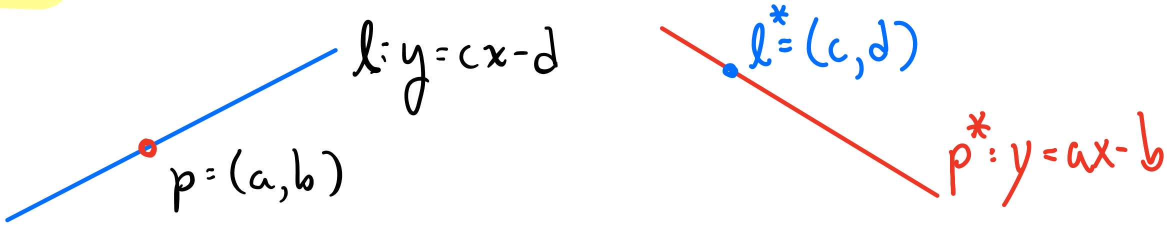
l^* is the point (c, d)

Observations:

Self-inverse: $p^{**} = p \qquad \ell^{**} = \ell$

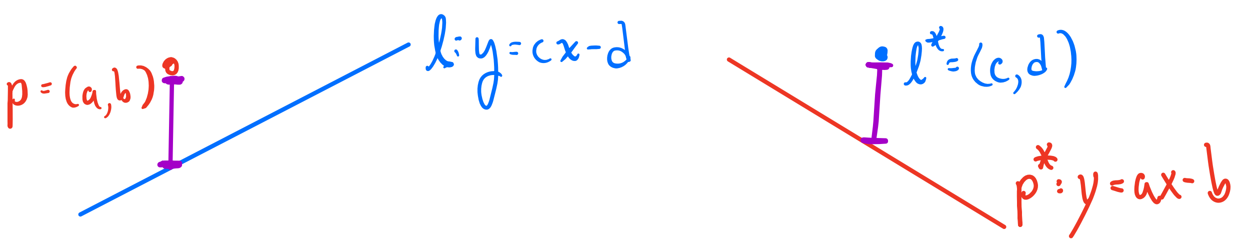
Incidence: p lies on ℓ iff ℓ^* lies on p^*

Proof: $b = c \cdot a - d \iff d = a \cdot c - b$



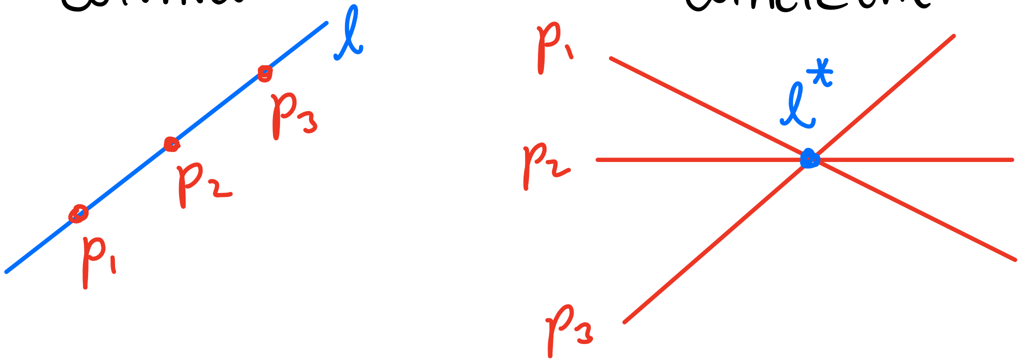
Order reversing: p lies above/below ℓ iff p^* passes below/above ℓ^*

Proof: $b > c \cdot a - d \iff d > a \cdot c - b$



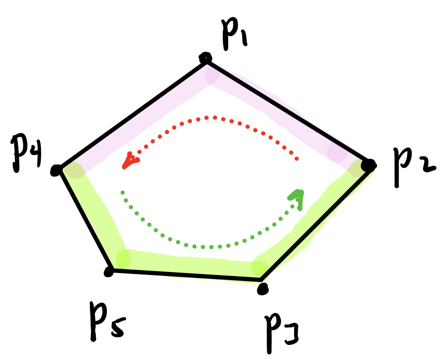
Degeneracy:

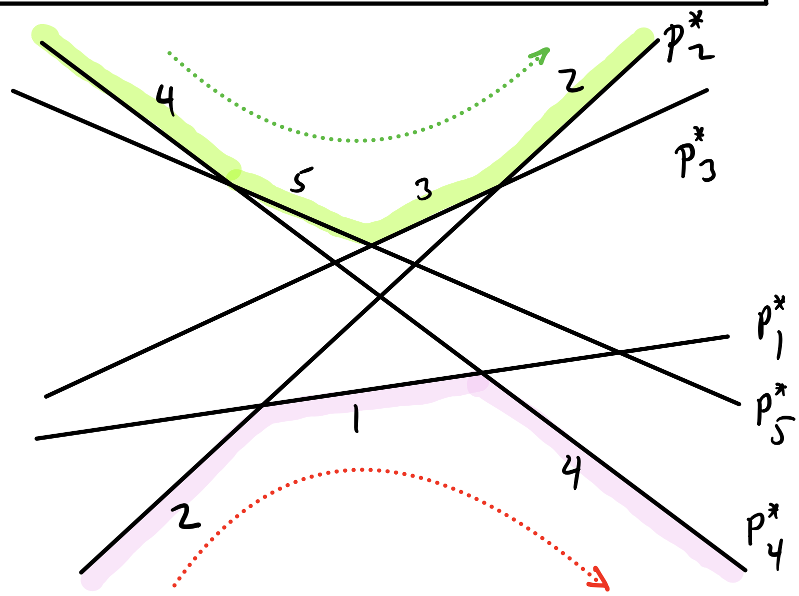
p_1, p_2, p_3 are collinear iff p_1^*, p_2^*, p_3^* are coincident



Hulls and Envelopes:

Lemma: Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$, the CCW order of points on P's upper/lower hull is same as left-right order of segments in P^*'s lower/upper envelope





Proof: (Sketch)

Consider edge $p_i p_j$ on upper hull of conv(P)

Let ℓ be line $\overleftrightarrow{p_i p_j}$ – All pts of P lie on or below ℓ

$\Leftrightarrow$ (order reversal) – All lines of P^* pass on or above point ℓ^*

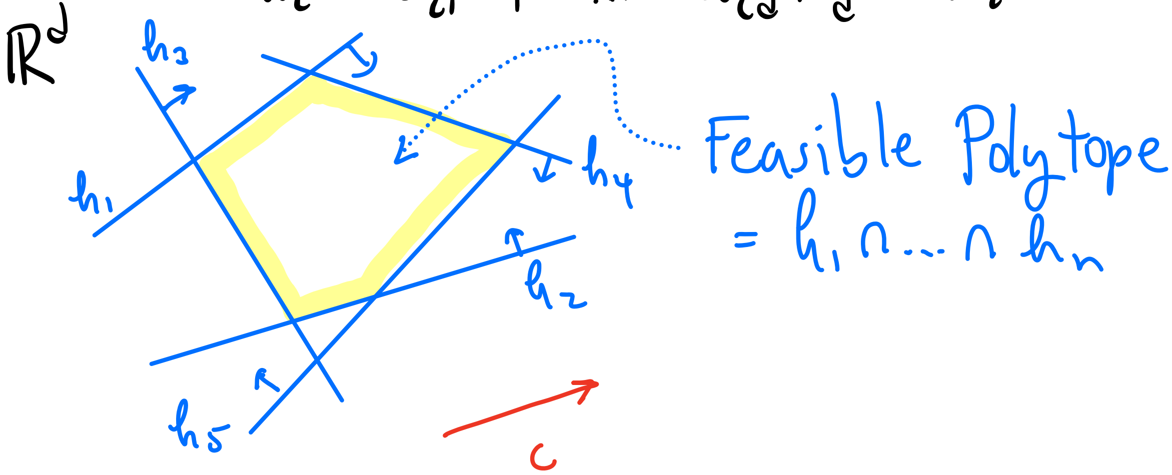
$\Leftrightarrow$ ℓ^* is vertex of lower envelope

CMSC 754 - Computational Geometry
Lecture 7: Linear Programming

Linear Programming (LP):

- Fundamental optimization problem in $\mathbb{R}^d$
- Given a set of n linear constraints (halfspaces) $H = \{h_1, \ldots, h_n\}$

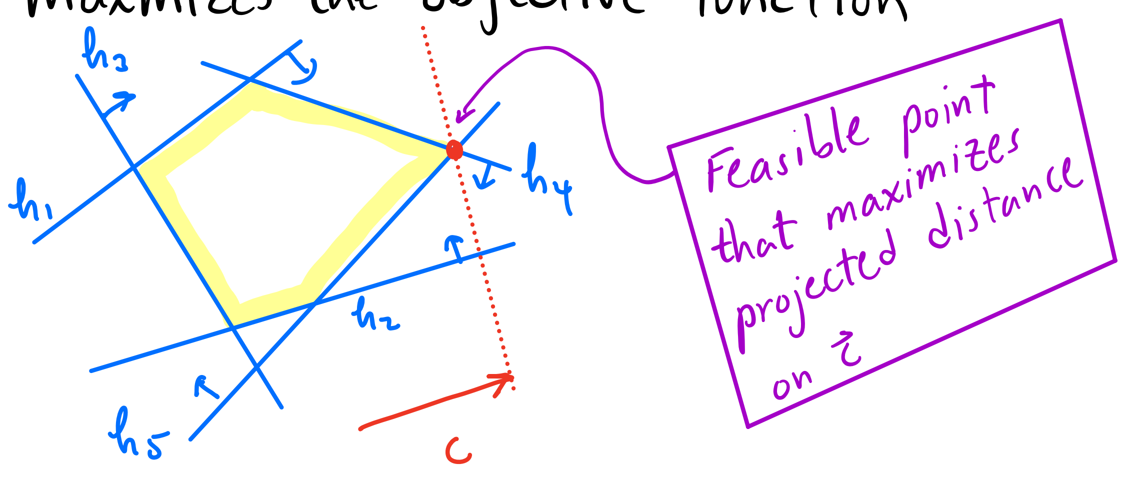
$$h_i : a_{i1} x_1 + \ldots + a_{id} x_d \leq b_i$$



- Given a linear objective function

$$f(\bar{x}) = c_1 x_1 + \ldots + c_d x_d = c^T x$$

LP: Find the vertex of the feasible polytope that maximizes the objective function



Matrix form:

Given $c \in \mathbb{R}^d$ and $n \times d$ matrix A and $b \in \mathbb{R}^n$ find $x \in \mathbb{R}^d$ to:

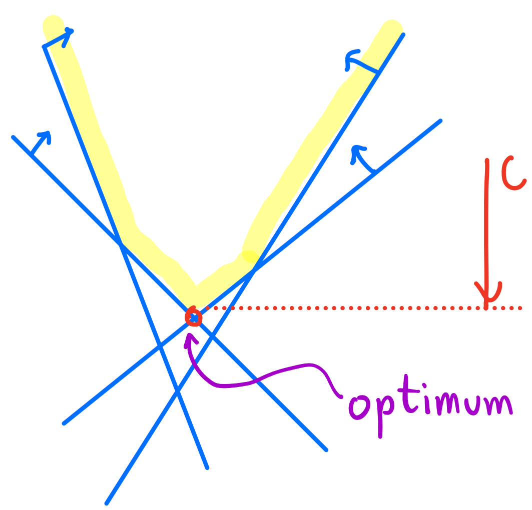
$$\begin{aligned} &\text{maximize: } c^T x \\ &\text{subject to: } Ax \leq b \end{aligned}$$

i^{th} row of A corresponds to h_i

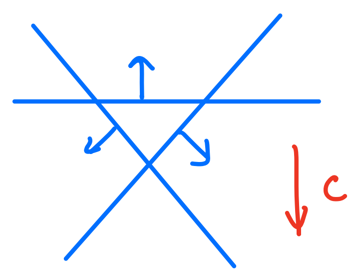
3 Possible Outcomes:

- **Feasible:** An optimal pt exists (gen'l position: a unique vertex of feasible polytope)
- **Infeasible:** No solution because feasible polytope is empty
- **Unbounded:** No (finite) solution because feasible polytope is unbounded in direction of objective fn.

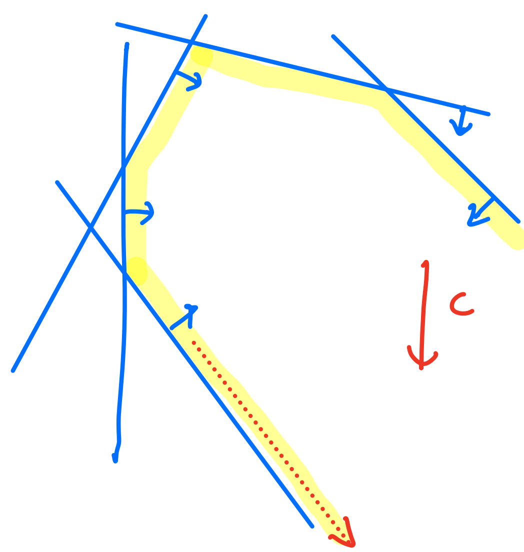
Feasible



Infeasible

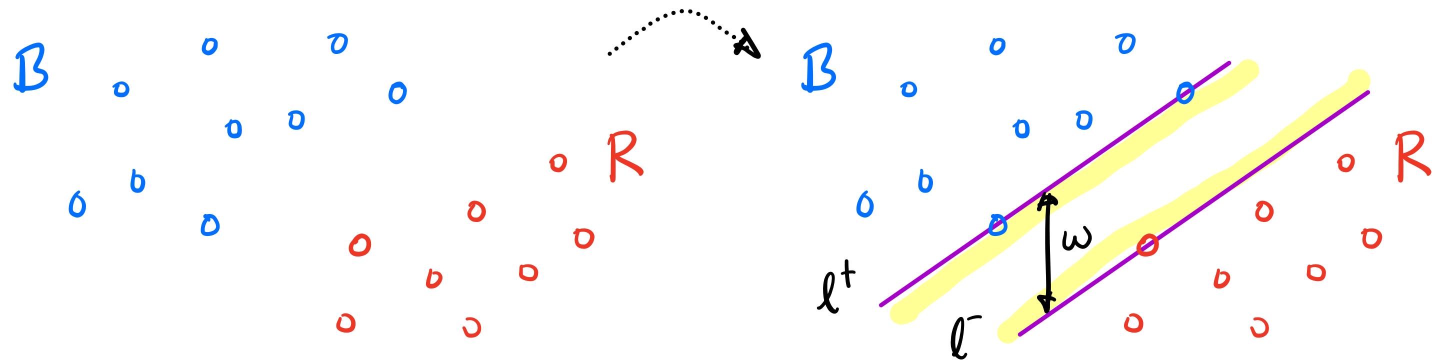


Unbounded



Example:

- Given two point sets B + R in $\mathbb{R}^2$ find lines of max. vertical distance with B above both + R below both



- **Lines**: $\ell^+ : y = e \cdot x + f^+ \quad \ell^- : y = e \cdot x + f^-$

- **Constraints**: $\forall p \in B,\ p_y \geq e \cdot p_x + f^+$ (above ℓ^+)
  $\forall p \in R,\ p_y \leq e \cdot p_x + f^-$ (below ℓ^-)

- **Objective**: maximize $w = f^+ - f^-$

LP in $\mathbb{R}^3$

Standard form: Find (e, f^+, f^-)

to maximize $f^+ - f^- \equiv \overbrace{(0,1,-1)}^{c} \cdot (e, f^+, f^-)$

subject to:

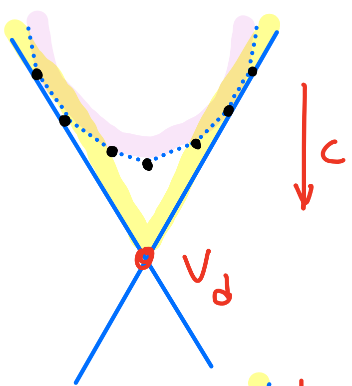
$$p_{ix} \cdot e + 1 \cdot f^+ + 0 \cdot f^- \leq p_{iy}, \ \forall p_i \in B$$
$$-p_{jx} \cdot e + 0 \cdot f^+ - 1 \cdot f^- \leq -p_{jy}, \ \forall p_j \in R$$

LP in constant-dimensional space

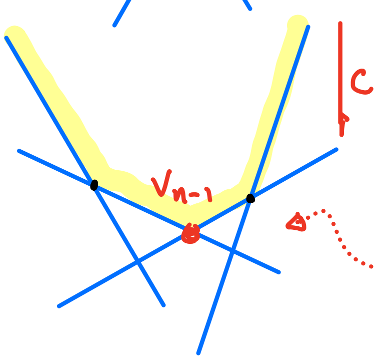
- Assume
  - n is large
  - d is a constant
- We'll present a (randomized) algorithm with (expected) running time $O(d!\,n) = O(n)$

Incremental Approach:

Overview:



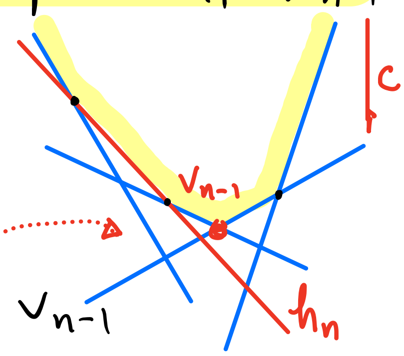
- Find d-halfspaces that define an initial vertex v_d (or report that LP is unbounded)
  - $\to O(dn)$ time (see our text)



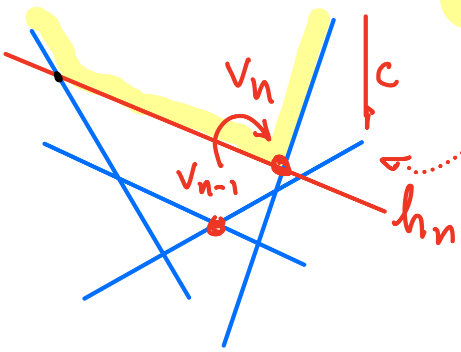
- Remove halfspace h_n and recursively compute LP on $n-1$ halfspaces $h_1 \ldots h_{n-1}$

  If infeasible $\to$ return

  else let v_{n-1} be opt



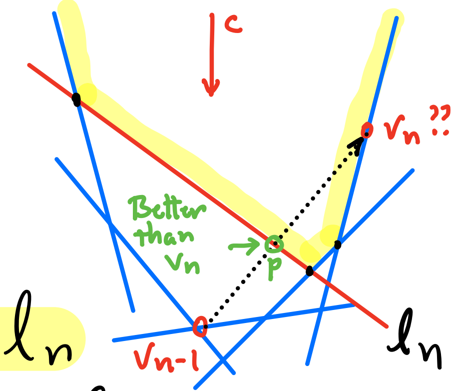
- Add back h_n
  - If ($v_{n-1} \in h_n$) return v_{n-1}
  - else ...



How to update opt. vertex?

Lemma: If $v_{n-1} \notin h_n$ then new opt vertex (v_n) lies on the hyperplane bounding h_n.

Proof: Let ℓ_n be hyperplane bounding h_n. Assume c directed downwards.



v_{n-1} – not feasible $\Rightarrow$ below ℓ_n

v_n – if not on ℓ_n $\Rightarrow$ above ℓ_n

Let $p = \ell_n \cap \overline{v_{n-1} v_n}$

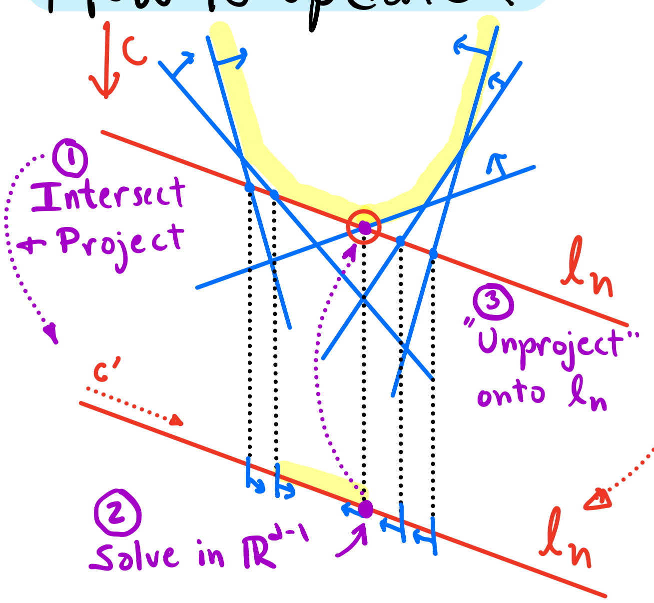
By convexity, $p \in$ feasible polytope

By linearity, obj. function gets progressively worse from $v_{n-1} \rightarrow v_n$

$\Rightarrow$ p is better solution than v_n

✗ contradiction!

How to update?



(See latex notes for details)

① Intersect $h_1, \ldots, h_{n-1}$ with ℓ_n + project $\vec{c}$ [Yields an LP in $\mathbb{R}^{d-1}$ with $n-1$ constraints]

② Solve this $(d-1)$-dim LP recursively (If $d=1$, solve by brute force $O(n)$)

③ "Unproject" solution back onto ℓ_n

Running time? Pretty bad - $O(n^d)$

- Let $W_d(n)$ be worst-case complexity for n halfspaces in dim d
- Recurrence:

$$W_d(n) = W_d(n-1) + d + [dn + W_{d-1}(n-1)]$$

solve LP on $h_1 \dots h_{n-1}$

Test $v_{n-1} \in h_n$

Project onto ℓ_n

Solve LP on ℓ_n

Claim: $W_d(n) = O(n^d)$ ← Too slow!

How to fix this?

Easy! Randomize the choice of h_n

Why?

$$\boxed{W_d(n) = W_d(n-1) + d} + \boxed{dn + W_{d-1}(n-1)}$$

This solves to $O(n)$

Only applies if $v_{n-1} \notin h_n$

This rarely happens!

Randomized Incremental Algorithm

Input: $H = \{h_1, \dots, h_n\}$ constraint halfspaces in $\mathbb{R}^d$

$c \in \mathbb{R}^d$ objective vector

Output: Optimum vertex v or error $\begin{cases} \text{unbounded} \\ \text{or} \\ \text{infeasible} \end{cases}$

(1) If ($d = 1$) solve LP by brute force - $O(n)$

(2) Find initial subset $\{h_1, \dots, h_d\}$ that provide initial optimum v_d (or return "unbounded") - $O(d \cdot n)$ (see text)

(3) Randomly select halfspace from $\{h_{d+1}, \dots, h_n\}$ - call it h_n. Recursively solve LP on remaining $n-1$ halfspaces → Let v_{n-1} be result

(4) If ($v_{n-1} \in h_n$) return v_{n-1} → $O(d)$

(5) else, project $\{h_1, \dots, h_{n-1}\} + c$ onto ℓ_n, → $O(dn)$ the bounding hyperplane for h_n. Solve recursively, letting v_n be result. Return v_n

Expected Case Running Time:

- Running time depends on (random) choice, h_n

- Let $T_d(n)$ be the expected-case running time, over all choices of h_n.

- Let p_n = probability that $v_{n-1} \notin h_n$

- To simplify, assume all halfspaces chosen randomly ($h_1, \dots, h_d$ aren't)

Recurrence:

$$T_d(n) = \begin{cases} 1 & \text{if } n=1 \\ n & \text{if } d=1 \\ T_d(n-1) + d + p_n\left(dn + T_{d-1}(n-1)\right) & \text{o.w.} \end{cases}$$

(3) Recursively compute v_{n-1}

(4) test if $v_{n-1} \in h_n$

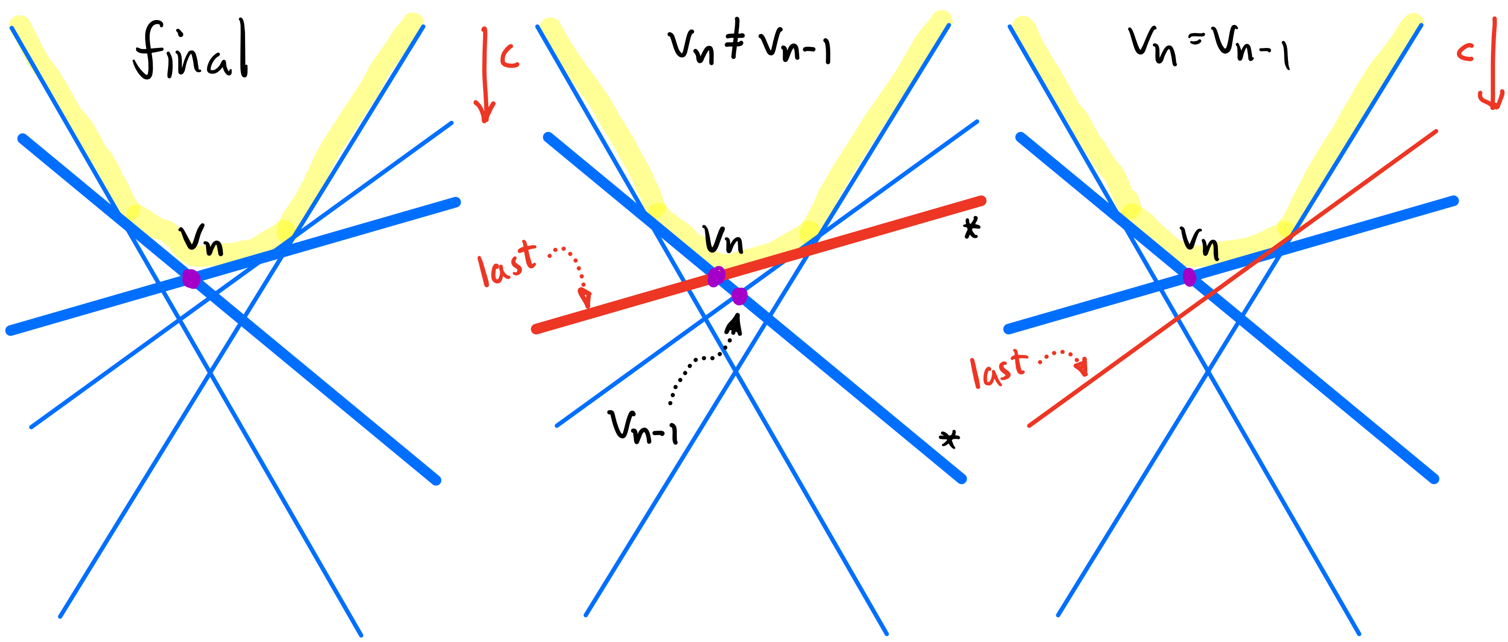
if not

(5) project $h_1, \dots h_{n-1}$ onto ℓ_n

(5) solve $d-1$ dim LP on projections

What is p_n? Backwards Analysis

- Let's consider the final configuration and ask – which halfspace came last and how does its choice affect things?



Obs: The optimum is determined by d halfspaces (assuming gen'l position)

- If h_n is any of these, $v_{n-1} \notin h_n$ + $v_n \neq v_{n-1}$ ☹
- Otherwise, $v_{n-1} \in h_n$ + $v_n = v_{n-1}$ ☺

$\Rightarrow p_n = d/n$ If $n \gg d$, p_n very small + bad case unlikely

Why is it called "backwards"?

- We consider final config. and look backwards to our last random choice

> **Lemma:** $T_d(n) \le \gamma_d\, d!\, n$, where γ_d is a constant depending on dimension

Proof: Induction on $n + d$

simplify

$$T_d(n) = T_d(n-1) + d + p_n\left(dn + T_{d-1}(n)\right)$$

by I.H. + def of p_n

$$\le \gamma_d \cdot d!\,(n-1) + d + \frac{d}{n}\left(d \cdot n + \gamma_{d-1}(d-1)!\, n\right)$$

$$= \gamma_d\, d!\,(n-1) + d + \left(d^2 + \gamma_{d-1}\, d!\right)$$

$$= \gamma_d\, d!\, n + \left(d + d^2 + \gamma_{d-1}\, d! - \gamma_d\, d!\right)$$

want:

$$\leq \gamma_d\, d!\, n$$

Suffices to select γ_d such that

$$d + d^2 + \gamma_{d-1}\, d! - \gamma_d\, d! \leq 0$$

$$\iff d!\,\gamma_d \geq d + d^2 + \gamma_{d-1}\, d!$$

We can satisfy this by setting:

$$\boxed{\begin{aligned} \gamma_1 &\leftarrow 1 \\ \gamma_d &\leftarrow \frac{d+d^2}{d!} + \gamma_{d-1} \end{aligned}}$$

$\Rightarrow \gamma_d$ is a constant depending on dim

$\square$

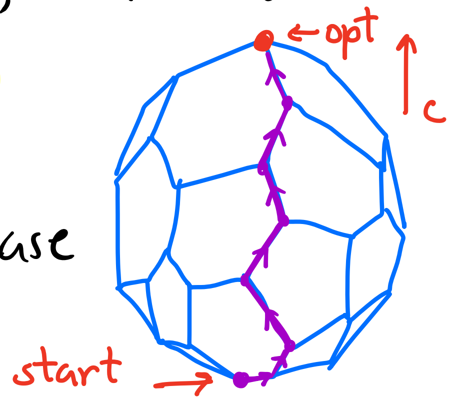
Summary:

- Randomized algorithm for LP
- Expected run time of LP is $O(d!\, n) = O(n)$
  (since we assume d is constant)
- Variation depends on random choices, not input
- (Seidel) Prob of running slower extremely small

A Bit of History (Optional)

1940's: Used in operations research (Econ, Business)
Kantorovich, Dantzig, von Neuman

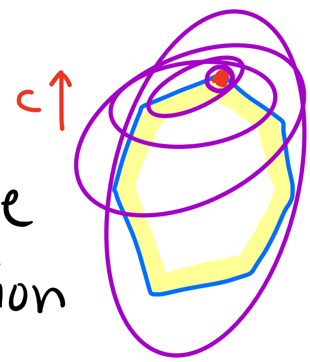
Dantzig (1947) – **Simplex algorithm**



- fast in practice
- exponential in worst case

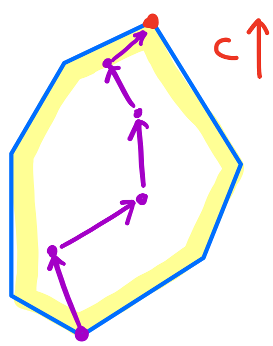
  feasible polytope may have $O(n^{\lfloor d/2 \rfloor})$ vertices
- Karp – Not known to be NP-hard

Khachiyan (1979) – **Ellipsoid Algorithm**



- (Weakly) polynomial time
  ↳ Time depends on precision
- Compute smaller + smaller ellipsoids containing optimum

Karmarkar (1984) – **Interior-Point Methods**



- Move through polytope's interior
- (Weakly) polynomial
- Practical

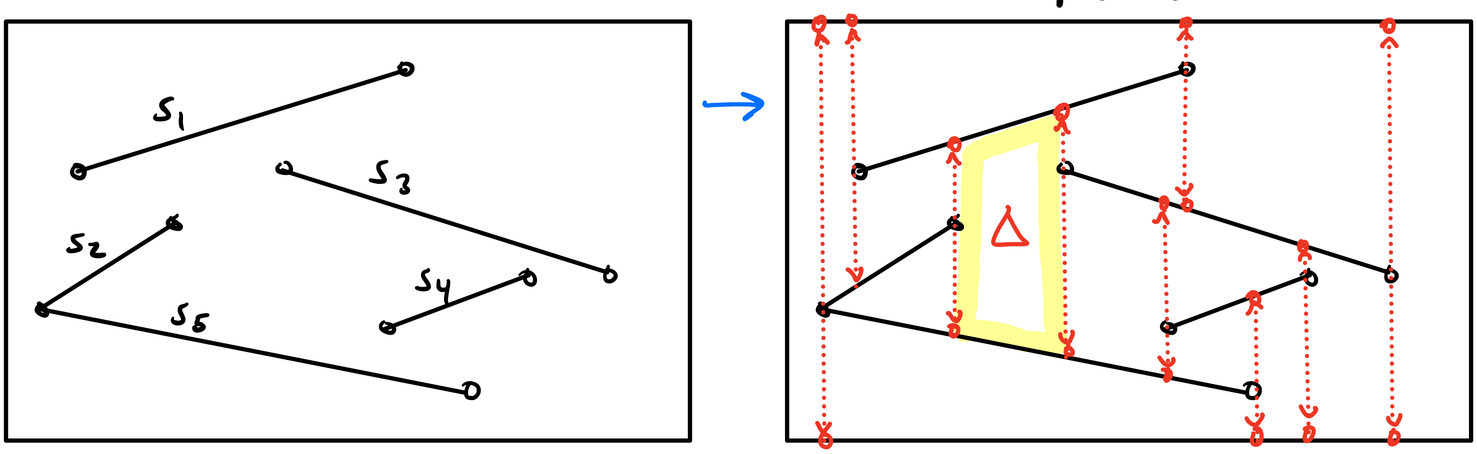
Open – Is there a poly. time (purely combinatorial) algorithm?

CMSC 754 - Computational Geometry
Lecture 8 - Trapezoidal Maps

Trapezoidal Maps:

- Given a set $S = \{s_1, \ldots, s_n\}$ of line segments in $\mathbb{R}^2$, which we assume do not intersect (except at their endpts)
- General position: No duplicate x-coords + no vertical segments
- Enclose in large bounding rectangle
- Shoot a bullet path vertically above + below each endpt until it hits something

S $\qquad\qquad\qquad\qquad$ $T(S)$

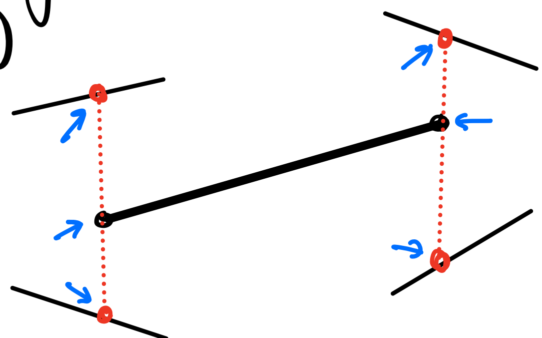


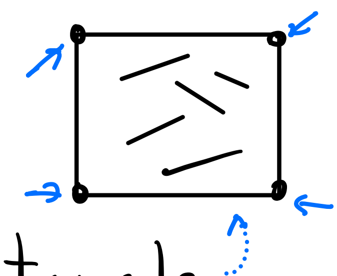
- This subdivides the rectangle into trapezoids (degenerating possibly to triangles)

Lemma: If $|S| = n$, $T(S)$ has $\leq 6n+4$ vertices and $\leq 3n+1$ trapezoids

Proof:

- Each segment contributes 6 vertices to $T(S)$

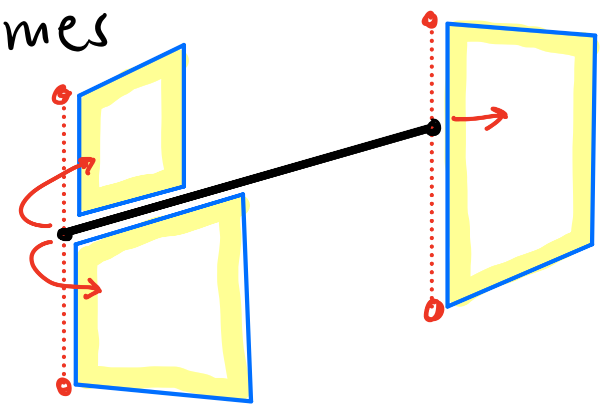


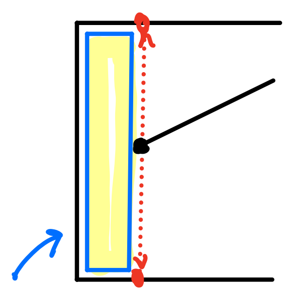


Plus 4 for the bounding rectangle

$\Rightarrow 6n+4$ vertices

- Charge each trapezoid to vertex on its left side. Each segment is charged 3 times

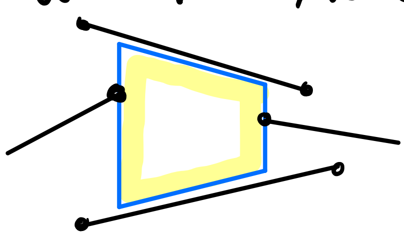




$\Rightarrow 3n + 1$ for leftmost trap.

$\square$

Obs: Each trapezoid owes its existence to ≤ 4 segments



Construction:

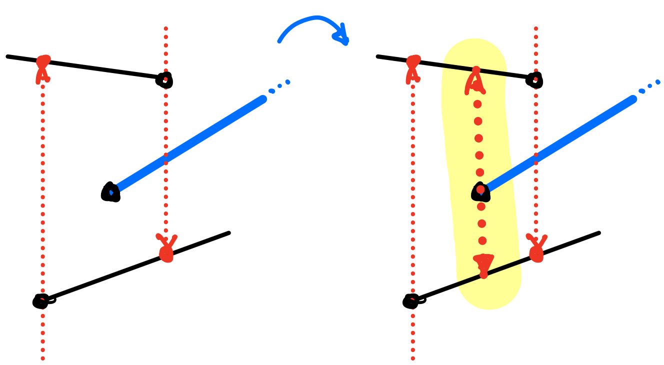
- Plane sweep - $O(n \log n)$ [exercise]
- Randomized incremental - $O(n \log n)$ [this lect]

Incremental Construction:

Random order

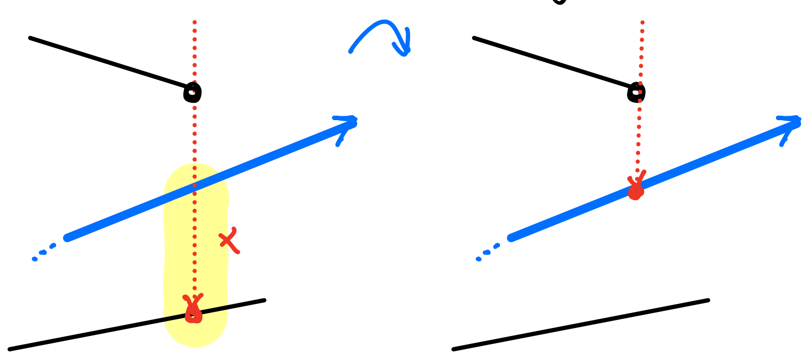
- Add segments one-by-one
- Update the map after each insertion
- Two types of updates:
  - End pt of new segment
    - shoot bullet paths up + down
  - Crash through a vertical wall
    - trim the wall back

End point:



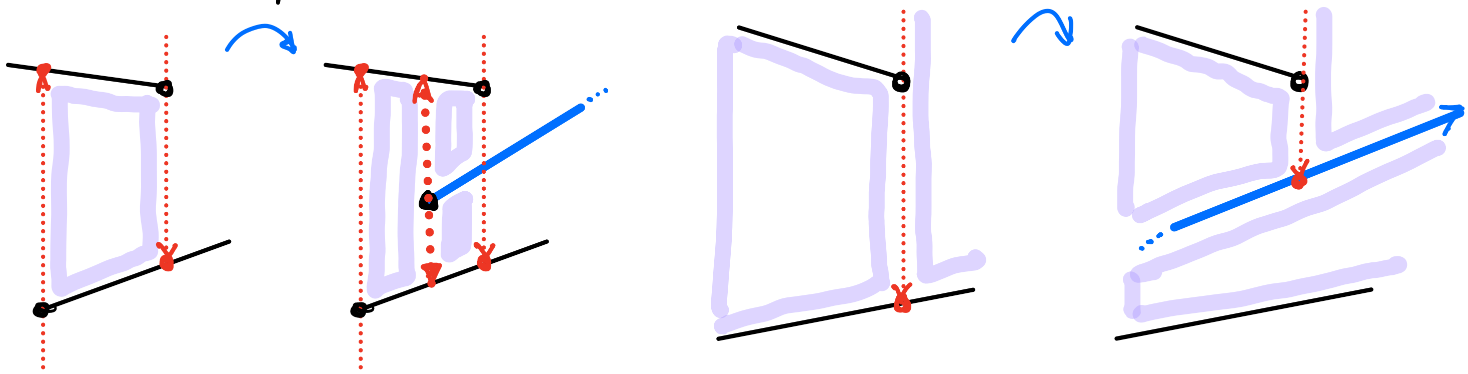
Find the trapezoid containing this end point, and add vertical segments to top + bottom

Crash through wall:



Determine whether the shooting vertex is above or below, and trim away the excess

These updates implicitly generate new trapezoids + destroy old ones



Running time: to insert segment s_i, $i=1,\dots,n$

- Find trapezoid containing s_i's left endpt } $O(\log n)$ (next lect.)

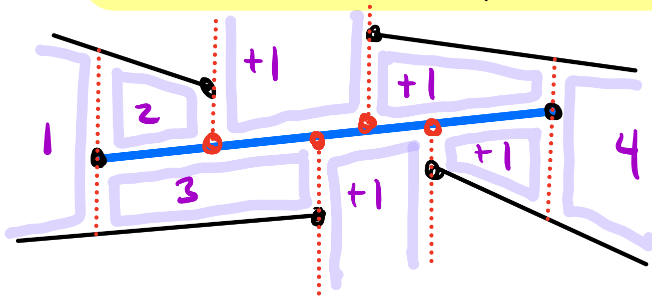
- Trace segment through trapezoids + update } Depends! $O(1) \dots O(n)$

> **Lemma:** For $1 \le i \le n$, let k_i denote the number of new trapezoids created by insertion of i^{th} segment. Ignoring the time to locate the left endpt, the insert time is $O(k_i)$

(Note: k_i is a random variable, depending on insertion order $O(1) \dots O(n)$)

Proof: Let w_i denote num. of walls hit.

$k_i = 4 + w_i$



Insert time ~ $O(2 + 2 + w_i) = O(k_i)$

Bullets for left end pt

Bullets for right

Trim walls hit

$\square$

Overall run time: (Ignoring endpt location)

Worst-case: Adding segment i can create $O(i)$ new trapezoids

$$\Rightarrow W(n) = \sum_{i=1}^{n} i = O(n^2)$$

Expected-case: We will show that if segs are inserted in random order, $E(k_i) = O(1)$

Wow - This does not depend on i!

$$\Rightarrow \text{Exp. time} = \sum_{i=1}^{n} E(k_i) \leq n \cdot O(1) = O(n)$$

(ignoring left endpt location)

Lemma: Assuming segments are inserted in random order, $E(k_i)$ (the expected number of new trapezoids with i^{th} insert) is $O(1)$.

$\mathcal{T}_i$ does not depend on insert order

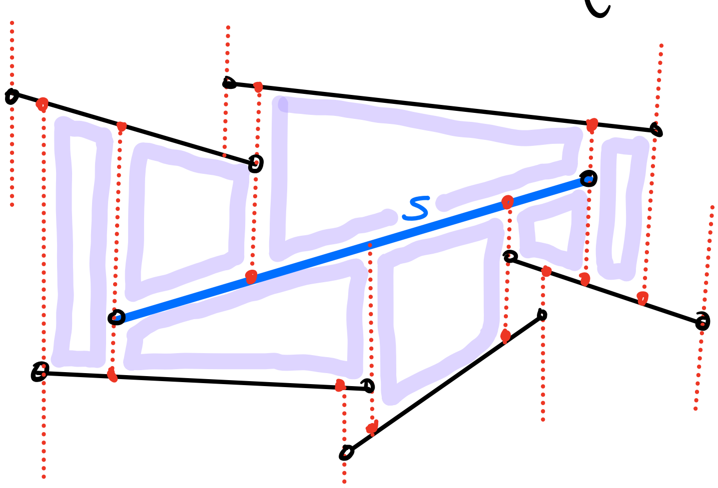
Proof: (Backwards analysis)

- Let $\mathcal{T}_i$ = trapezoidal map after $S_i = \{s_1, s_2, \ldots, s_i\}$
- Each seg. is equally likely to be last

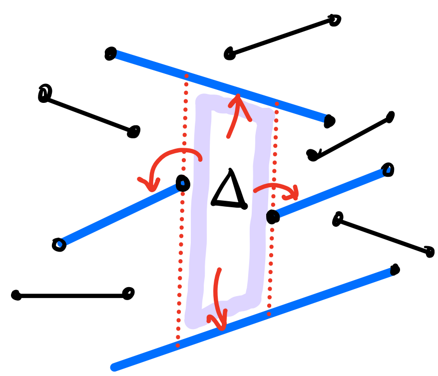
$$\text{Prob}(s_i \text{ is last inserted}) = 1/i$$

- Given any trapezoid $\Delta \in \mathcal{T}_i$ and any segment $s \in \{s_1, \ldots, s_n\}$ we say Δ depends on s if Δ would have been created if s was inserted last.

$$\delta(\Delta, s) = \begin{cases} 1 & \text{if } \Delta \text{ depends on } s \\ 0 & \text{o.w.} \end{cases}$$



Trapezoids that depend on s



Segments on which Δ depends

Note: $\delta(\Delta, s)$ does not depend on insertion order

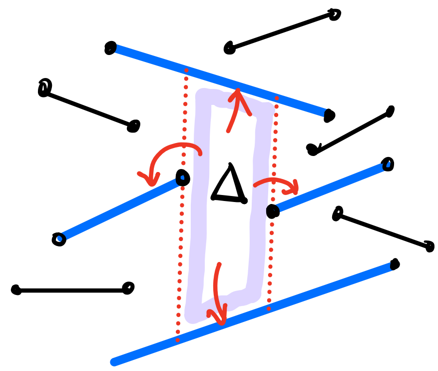
$$E(k_i) = \sum_{s \in S_i} \text{Prob}(s \text{ inserted last}) \cdot \begin{pmatrix}\text{Num. of traps} \\ \text{that depend on } s\end{pmatrix}$$

$$= \sum_{s \in S_i} (1/i) \sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s)$$

(Reverse sum. order)

$$= 1/i \sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s)$$

$$= 1/i \sum_{\Delta \in \mathcal{T}_i} \boxed{\sum_{s \in S_i} \delta(\Delta, s)}$$



$$\leq 1/i \sum_{\Delta \in \mathcal{T}_i} 4$$

Δ depends on at most 4 segs

$$= 4/i \cdot (\text{No. of trapezoids in } \mathcal{T}_i)$$

$$\leq 4/i\,(3i+1) \quad \text{[By earlier lemma]}$$

$$= 12 + 4/i = O(1) \qquad \square$$

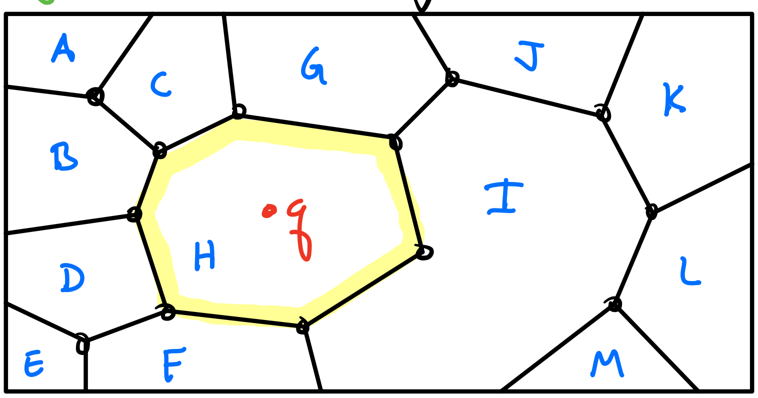
Summary:

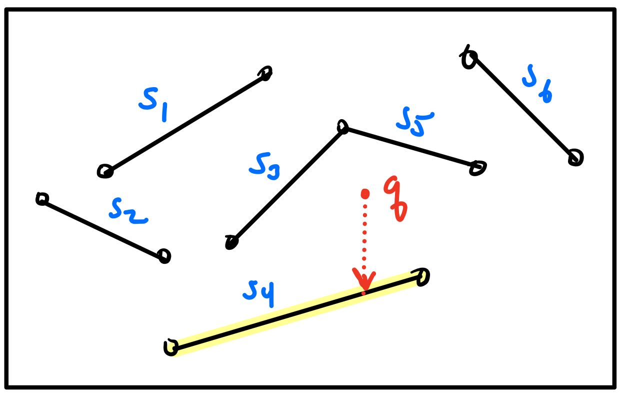
- Shown that if segs. inserted in random order, total no. of updates $O(n)$
- Next: How to locate left endpts.

CMSC 754 - Computational Geometry
Lecture 9: Planar Point Location (via Trap. Maps)

Planar Point Location:

Given a subdivision of the plane (cell complex), build a data structure so that for any query pt, can find the cell containing it.

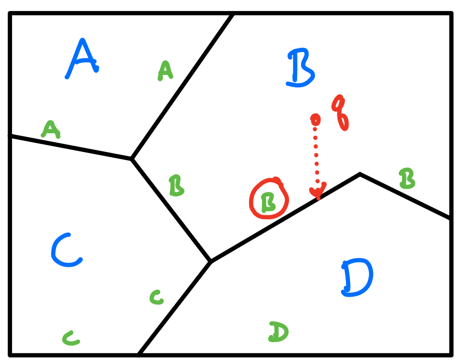




Vertical Ray Shooting:

Given a set of disjoint line segments in the plane, build a data structure s.t. given any query pt q, can report the segment immediately below.

Ray Shooting ⇒ Point Location



Label each segment with region just above

Data structure for vertical ray shooting:

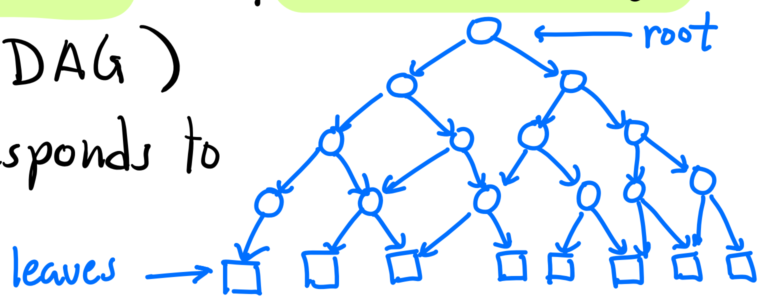
Approach: Build trapezoidal map + ray shooting structure simultaneously

$S = \{s_1, \ldots, s_n\}$ Randomly permuted $\rightarrow \mathcal{T}(S)$
$S_i = \{s_1, \ldots, s_i\}$ $\rightarrow$ Partial map $\mathcal{T}(S_i) = \mathcal{T}_i$

Recall: In expectation, each insertion results in $O(1)$ changes to structure.

Overview:

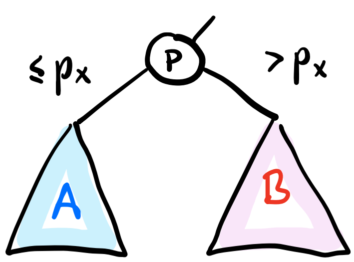
- Rooted binary tree with shared subtrees (a rooted DAG)
- Each leaf corresponds to a trapezoid

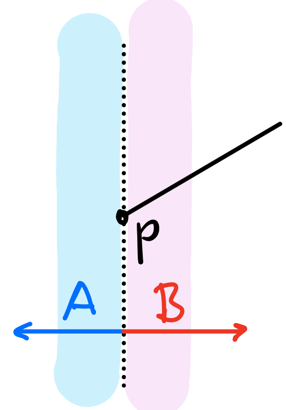


- Each trapezoid occurs exactly once as leaf

- Internal nodes – two types

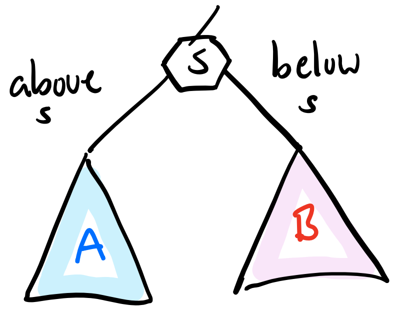
x-Node:
Labeled with an endpt p

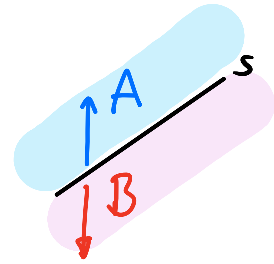




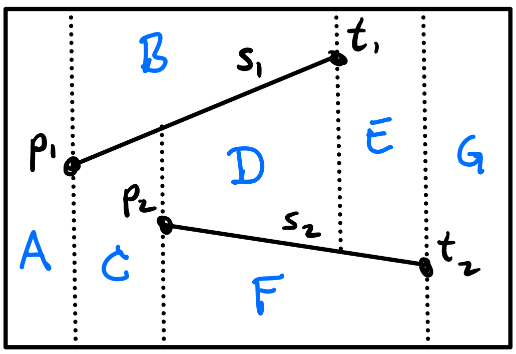
y-Node:

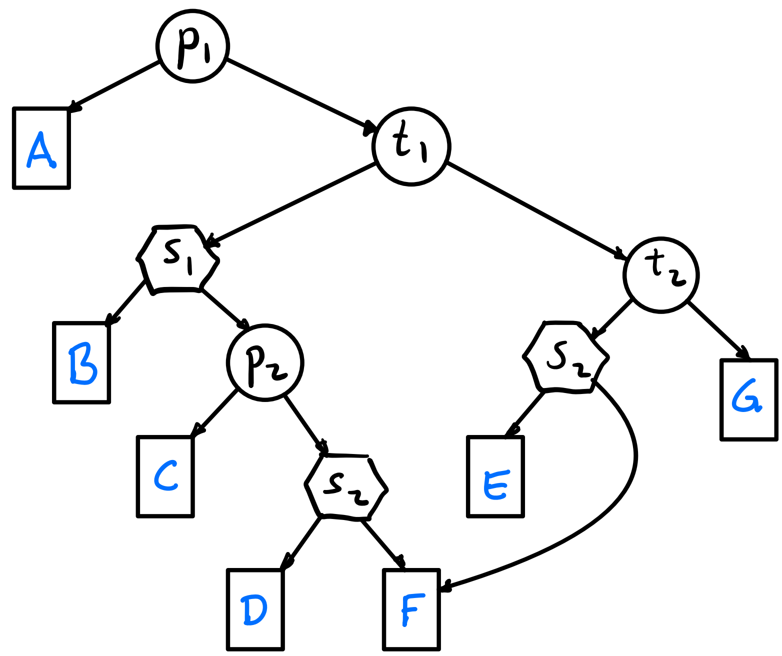
Labeled with a segment s



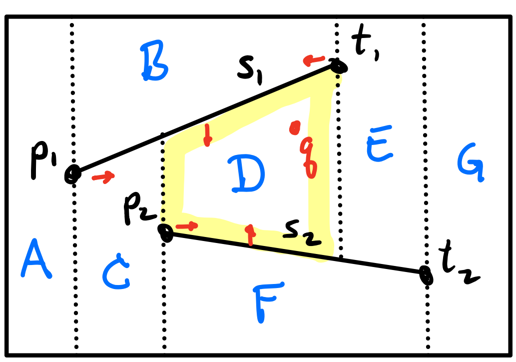


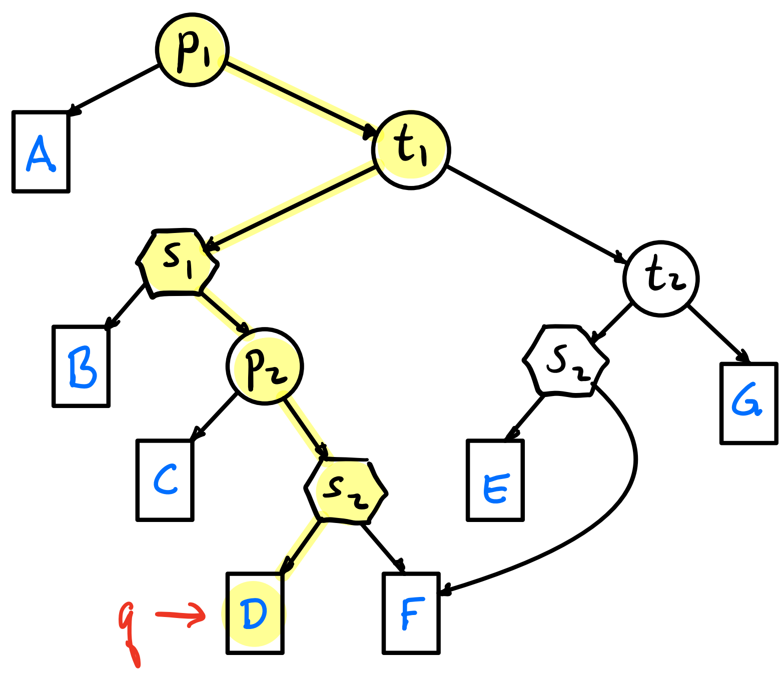
Example:





Query processing:



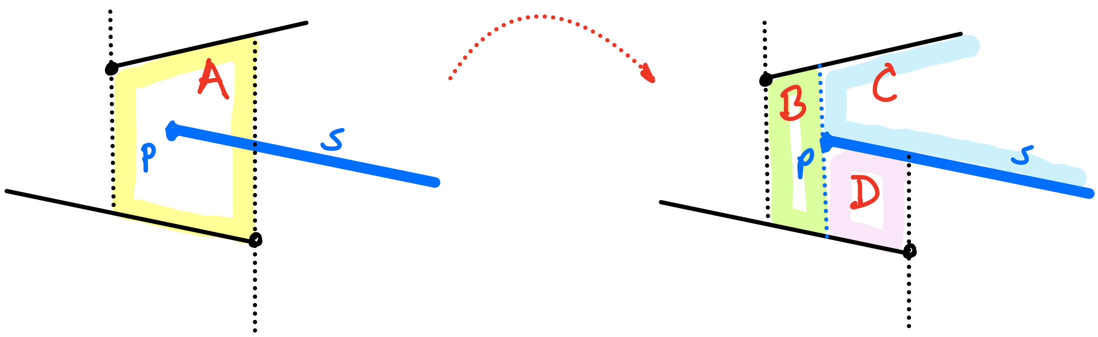


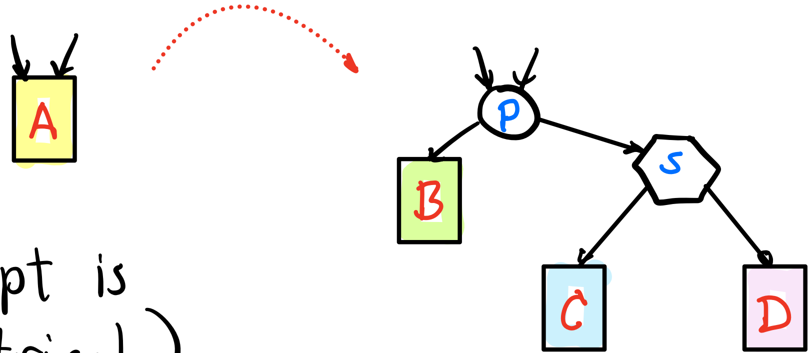
Incremental Construction:

- As segments are added: $s_1, s_2, \dots, s_i$
  we build structure for $\mathcal{T}(S_1), \mathcal{T}(S_2) \dots \mathcal{T}(S_i)$

- Update process:
  - Each added segment causes some trapezoids to go away + others created
  - We replace old leaves with new structures
  - By sharing, only one leaf per trapezoid

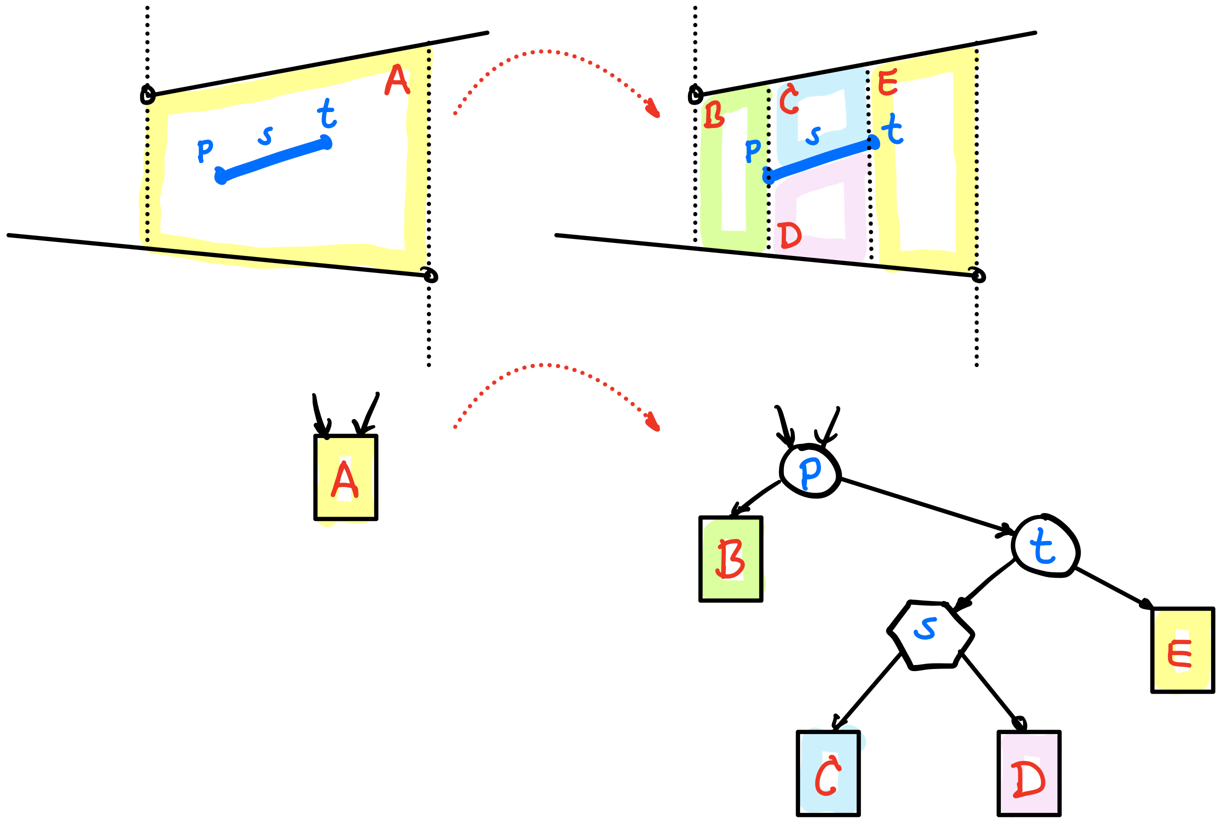
1: Single endpt in trapezoid (left or right):



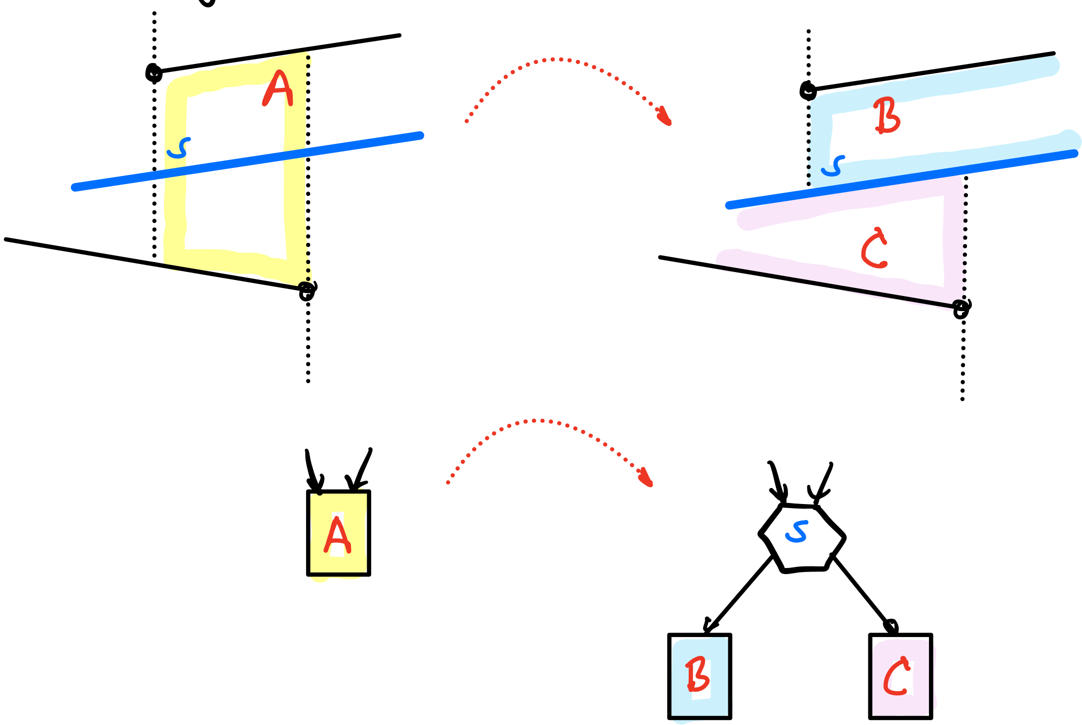


(Right endpt is symmetrical)

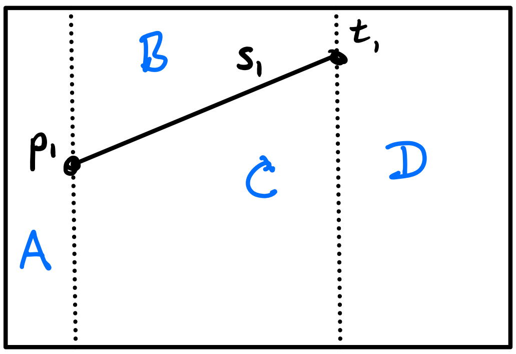
2: Two segment endpts in same trapezoid

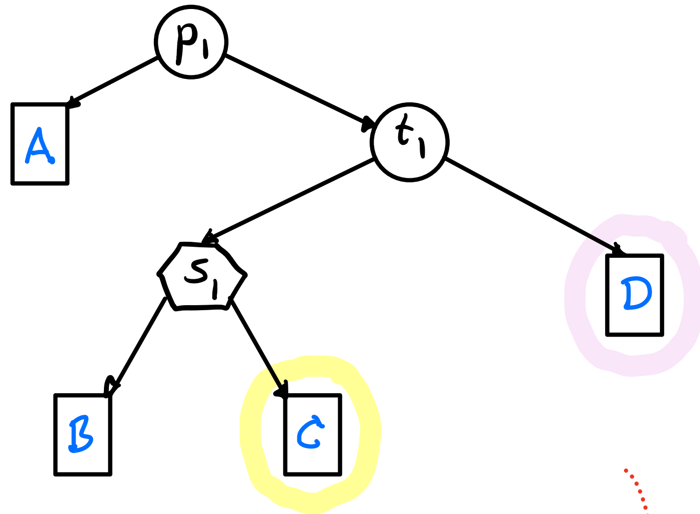


3: No segment endpt in trapezoid

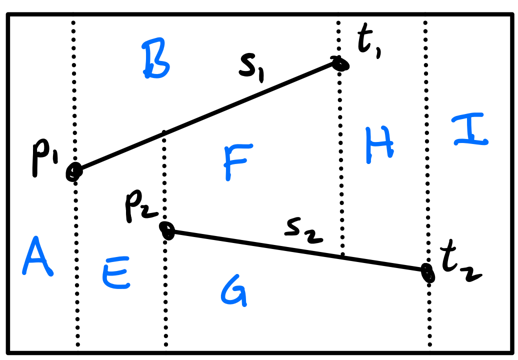


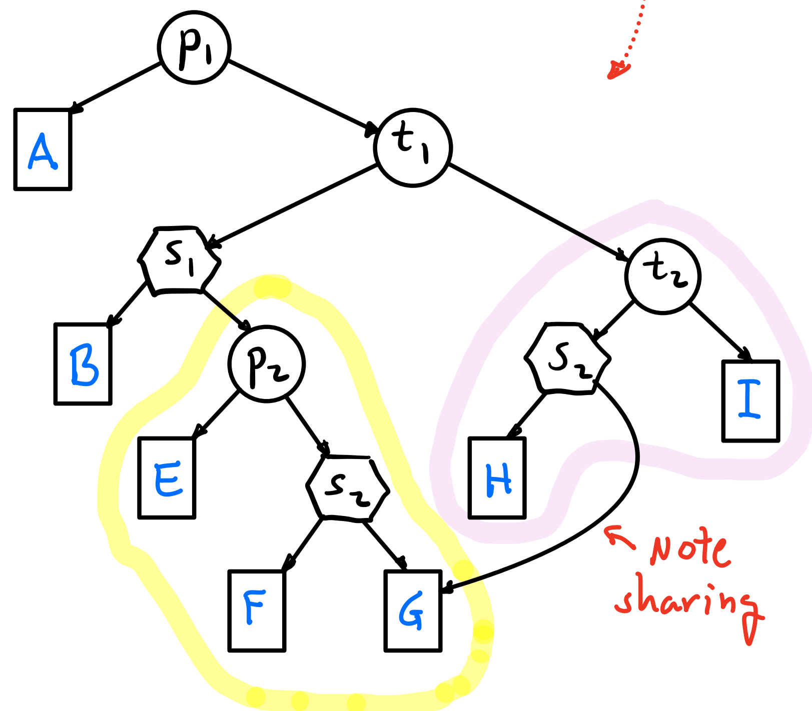
Example:





insert $s_2 = \overline{p_2 t_2}$





Analysis:

Will show if segs are inserted in random order, expected space is $O(n)$ + expected search time for any fixed query pt is $O(\log n)$

Thm: The expected case space is $O(n)$

Proof: Last lecture we showed that expected no. of changes is $O(1)$ per seg $\Rightarrow$ total changes $O(n)$

Number of new nodes $\sim$ number of changes
$\Rightarrow$ final expected size is $O(n)$

Thm: Given a fixed query pt $q \in \mathbb{R}^2$, the expected search depth for q is $O(\log n)$

[Huh? Does this imply that depth of search tree is $O(\log n)$ in expectation?
No – But see our text for a proof of this.]

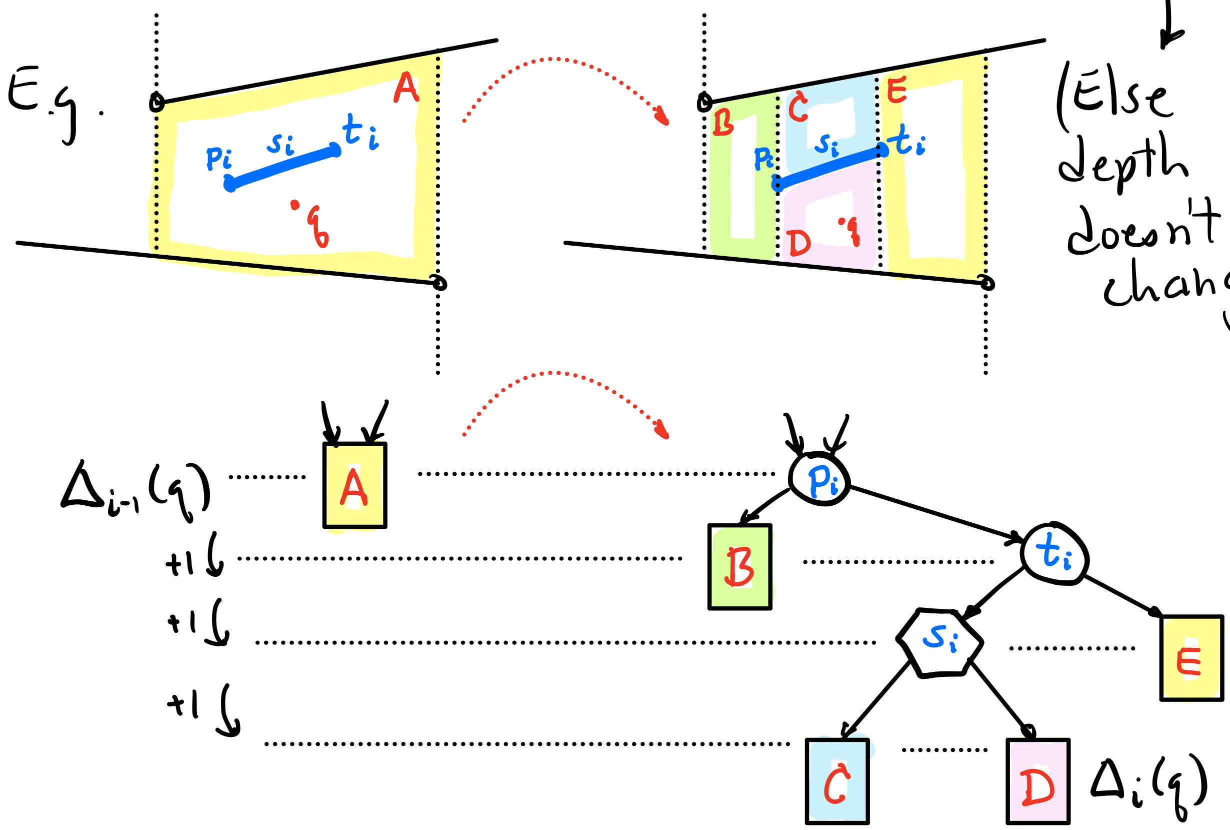
Proof:

- Let q be any fixed query pt.
- Let $\Delta_i(q)$ be the trapezoid containing q after the insertion of s_i $(1 \leq i \leq n)$
  - Note: Sometimes $\Delta_i(q) = \Delta_{i-1}(q)$
    (s_i had no impact)
  - What if $\Delta_i(q) \neq \Delta_{i-1}(q)$?

- For $1 \le i \le n$, let $X_i(q) = \begin{cases} 1 & \text{if } \Delta_i(q) \ne \Delta_{i-1}(q) \\ 0 & \text{o.w.} \end{cases}$

- If $X_i(q) = 1$, $\text{depth}(\Delta_i) \le 3 + \text{depth}(\Delta_{i-1})$

(Else depth doesn't change)



Let $D(q)$ the expected depth of q's trapezoid in the final structure.

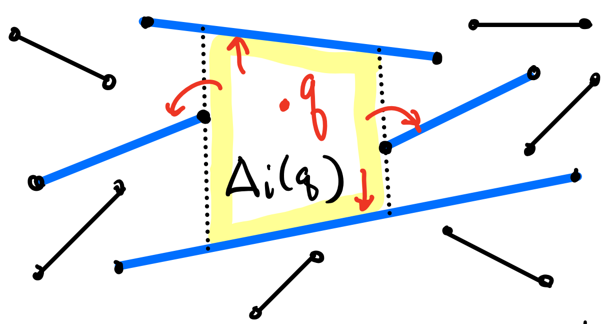
$$D(q) \le 3 \sum_{i=1}^{n} E(X_i(q))$$

$$= 3 \sum_{i=1}^{n} \text{Prob}(\Delta_i(q) \ne \Delta_{i-1}(q))$$

- We assert that $\text{Prob}(\Delta_i(q) \neq \Delta_{i-1}(q)) \leq 4/i$

  - Backwards analysis:
    - Each of the existing i segs is equally likely to be last (prob $= 1/i$)

    - $\Delta_i(q) \neq \Delta_{i-1}(q)$ iff last segment is one of the 4 segments incident to $\Delta_i(q)$



$\Rightarrow \text{Prob}(\Delta_i(q) \neq \Delta_{i-1}(q)) \leq 4/i$

- Substituting: Expected depth of q's trapezoid

$$D(q) \leq 3\sum_{i=1}^{n} E(X_i(q)) = 3\sum_{i=1}^{n} \text{Prob}(\Delta_i \neq \Delta_{i-1})$$

$$\leq 3 \cdot \sum_{i=1}^{n} 4/i = 12 \sum_{i=1}^{n} 1/i \quad \text{(Harmonic series)}$$

$$\cong 12 \ln n = O(\log n) \qquad \square$$

Summary:

- Last time we showed that randomized incremental alg. took $O(1)$ time in expectation per segment, ignoring time to locate left end pt.

- Today, we presented a data structure with query time $O(\log n)$ for pt location

$\Rightarrow$ Total expected construction time is:

$$T(n) = \sum_{i=1}^{n} \Big(\underbrace{(\log i)}_{\text{locate left end pt}} + \underbrace{1}_{\text{update structure}} \Big)$$

$$= O(n \log n)$$

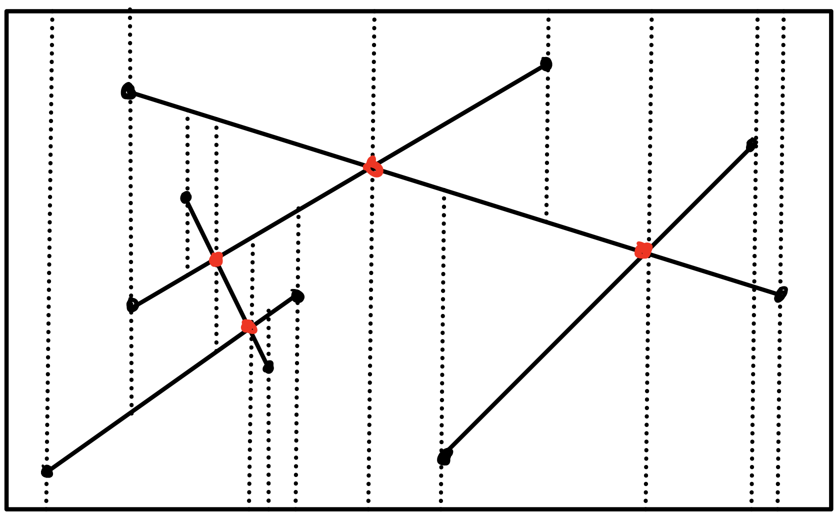
- Space + Query time are in expectation
  - Can we guarantee them?

  $\rightarrow$ Yes: Just rebuild if things go wrong (Increases expected construct time slightly, but still $O(n \log n)$.)

(see text for details)

Line segment intersection (Revisited):

- Can extend trap. maps to intersecting segs.



- Randomized construction can be easily generalized.

Expected time: $O(n \log n + m)$

where $m = \#$ of intersections

This beats plane sweep! $O((n+m)\log n)$

CMSC 754 - Computational Geometry
Lecture 10: Voronoi Diagrams

Metric Spaces: Distances modeled as **metric space** (X, f): $f : X \times X \to \mathbb{R}^{\geq 0}$, s.t. for all $p, q, r \in X$:

- **Symmetry:** $f(p,q) = f(q,p)$
- **Positivity:** $f(p,q) \geq 0$ and $f(p,q) = 0$ iff $p = q$
- **Triangle Inequality:** $f(p,q) \leq f(p,r) + f(r,q)$

Euclidean Distance: for $p, q \in \mathbb{R}^d$:

$$\|p - q\| = \left[\sum_i (p_i - q_i)^2\right]^{1/2}$$

Voronoi Diagram:

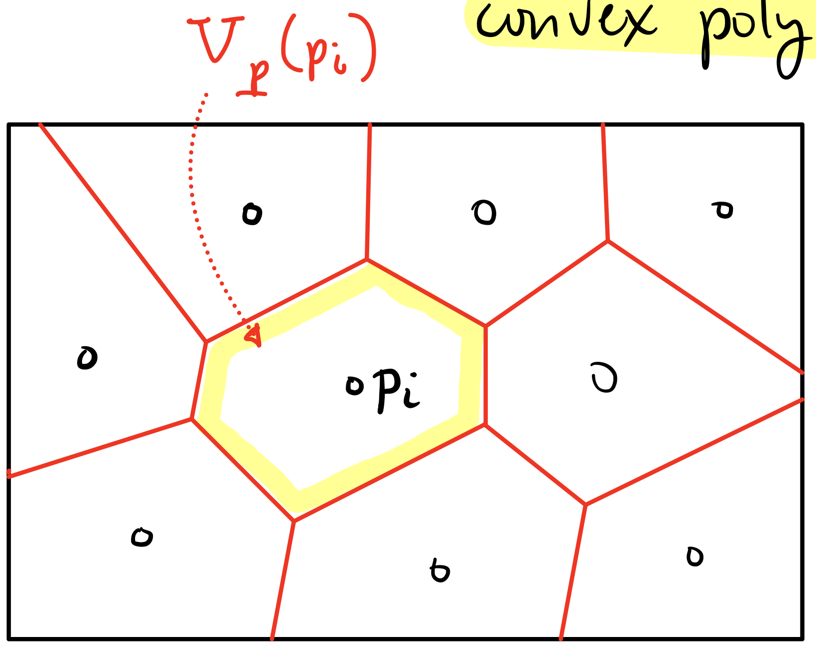
A fundamental structure for metric spaces.

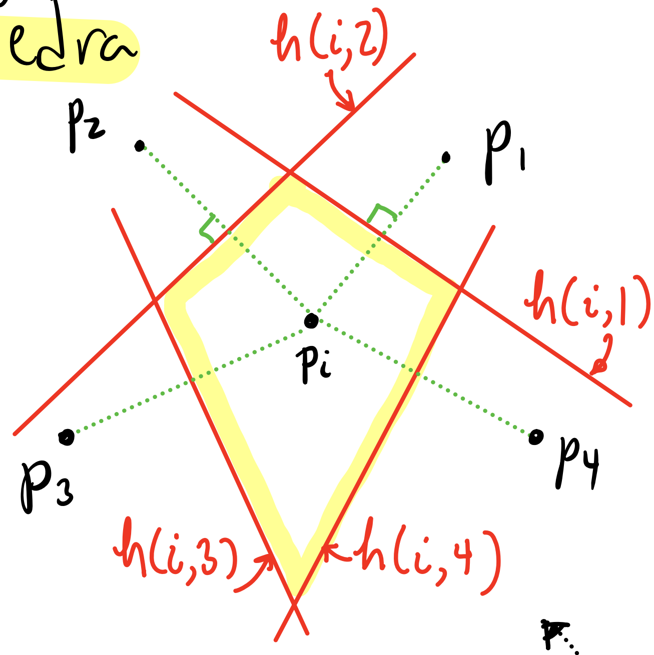
Given a point set $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^d$ called **sites**, we want to subdivide space based on each site's **"region of influence"**

Def: Voronoi cell for site p_i

$$V_P(p_i) = \{ q \in \mathbb{R}^d \mid \|p_i - q\| < \|p_j - q\|, \forall j \neq i \}$$

Obs: - Voronoi cells are disjoint
- For Euclidean dist, Voronoi cells are (possibly unbounded) convex polyhedra





Let $h(i,j) = \{ q \mid \|p_i - q\| < \|p_j - q\| \}$

$h(i,j)$ - halfspace bounded by perpendicular bisector between p_i & p_j

$$\mathrm{Vor}(p_i) = \bigcap_{j \neq i} h(i,j)$$ – intersection of halfspaces ⇒ polytope

Def: Vor(P) is the subdivision (cell complex) induced by P's voronoi cells.

- Vor(P) covers $\mathbb{R}^d$
- Has n cells (faces of dim d)
- Polyhedral subdivision (for Euclidean dist)
- Combinatorial complexity:

  $\mathbb{R}^2$: $O(n)$ edges + vertices

  $\mathbb{R}^d$: $O(n^{\lceil d/2 \rceil})$ size [Closely related to convex polytopes in $\mathbb{R}^{d+1}$]

Many applications:

Nearest neighbor search:

Preprocess a set of sites $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ s.t. given any query point $q \in \mathbb{R}^d$ can find q's nearest site

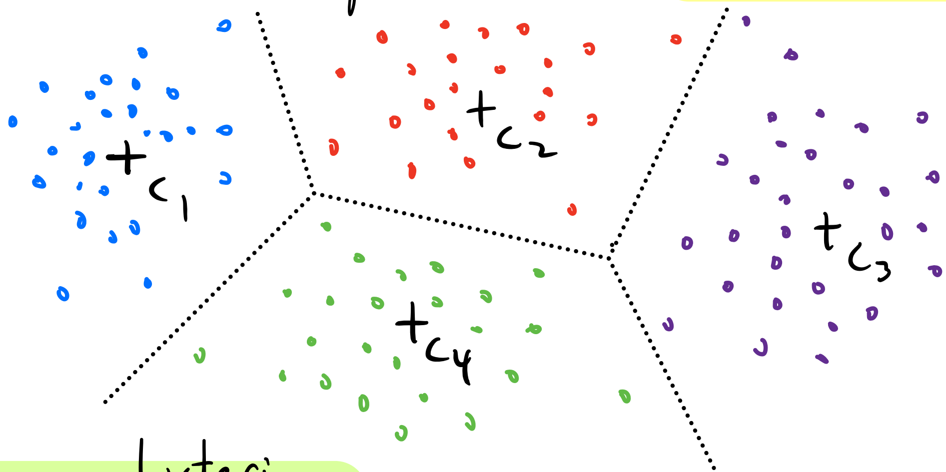
How? - Compute Vor(P)

- Build a point-location data structure for Vor(P)

[Optimal in $\mathbb{R}^2$. Not as good in $\mathbb{R}^d$.]

Point-based Clustering:

- Given set T of training points, group them into k clusters
- Clusters are defined by k cluster centers $\{c_1, \dots, c_k\}$
- Cluster membership based on closest center



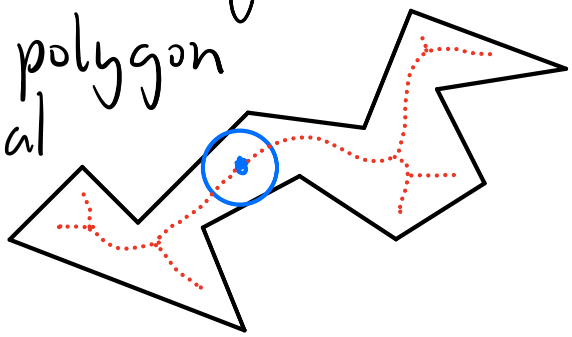
- k-means clustering

Variations:

- Other metrics: L_1-Vor diagram (Manhattan distance)
- Weighted pts:
  - Multiplicative: $\text{dist}(q, p_i) = \alpha_i \| p_i - q \|$
  - Additive: $\text{dist}(q, p_i) = \| p_i - q \| + \omega_i$
- k^{th} Nearest:
  - $\text{Vor}_k(P)$ = subdivide based on k^{th} closest

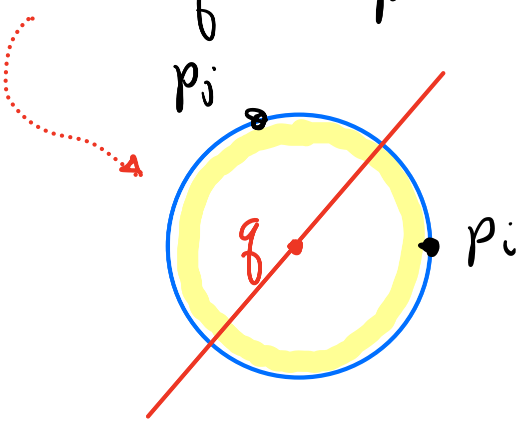
$Vor_n(P)$ = farthest point Vor. diag

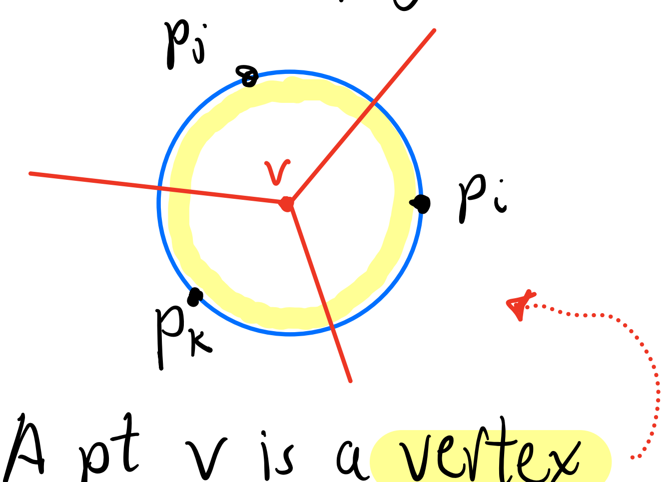
- Other shapes:
  - Voronoi diagram of line segments
  - Medial axis of polygon
    centers of maximal disks



Properties of the Voronoi Diagram:

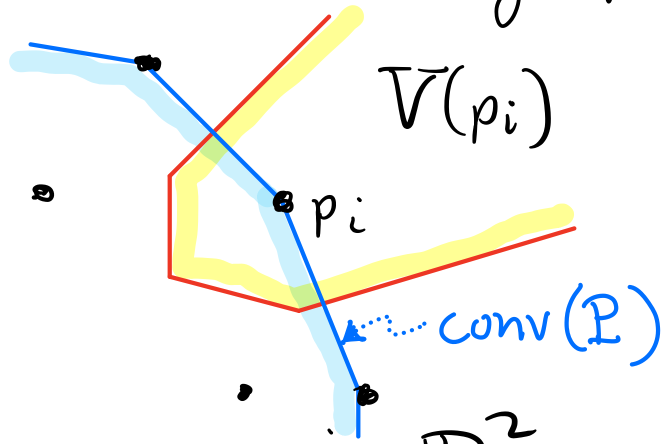
Empty-circle Property: A pt q is on an edge of the Vor. diag iff there is a circle centered at q that passes through 2 sites & is otherwise empty.





Circumcircle Property: A pt v is a vertex of the diagram iff it is the center of a circle passing through 3 sites & is otherwise empty.

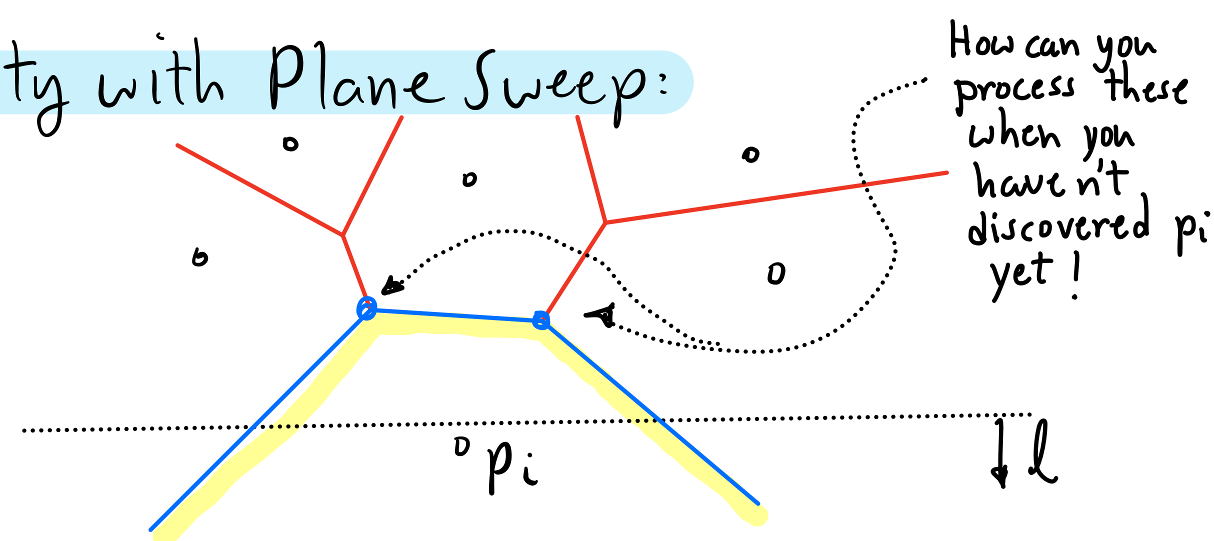
Hull Property: A site p_i has an unbounded Voronoi cell iff p_i is on boundary of convex hull of P.



Constructing Voronoi Diagrams in $\mathbb{R}^2$

- Incremental – add a site; update (best if randomized)
- Divide + Conquer – $O(n \log n)$
- Plane Sweep (this lecture)
  – Fortune's Algorithm – $O(n \log n)$

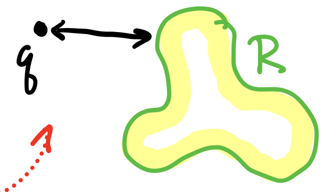
Difficulty with Plane Sweep:



Clever twist: We'll maintain two sweeping structures: **sweep line** + **beach line**

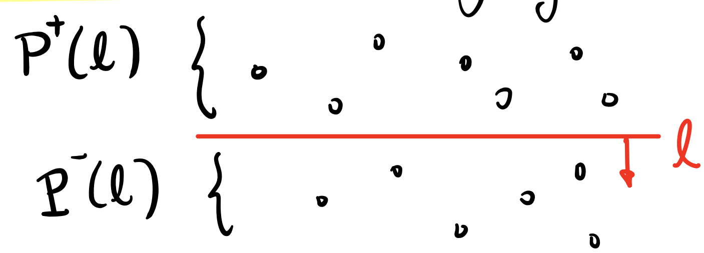
Def: Given a set of pts R and pt q, define

$$\mathrm{dist}(q,R) = \min_{p \in R} \|p-q\|$$



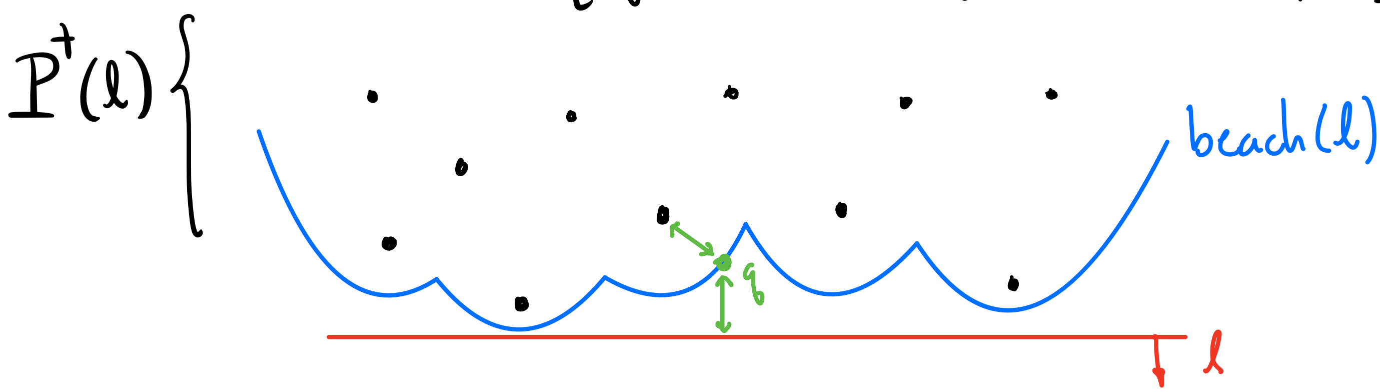
Given a sweep line ℓ (horizontal + moving down) define

$P^+(\ell)$ to be sites lying above ℓ



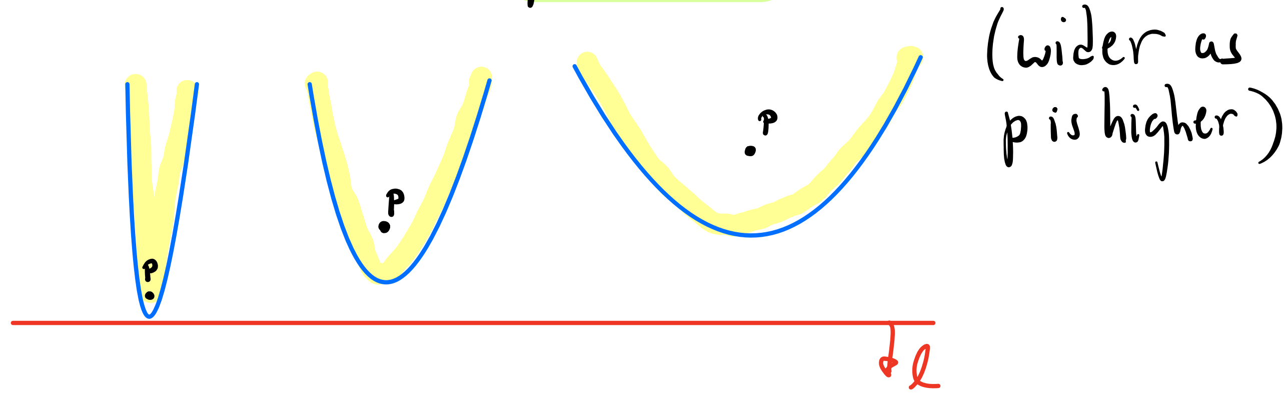
Given sweep line ℓ, define the **beach line** to be set of pts $q \in \mathbb{R}^2$ that are **equidistant from $P^+(\ell)$ and ℓ**

$$\mathrm{beach}(\ell) = \{ q \in \mathbb{R}^2 \mid \mathrm{dist}(q, P^+(\ell)) = \mathrm{dist}(q,\ell) \}$$



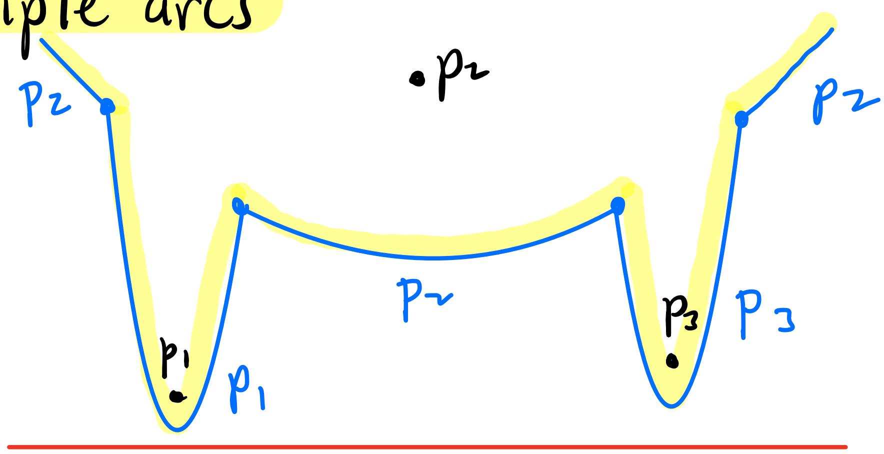
Beach-line Structure:

The points equidistant to a site p + line ℓ form a parabola



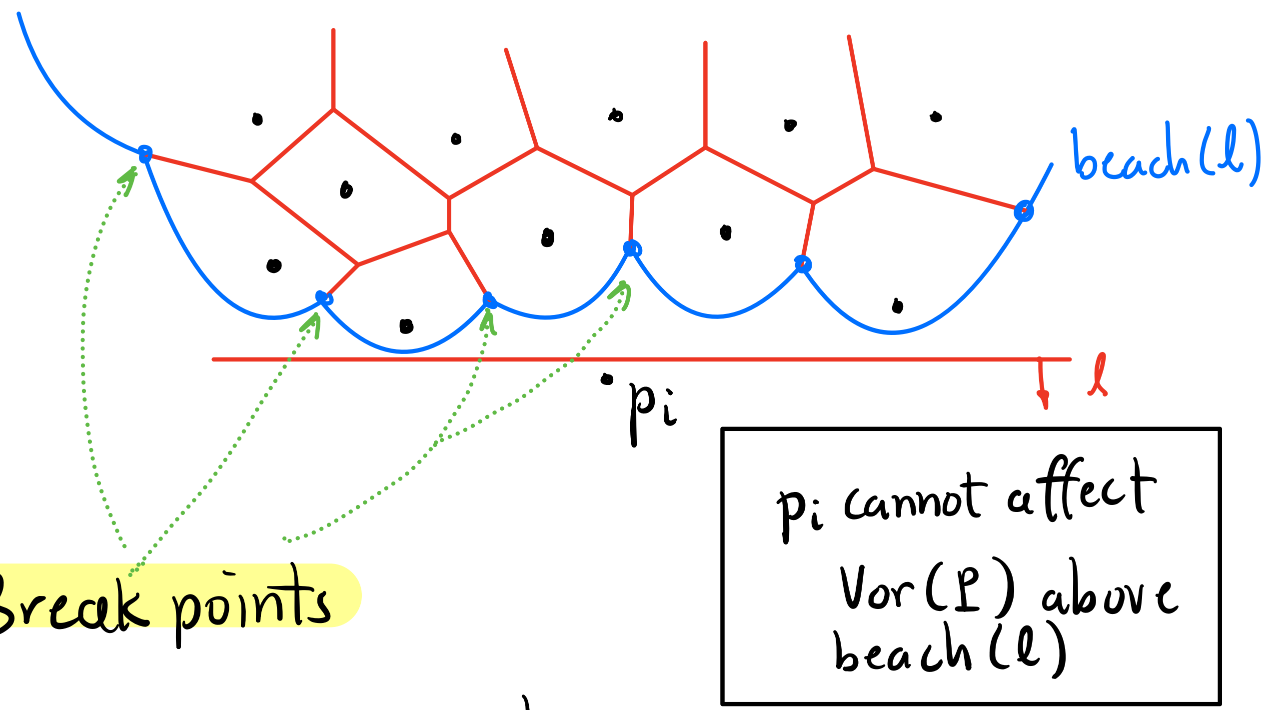
The beach line is the lower envelope of these parabolas for all sites in $P^+(\ell)$

- Beach line is x-monotone
- A single site may contribute 0, 1, or multiple arcs



- Total complexity is $\mathcal{O}(|P^+(\ell)|) = \mathcal{O}(n)$
[Proof: Exercise]

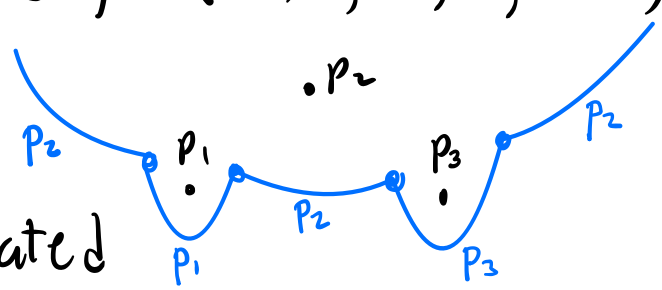
Key: The portion of Vor(P) above the beach line is "safe" from sites lying below ℓ.



Fortune's Algorithm:

Sweep-line status:

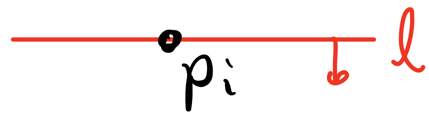
- y-coord of sweep line
- seq. of sites (left to right) that contribute arc to beach line (e.g. $\langle 2,1,2,3,2 \rangle$)
- Parabolic arcs not computed
- Breakpts generated as needed



Voronoi diagram: Portion of Voronoi diagram (rep. as DCEL) above beach line is stored/updated.

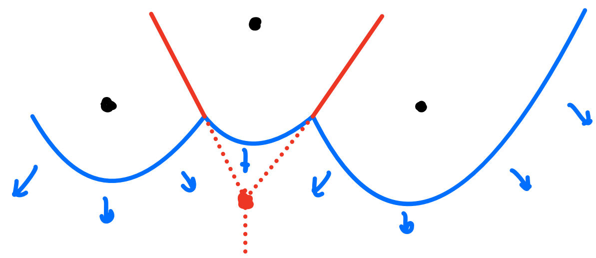
Events:

Site event: Sweep line passes over a site



Vertex event (circle event):
- A new Voronoi vertex is discovered

$\equiv$ An arc on beach line vanishes

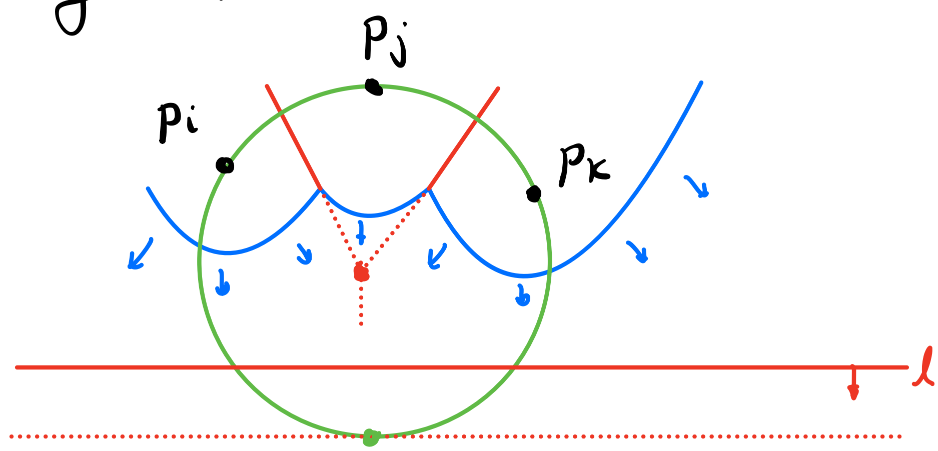


Priority Queue: Stores y-coords for sweep line at events.

Site events: Easy - just y-coord of site (static)

Vertex events: Tricky! (see below)

Scheduling vertex events:



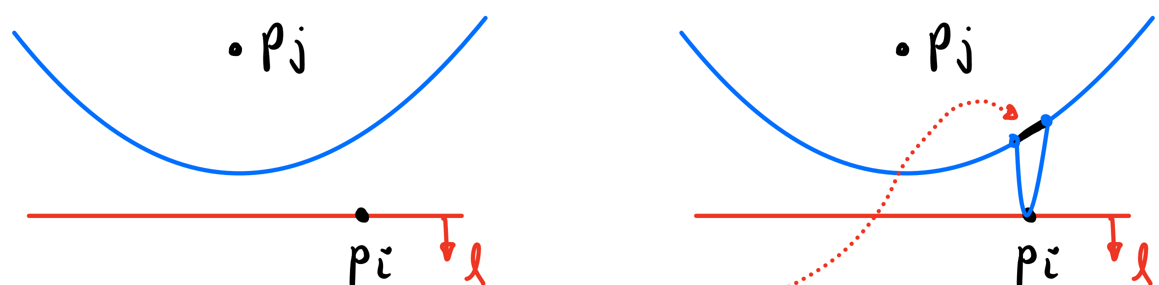
- For each consecutive triple $\langle \ldots p_i, p_j, p_k \ldots \rangle$ on beach line compute lowest y-coord of circumcircle(p_i, p_j, p_k)
- Schedule vertex event when sweep line reaches this y-coord.

Site Event: for site p_i

(1) Find arc of beach line above p_i

(2) Split this arc:

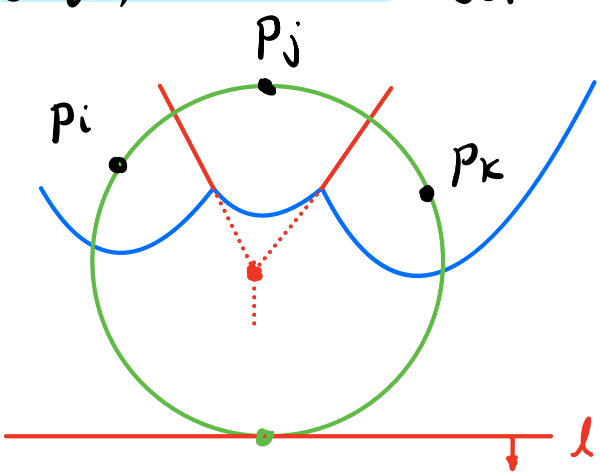
$$\langle \ldots, p_j, \ldots \rangle \Rightarrow \langle \ldots p_j, p_i, p_j, \ldots \rangle$$

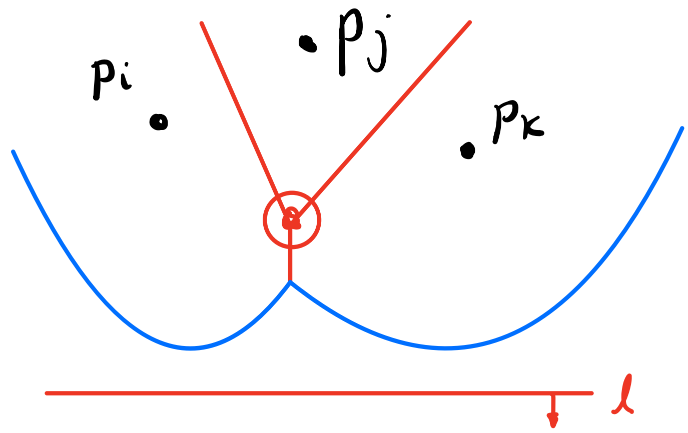


(3) Create a new "dangling edge" (between p_i + p_j) add to Voronoi diagram.

(4) Update priority queue vertex events (below)

Vertex Event: for triple $\langle p_i, p_j, p_k \rangle$





(1) Delete p_j's arc from beach line

$$\langle \cdots p_i \; p_j \; p_k \cdots \rangle \Rightarrow \langle \cdots p_i \; p_k \cdots \rangle$$

(2) Create new Voronoi vertex joining edges $p_i p_j$ + $p_j p_k$ in diagram

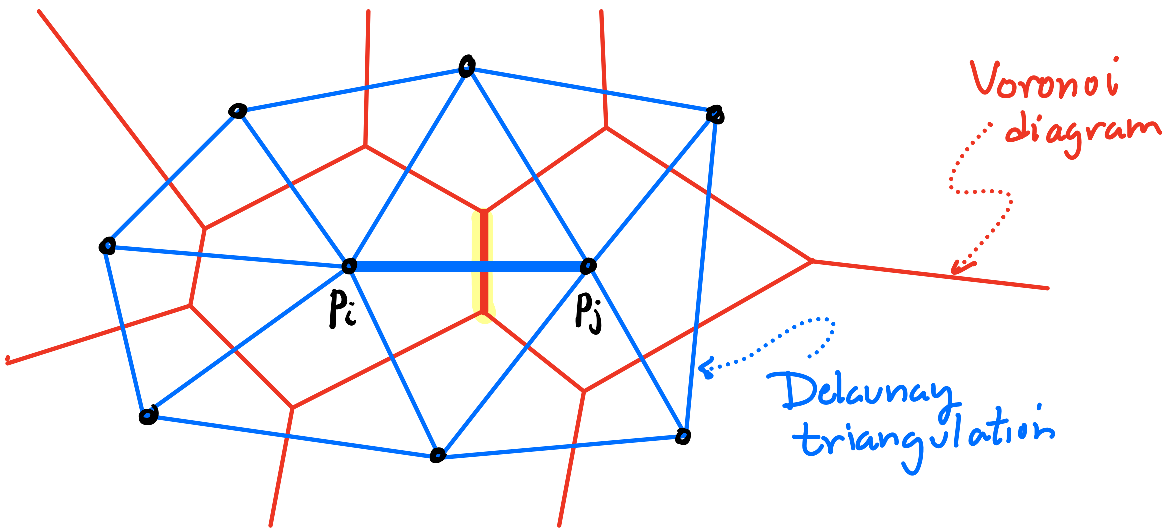
(3) Start new (partial) Voronoi edge for $p_i p_k$

(4) Update priority queue vertex events

Analysis:
- $O(n)$ events
- $O(\log n)$ per event
- $O(n \log n)$ total time

CMSC 754 - Computational Geometry
Lecture 11: Delaunay Triangulations (Properties)

Last lecture - Voronoi Diagrams
This - The dual structure - Delaunay Triangulations



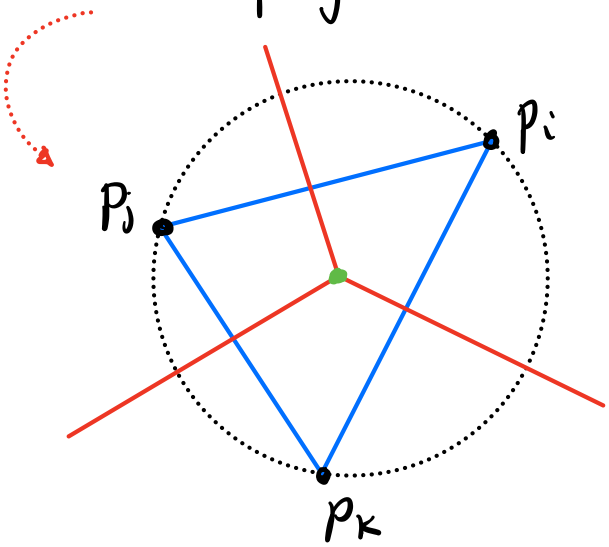
Delaunay Triangulation:
Given a set $P = \{p_1, \ldots, p_n\}$ of sites in $\mathbb{R}^2$, the Delaunay Triangulation is the cell complex whose vertices are sites & there is an edge $\overline{p_i p_j}$ iff $V(p_i)$ & $V(p_j)$ share a common edge. Called $DT(P)$

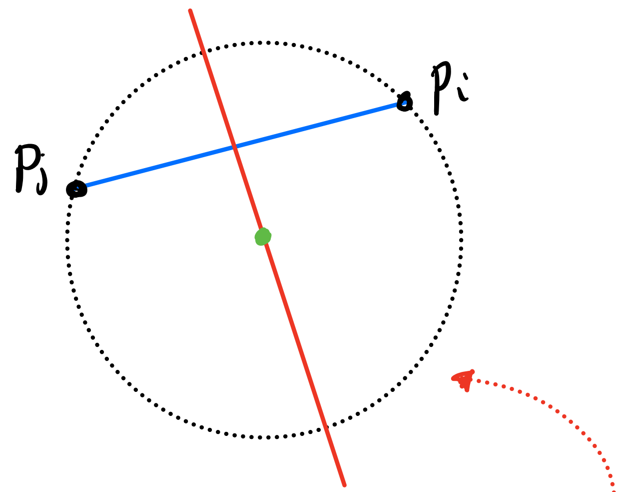
Properties:

Triangulation: If general position (no four sites cocircular), the internal faces are all triangles

Hull: The boundary of the external face is the boundary of conv(P)

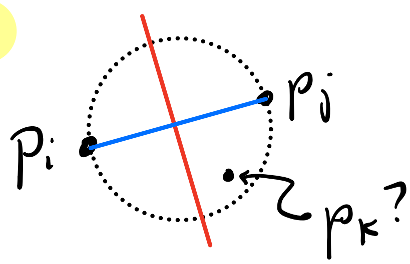
Circumcircle: The circumcircle of any triangle is empty (no sites in its interior)





Empty Circle: Sites p_i + p_j are adjacent in DT(P) iff there is an empty circle through p_i + p_j.

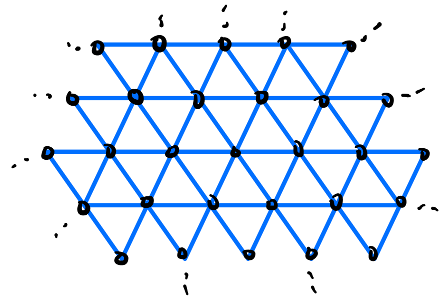
Closest Pair: The closest pair of sites are Delaunay neighbors



- Consider the circle with diameter $\overline{p_i p_j}$.
- No site p_k can lie within
- Apply empty circle prop.

Combinatorial Complexity:

By applying Euler's formula, there are at most $2n$ triangles and at most $3n$ edges



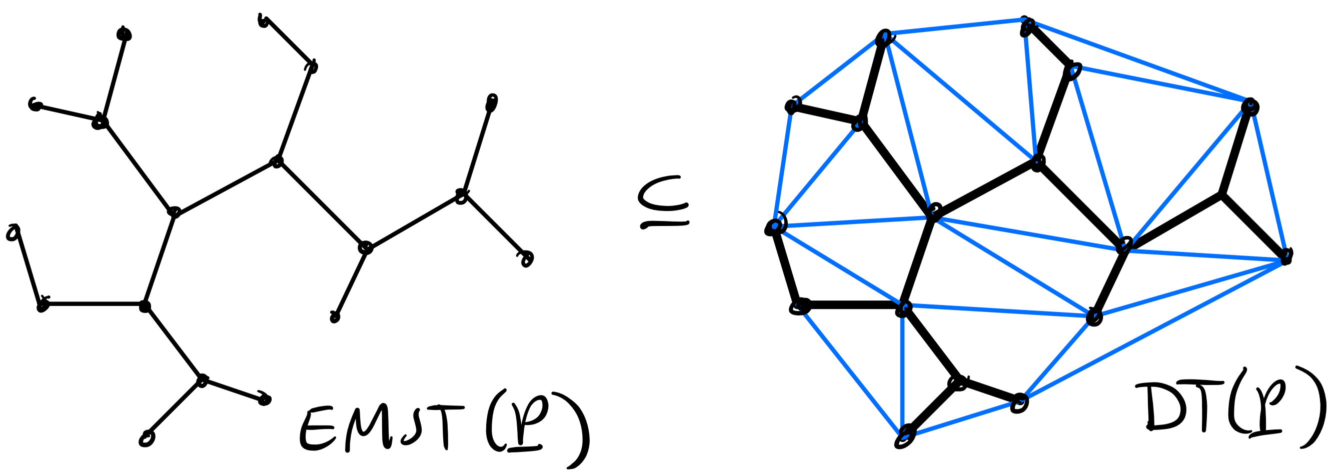
[In $\mathbb{R}^d$, size is $O(n^{\lceil d/2 \rceil})$]

Euclidean Minimum Spanning Tree: (EMST)

Euclidean graph: Complete graph on vertex set $P = \{p_1, \dots, p_n\}$, where edge weight is Euclidean distance $(w(p_i, p_j) = \| p_i - p_j \|)$

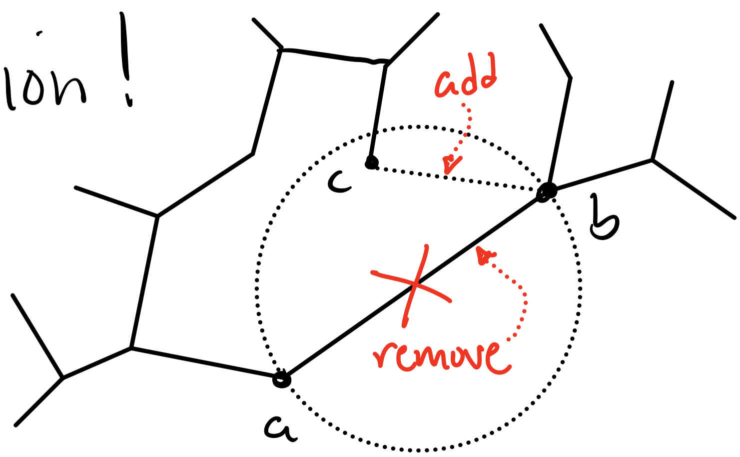
EMST(P) = MST of Euclidean graph (lowest weight tree spanning P)

$\boxed{\text{Thm: } \mathrm{EMST}(P) \subseteq \mathrm{DT}(P)}$



Proof: (Contradiction)

- Suppose some edge $\overline{ab} \in EMST(P)$ but not in $DT(P)$
- Empty circle $\Rightarrow$ circle with diameter $\overline{ab}$ contains site c
- $\|ac\| < \|ab\|$
  $\|bc\| < \|ab\|$
- Can remove $\overline{ab}$ from EMST + replace with either $\overline{ac}$ or $\overline{bc}$ to produce a spanning tree of lower weight
- Contradiction!



$\square$

Minimum Weight Triangulation: **No!**

MWT(P) = triangulation of P whose sum of edge lengths is minimum

Generally $MWT(P) \neq DT(P)$

Notation: Given graph $G=(V,E)$ and vertices $u,v \in V$, let $\delta_G(u,v)$ = shortest path distance in G from u to v.

Spanner Properties:

Given a graph G and $t \geq 1$, a t-spanner is a subgraph G' of G on same vertex set s.t. $\forall\, u,v \in V$,

$$\delta_{G'}(u,v) \leq t \cdot \delta_G(u,v)$$

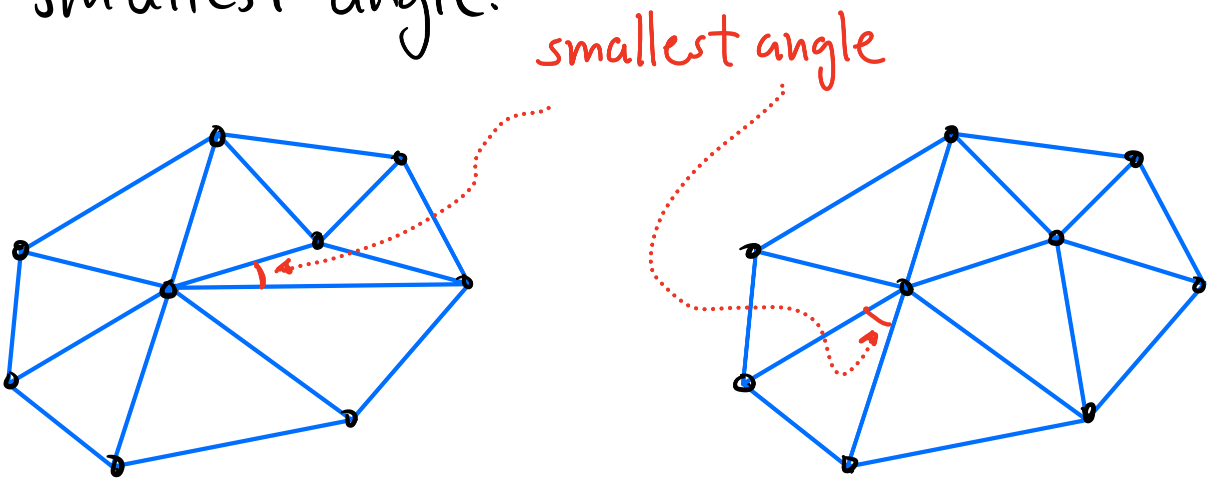
(Path lengths don't stretch too much)

Theorem (Keil + Gutwin, '92) Given a set P of sites in the plane, $DT(P)$ is a $4\pi\sqrt{3}/9 \approx 2.418$ spanner of the Euclidean graph. That is, $\forall\, p,q \in P$

$$\delta_{DT(P)}(p,q) \leq \frac{4\pi\sqrt{3}}{9} \cdot \|p-q\|$$

Avoids skinny triangles:

Let P be a set of sites in the plane. Among all possible triangulations of P, $DT(P)$ maximizes the size of the smallest angle.

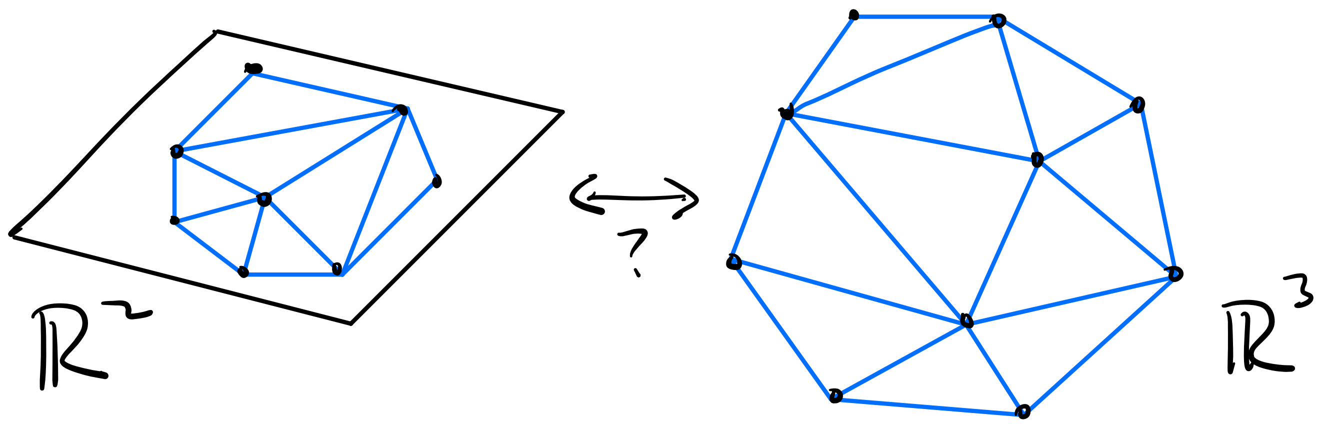


other triangulation $\qquad\qquad$ $DT(P)$

> **Thm:** If all angles of all triangles are ordered small to large, $DT(P)$ is the largest lexicographically compared to all triangulations of P.

(See full lecture notes)

Relationship to polytopes in $\mathbb{R}^{d+1}$



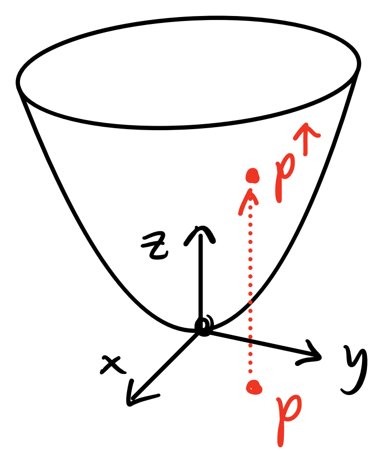
Delaunay triangulation in $\mathbb{R}^d$ is the projection of a lower convex hull in $\mathbb{R}^{d+1}$

Voronoi diagram in $\mathbb{R}^d$ is the projection of a lower envelope of hyperplanes in $\mathbb{R}^{d+1}$

→ We'll prove the first only: $\mathbb{R}^2 \to \mathbb{R}^3$

Consider the paraboloid:

$$z = f(x,y) = x^2 + y^2$$



Given $p = (p_x, p_y)$, define $p^\uparrow$ to be $(p_x, p_y, p_x^2 + p_y^2)$

Lemma:

Three pts $p, q, r \in \mathbb{R}^2$ have an empty circumcircle w.r.t. P	$\Leftrightarrow$	Three pts $p^\uparrow, q^\uparrow, r^\uparrow$ lie on plane h with all pts of $P^\uparrow$ above

- Let $c = (c_x, c_y)$ be center of circumcircle through p, q, r & let r be its radius
- The plane tangent to paraboloid at $c^\uparrow$ is:

$$z = 2c_x \cdot x + 2c_y \cdot y - (c_x^2 + c_y^2)$$

- Shift this plane up by distance r^2:

$$h: \quad z = 2c_x \cdot x + 2c_y \cdot y - (c_x^2 + c_y^2) + r^2$$

- All 3 lifted pts lie on this plane:

p_x on circle: $(p_x - c_x)^2 + (p_y - c_y)^2 = r^2$

$$\Leftrightarrow (p_x^2 - 2p_x c_x + c_x^2) + (p_y^2 - 2p_y c_y + c_y^2) = r^2$$

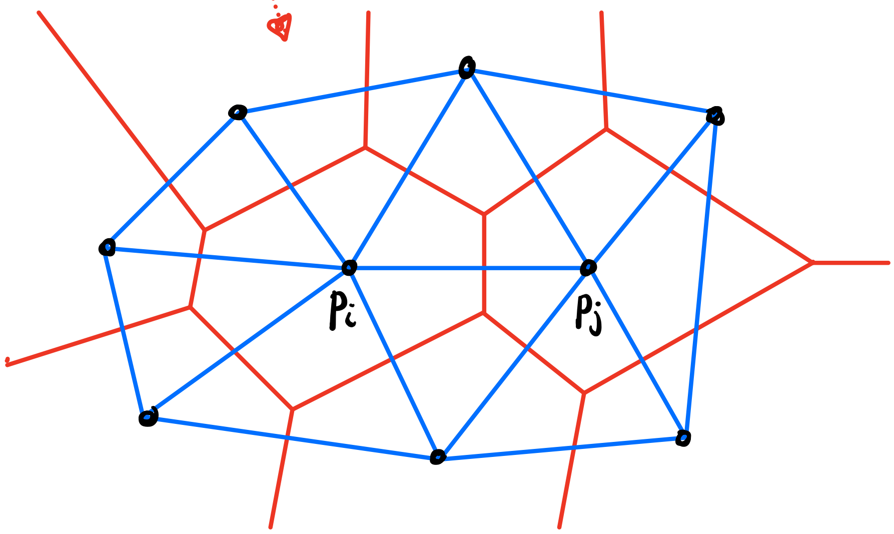
$$\Leftrightarrow p_x^2 + p_y^2 = 2c_x \cdot p_x + 2c_y \cdot p_y - (c_x^2 + c_y^2) + r^2$$

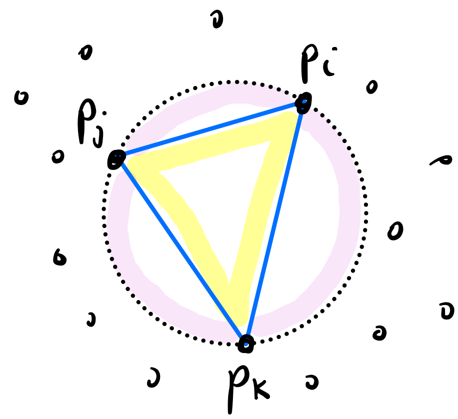
$$\Leftrightarrow p_z^\uparrow = 2c_x \cdot p_x^\uparrow + 2c_y p_y^\uparrow - (c_x^2 + c_y^2) + r^2$$

$\Leftrightarrow$ $p^\uparrow$ lies on plane h

CMSC 754 - Computational Geometry
Lecture 12: Delaunay Triangulations (Construction)

Last lecture: - Delaunay triangulation + properties
- Given a set $P = \{p_1, \dots, p_n\}$ of sites, DT(P) is the dual of Vor(P)

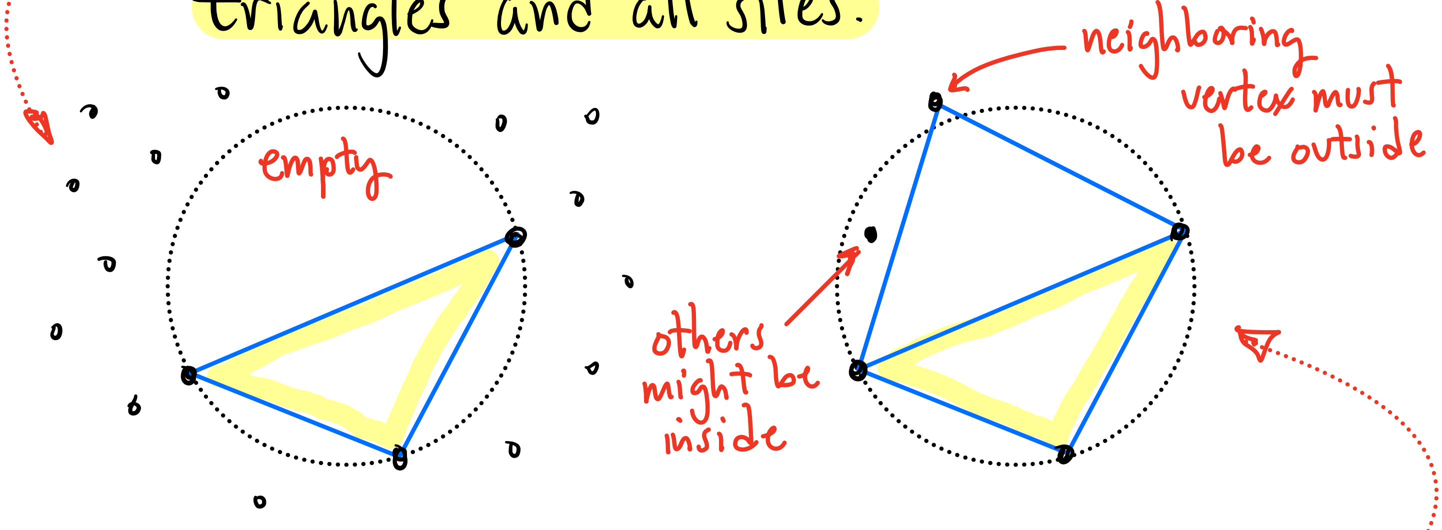




Circumcircle Property:
$\Delta p_i p_j p_k \in DT(P)$ iff circumcircle of p_i, p_j, p_k contains no sites

Local/Global Delaunay:

- A triangulation is globally Delaunay if circumcircle property holds for all triangles and all sites.



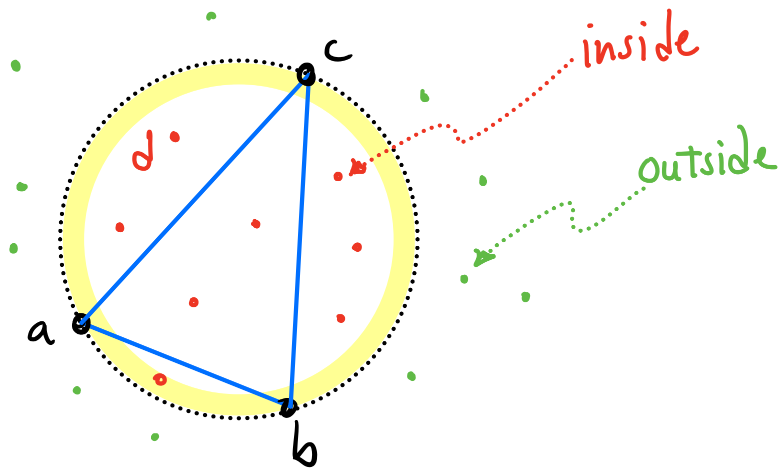
- A triangulation is locally Delaunay if circumcle property holds the vertices of every pair of adjacent triangles.

Does it matter? No.

Thm (Delaunay): A triangulation is globally Delaunay iff it is locally Delaunay.

(See lecture notes/text for proof)

Incircle Test: Given points a, b, c + $d \in \mathbb{R}^2$, does d lie in circumcircle of Δabc?
(Assume a, b, c given in CCW order)



inCircle(a, b, c; d): d is inside if

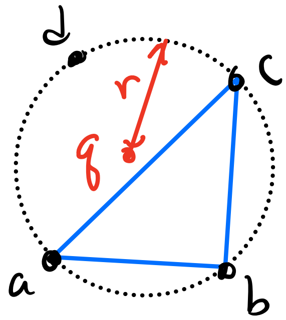
$$\det \begin{pmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{pmatrix} > 0$$

Obs:

- This is an orientation test in $\mathbb{R}^3$
- Generalizes to any dimension
- Computable in $O(1)$ time in any fixed dimension.

Why? Consider boundary case → cocircular

Center $q = (q_x, q_y)$ radius $= r$



$$\Rightarrow (a_x - q_x)^2 + (a_y - q_y)^2 = r^2$$

$$\Rightarrow \boxed{-2q_x}\, a_x \boxed{-2q_y} \cdot a_y + \boxed{1} \cdot (a_x^2 + a_y^2) + \boxed{(q_x^2 + q_y^2 - r^2)} = 0$$

Same applies to b, c, d ⇒

$$\begin{pmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{pmatrix} \begin{pmatrix} -2q_x \\ -2q_y \\ 1 \\ q_x^2 + q_y^2 - r^2 \end{pmatrix} = 0$$

A linear combination of columns is identically 0

⇒ column vectors are lin. dependent

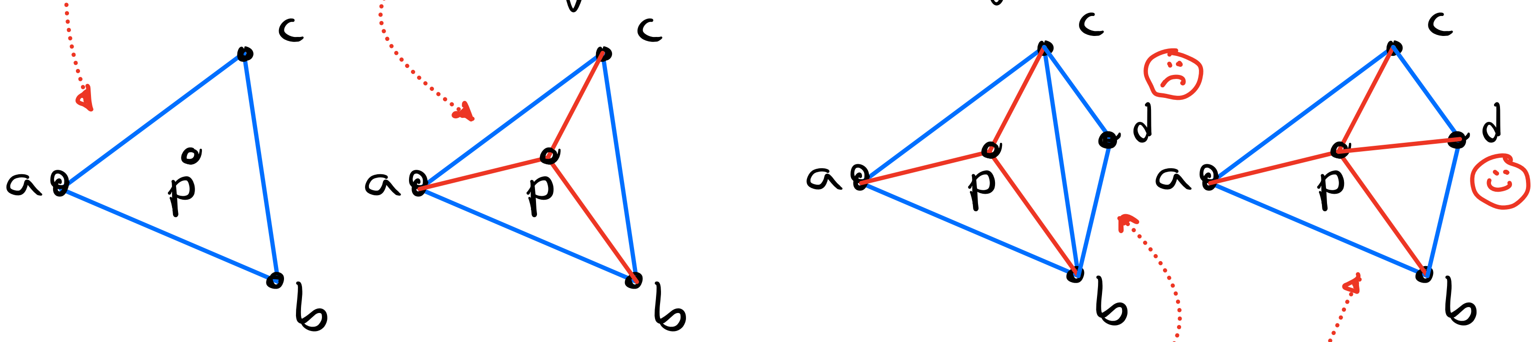
⇒ det of matrix is 0

(Randomized) Incremental Construction:

- Add sites one-by-one in random order + update the triangulation after each.

① Find triangle Δabc containing the new site p.

② Add edges connecting p to a, b, c



③ Check neighboring triangles for violations of local Delaunay

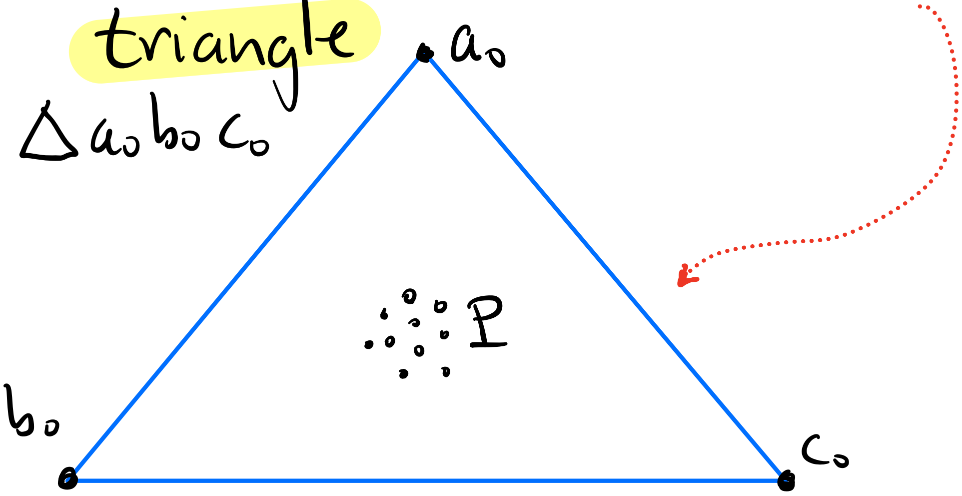
④ Apply edge flips to correct these

⑤ Repeat until local Delaunay for all neighbors

Sentinel sites:

- If new site is not in convex hull – it's not in any triangle!

- Fix: Enclose points in a HUGE triangle $\triangle a_0 b_0 c_0$



How huge? No circumcircle from P should contain a_0, b_0 or c_0

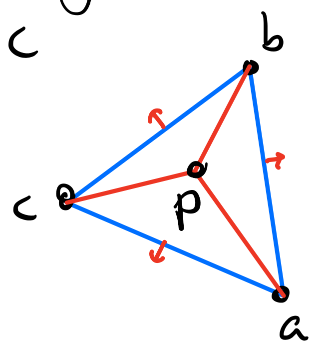
(see text for how)

Build-Delaunay($P = \{p_1, \ldots, p_n\}$)

- Create sentinel triangle $\triangle a_0 b_0 c_0$ containing P
- Randomly permute P
- for $i = 1$ to n Insert(p_i)

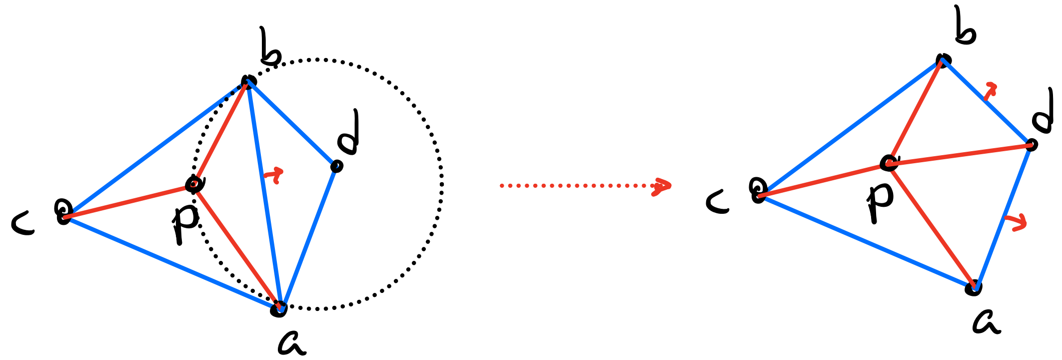
Insert(p):

- Find $\triangle abc$ containing p
- Add edges pa, pb, pc
- SwapTest(ab)
- " (bc)
- " (ca)

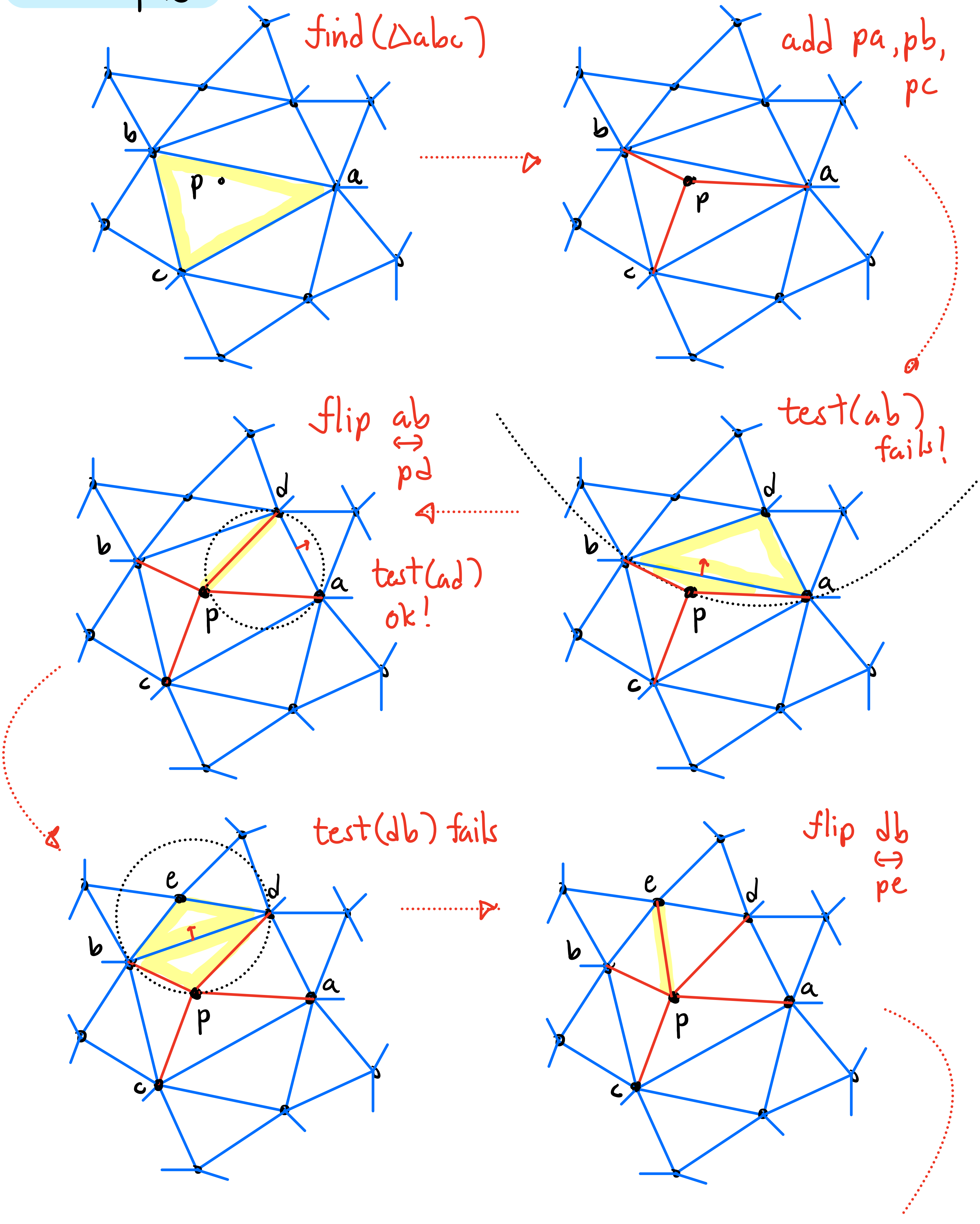


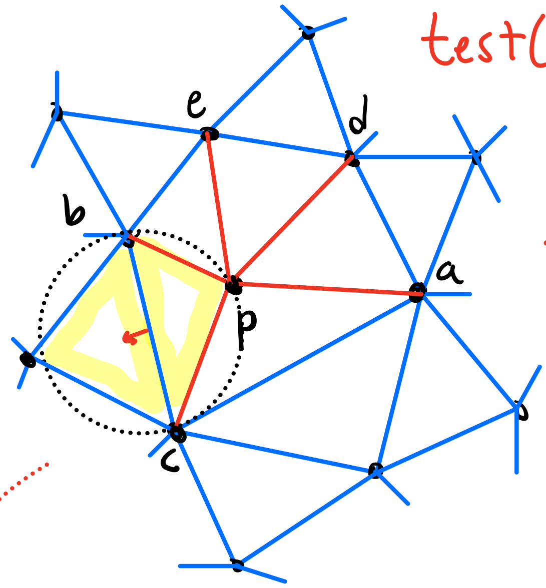
SwapTest(ab):

- if (ab is edge of external face)
  return
- d ← vertex opposite p on ab
- if (inCircle(p, a, b, d))
  - flip edge ab (for pd)
  - SwapTest(ad)
  - SwapTest(db)

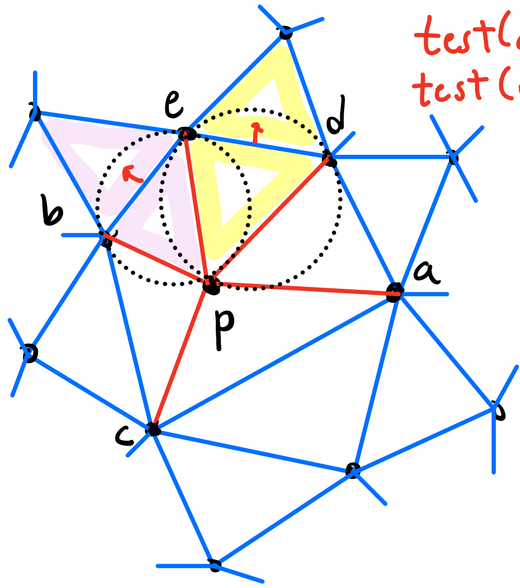


Example:

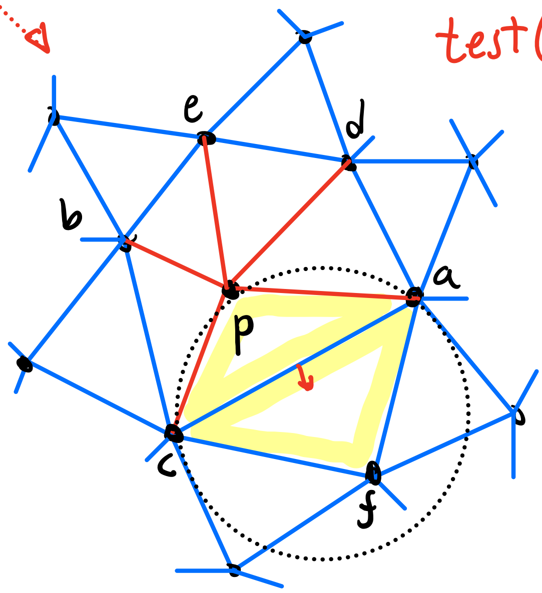

find(△abc)
add pa, pb, pc
test(ab) fails!
flip ab ↔ pd
test(ad) ok!
test(db) fails
flip db ↔ pe



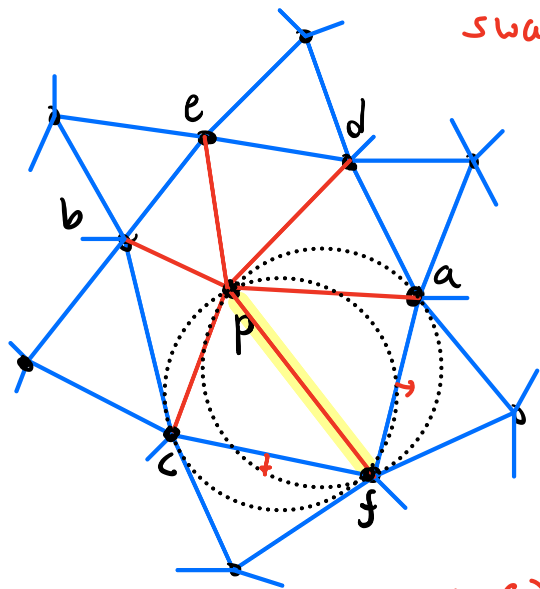
test(bc) ok!



test(de) ok!
test(cb) ok!



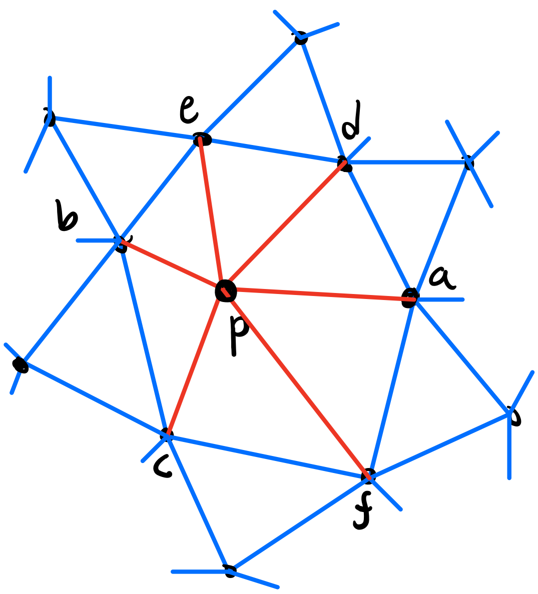
test(ca) fails



swap ca ↔ pf

test(cf) ok!
test(fa) ok!

Done!



Final

Note: All new edges are incident to p

Correctness:

- Only triangles that could violate local Delaunay are incident to p, + we check all
- By Delaunay's Thm, local $\Rightarrow$ global Delaunay

Running time:

- for each insertion $p_1 \dots p_n$
- find triangle containing $p_i \rightarrow O(\log n)$
- swaptests + edge flips $\rightarrow O(1)$

in expectation

- Total: $O(n \log n)$

Lemma: The expected update time (swaptests + edgeflips) is $O(1)$

Proof: (Backwards analysis)

- Update time ~ degree of p in final Δ-tion
- Every pt is equally likely to be last
- $\sum_i \deg(p_i) = 2(\#\text{edges}) \leq 2(3n) = 6n$
- Expected time ~ Average degree $\leq \frac{1}{n} \cdot 6n = 6$

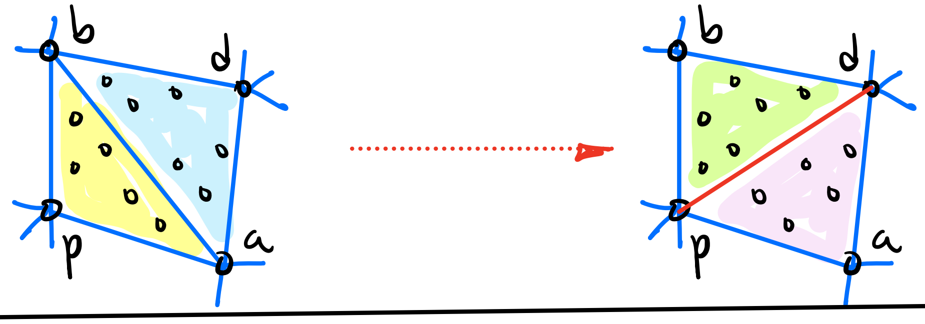
$\square$

Point-Location:

Bucketing:

called a "bucket"

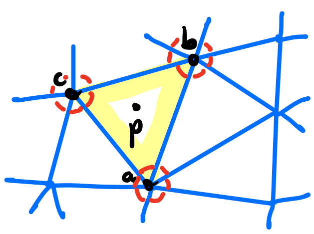
- Each triangle maintains future sites in this triangle
- To locate a point, just get its bucket id.
- When an edge flip is performed rebucket the affected sites



Lemma: Let p be any site. The probability that p is rebucket as result of i^{th} insertion is $\leq 3/i$

Proof: (Backwards Analysis)

- Sites are rebucketed only if in new triangle
- All new triangles are incident to last site
- Each site is equally likely to be last
- Prob(p is rebucketed) [let $p \in \Delta abc$]



$\leq$ Prob(a, b, or c was last inserted)

$\leq 3/i$ $\square$

Lemma: Total time for rebucketing is $O(n \log n)$ in expectation

Proof: Rebucket time (expected)

$$= \sum_{p \in P} \sum_{i=1}^{n} 1 \cdot \text{Prob}(p \text{ was rebucketed in } i^{th} \text{ insertion})$$

$$\leq \sum_{p \in P} \sum_{i=1}^{n} 3/i \approx \sum_{p \in P} 3 \cdot \ln n \quad \left(\begin{matrix}\text{Harmonic} \\ \text{series}\end{matrix}\right)$$

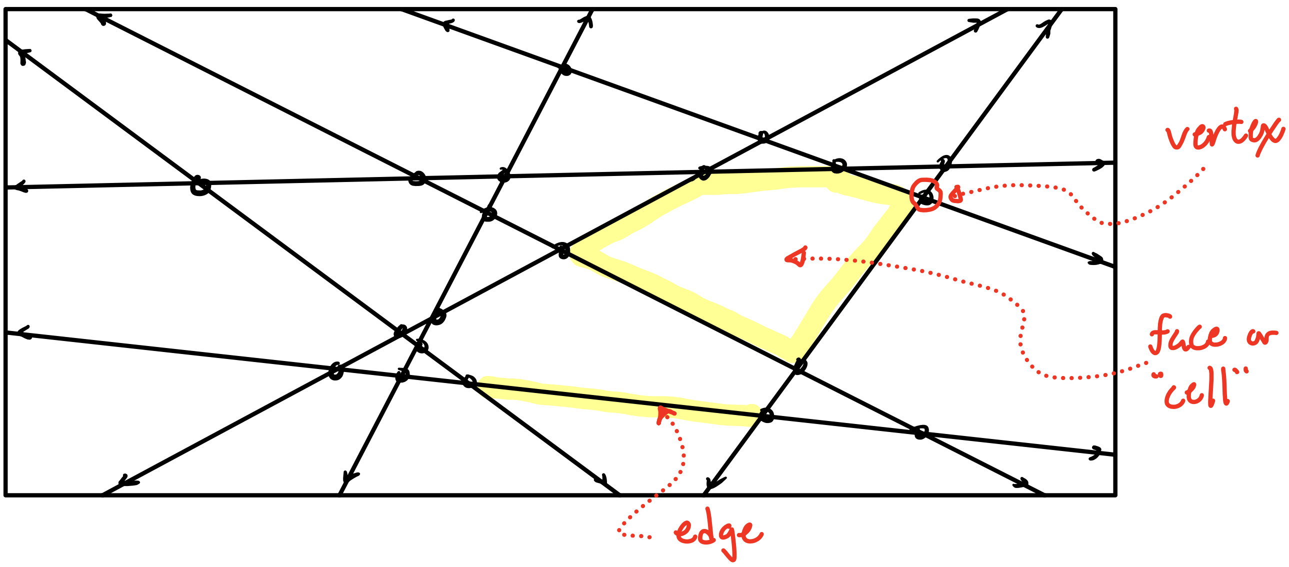
$$= 3 n \ln n$$

$$= O(n \log n) \qquad \square$$

CMSC 754 - Computational Geometry
Lecture 13 - Line Arrangements

Arrangement:

Given a set $L = \{\ell_1, \dots, \ell_n\}$ of lines in $\mathbb{R}^2$ (generally $(d-1)$-dim hyperplanes in $\mathbb{R}^d$), they subdivide the plane into a cell complex called the arrangement of L, or $A(L)$.



Combinatorial Properties:

Lemma: Given n lines L in gen'l position in $\mathbb{R}^2$:

(i) $A(L)$ has $\binom{n}{2} = \frac{1}{2} \cdot n(n-1)$ vertices

(ii) $A(L)$ has n^2 edges

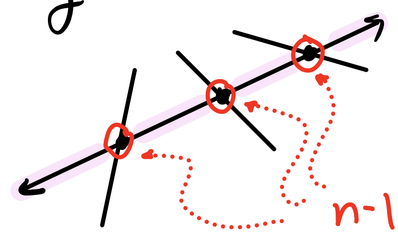
(iii) $A(L)$ has $\binom{n}{2} + n + 1 = \frac{1}{2}(n^2 + n + 2)$ cells

Proof:

(i) Each pair intersects once $= \binom{n}{2}$ ✓

(ii) Each line is split by $n-1$ others into n edges

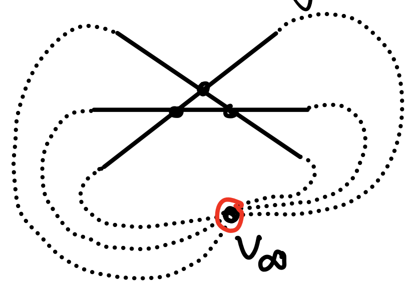
$\Rightarrow$ n^2 total ✓



(iii) Add a vertex at ∞ of degree n to tie off all unbounded edges

$v = \binom{n}{2} + 1$

$e = n^2$



By Euler's formula:

$$v - e + f = 2$$

$$\Rightarrow \left(\binom{n}{2}+1\right) - n^2 + f = 2$$

$$\Rightarrow f = 2 + n^2 - \left(\binom{n}{2}+1\right)$$

$$\Rightarrow f = 2 + n^2 - \frac{n(n-1)}{2} - 1$$

$$= \frac{1}{2}(n^2 + n + 2) \checkmark \qquad \square$$

[In $\mathbb{R}^d$, complexity is $\Theta(n^d)$]

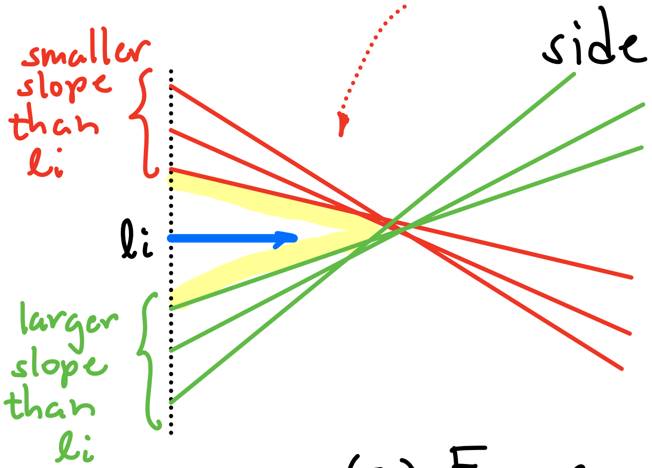
Incremental Construction: (not randomized)

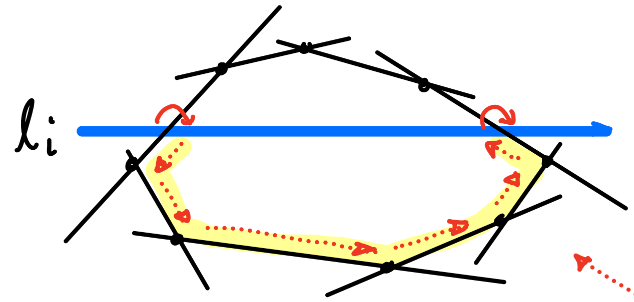
Idea: Add lines one by one (in any order)
Update the structure after each

Notation: $L_i = \{\ell_1, \ldots, \ell_i\}$

How to add the i^{th} line? ℓ_i

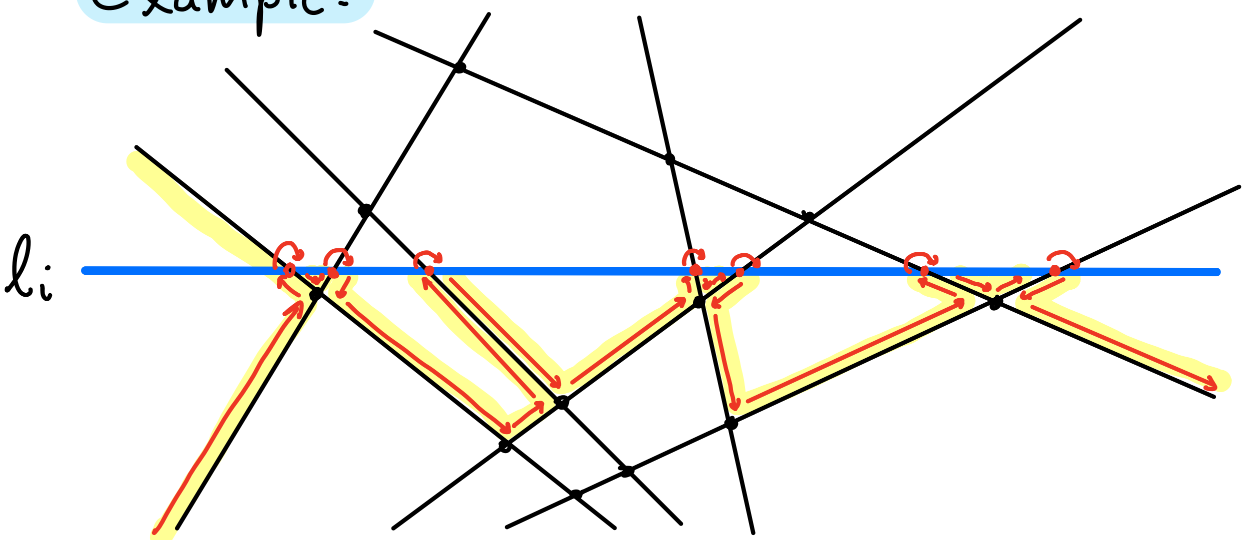
(1) Find the unbounded cell on left side where ℓ_i starts (slope based)





(2) For each face of $A(L_{i-1})$ that intersects ℓ_i, walk along its lower boundary to determine where it exits this cell

Example:



- Once we know entry-exit points on each face - we update arrangement in $O(i)$ time (DCEL)
- How long to crawl around edges?

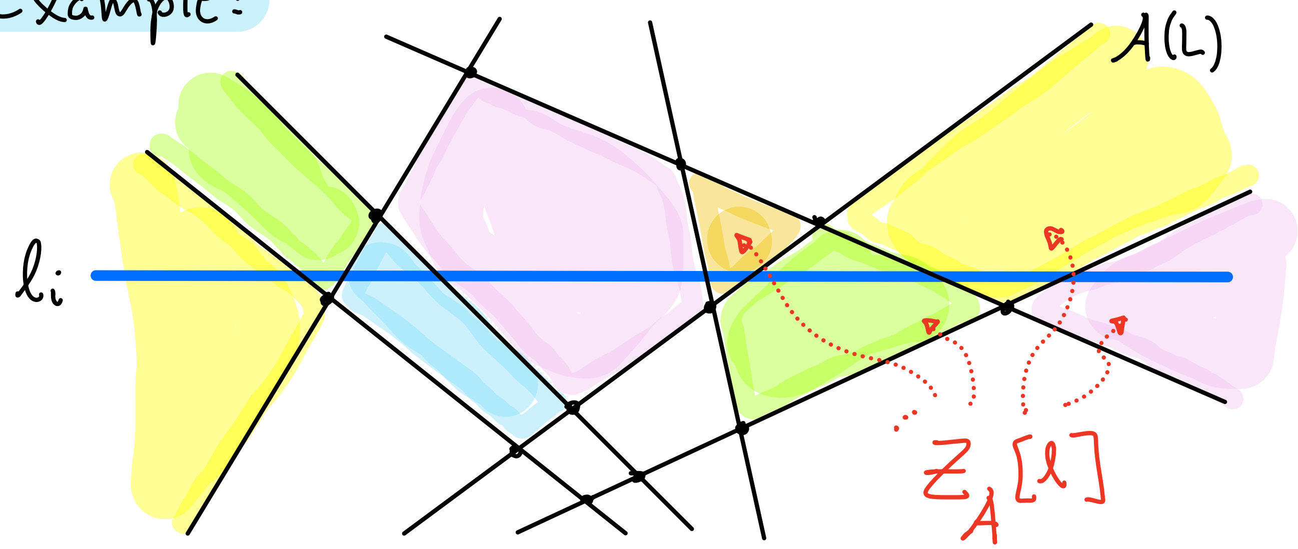
Naive analysis: On adding ℓ_i

- ℓ_i crosses i cells
- each cell may have as many as $i-1$ edges
- crawl takes $O(i(i-1)) = O(i^2)$ time
- total time $\cong \sum_{i=1}^{n} i^2 = O(n^3)$

Can it really be this bad?

Zone: Given an arrangement $A = A(L)$ and a line $\ell \notin L$, zone of ℓ in A, $Z_A(\ell)$ is the set of cells of A that ℓ intersects.

Example:



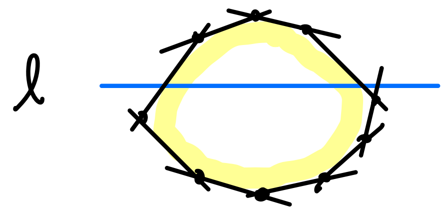
Obs: Crawl time $\leq$ no. of edges on the zone of ℓ_i in $A(L_{i-1})$ $[Z_{A(L_{i-1})}(\ell_i)]$

→ We'll show this is $O(i)$ not $O(i^2)$

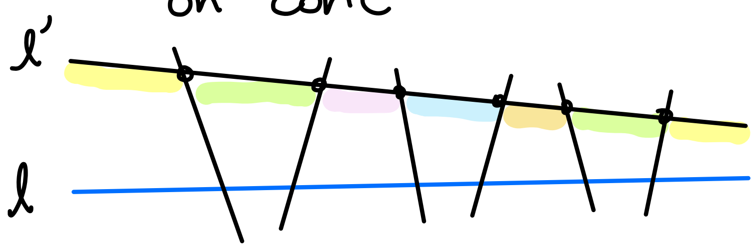
Theorem: (Zone Theorem) Given an arrangement $\mathcal{A}(L)$ where $|L| = n$ and any line $\ell \notin L$, the number of edges in $Z_{\mathcal{A}}(\ell) \leq 6n$

How to prove this?

cell by cell? Some cells have high complexity



line by line? Some lines appear many times on zone



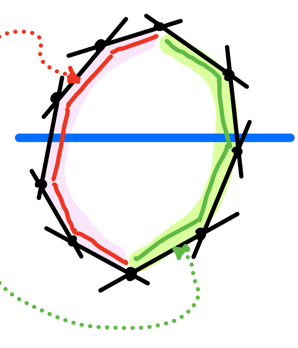
Our approach:

- Partition edges of zone into two classes (left side + right side)
- Show (by induction) at most 3n of each

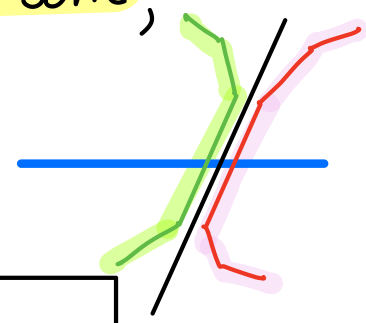
A zone edge is:

left bounding: on left side of cell

right bounding: on right side of cell



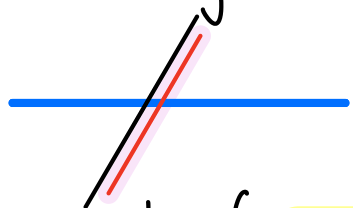
Note: Some edges appear twice in the zone, both as left/right bounding



Claim: At most $3n$ left-bounding edges.

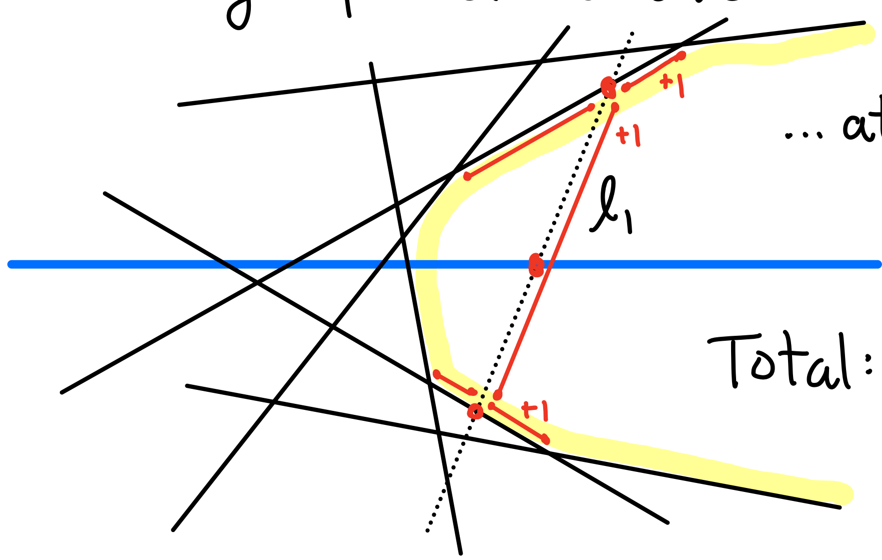
Proof: By induction on n

$n=1$: Just one LB edge $1 \leq 3 \cdot 1$ ✓



$n \geq 2$: I.H. arrangement of $n-1$ lines has $\leq 3(n-1)$ LB edges in zone

- Let $\ell_1 \in L$ be rightmost line to cross ℓ
- Removing $\ell_1 \Rightarrow$ at most $3(n-1)$ LB edges
- Adding ℓ_1 back creates ...



... at most 3 new LB edges

Total: $\leq 3(n-1)+3$
$= 3n$ $\square$

Thm: Given a set L of n lines in $\mathbb{R}^2$, $\mathcal{A}(L)$ can be built in time $O(n^2)$ [and has size $O(n^2)$ ... so this is optimal]

Proof: - Apply incremental construction
- Inserting ℓ_i takes time ~ no. of edges in $Z_{\mathcal{A}(L_{i-1})}(\ell_i) \leq 6(i-1)$
- Total time $\leq \sum_{i=1}^{n} 6(i-1) = 6\sum_{i=0}^{n-1} i = O(n^2)$

Applications:

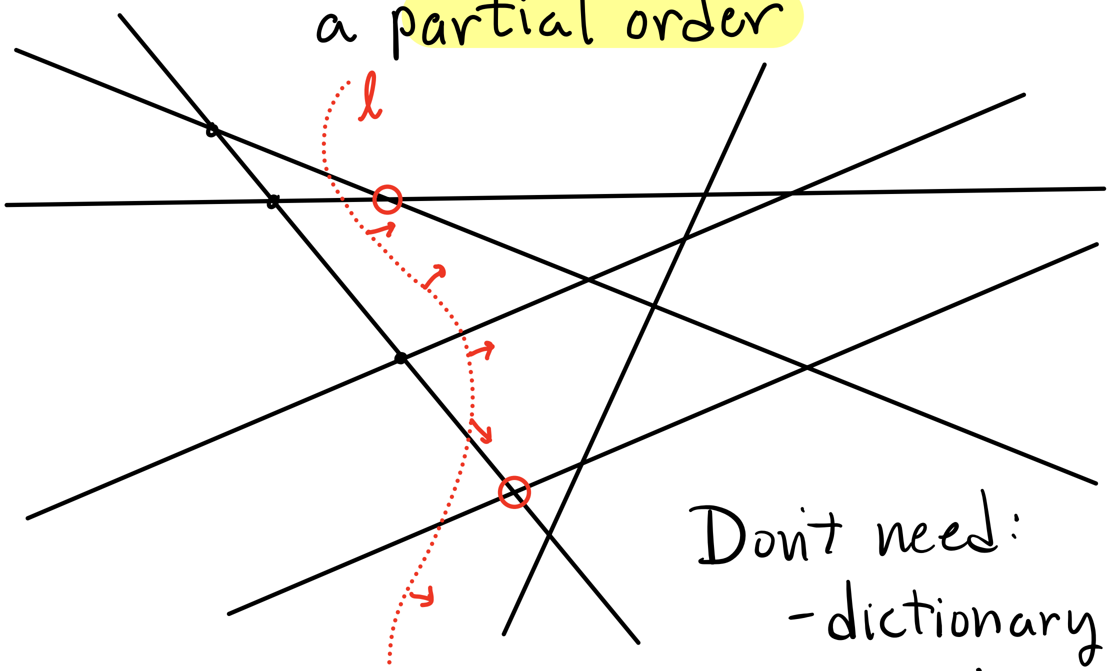
Line arrangements can be used to solve many problems - mostly $O(n^2)$ time
- often using duality

How to process an arrangement?
- Build it + traverse it like a graph
  $O(n^2)$ time, $O(n^2)$ space
- Plane sweep
  $O(n^2 \log n)$ time, $O(n)$ space
- Topological plane sweep
  $O(n^2)$ time, $O(n)$ space

Not covered, but applicable pretty much whenever plane sweep is. ← you may assume this

Topological plane sweep:

- A relaxed version of plane sweep
- Vertices are not swept in strict left to right order, but based on a partial order



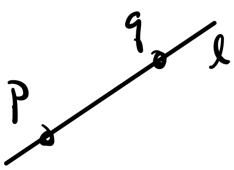
Don't need:
- dictionary
- priority queue

$\Rightarrow$ saves $\log n$ factor

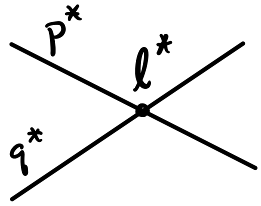
Recall: Dual transformation

$p = (a, b) \longleftrightarrow p^*: y = ax - b$

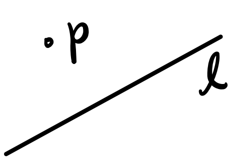
$\ell: y = ax - b \longleftrightarrow \ell^*: (a, b)$



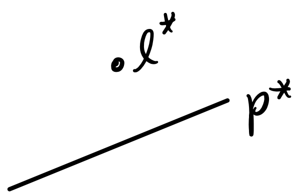
$\longleftrightarrow$



incidence preserving



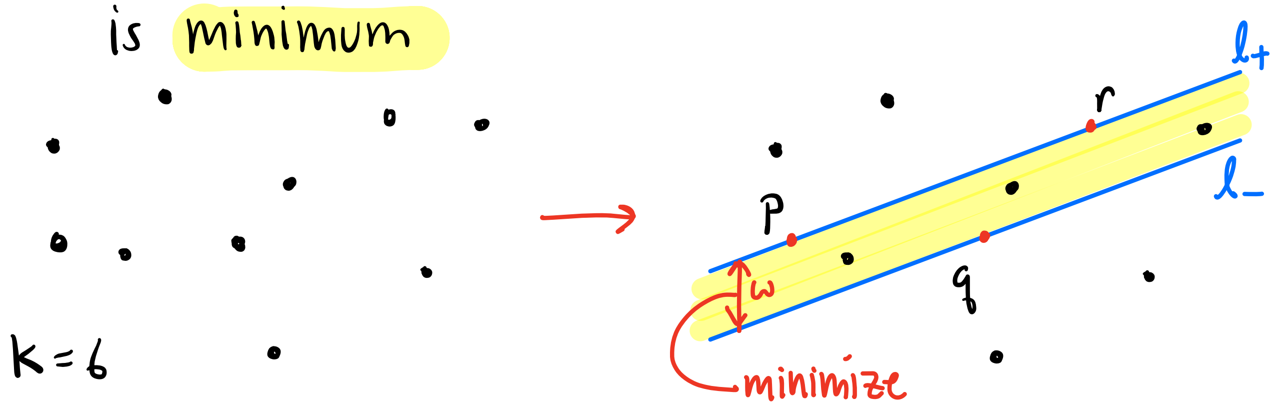
$\longleftrightarrow$



order reversing

Narrowest k-corridor:

- Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$ and integer $3 \leq k \leq n$, find pair of parallel (non vertical) lines that enclose k pts so that vertical distance between lines is minimum



Primal form:

- Let ℓ_+ + ℓ_- be upper + lower lines of "slab"

parallel ⇒ same slope

$$\ell_+ : y = ax - b_+ \qquad b_+ \leq b_-$$
$$\ell_- : y = ax - b_-$$

order reversed due to negation

- Vertical width: $w = b_- - b_+$
- k pts of P lie on or between ℓ_- + ℓ_+

Local optimality:

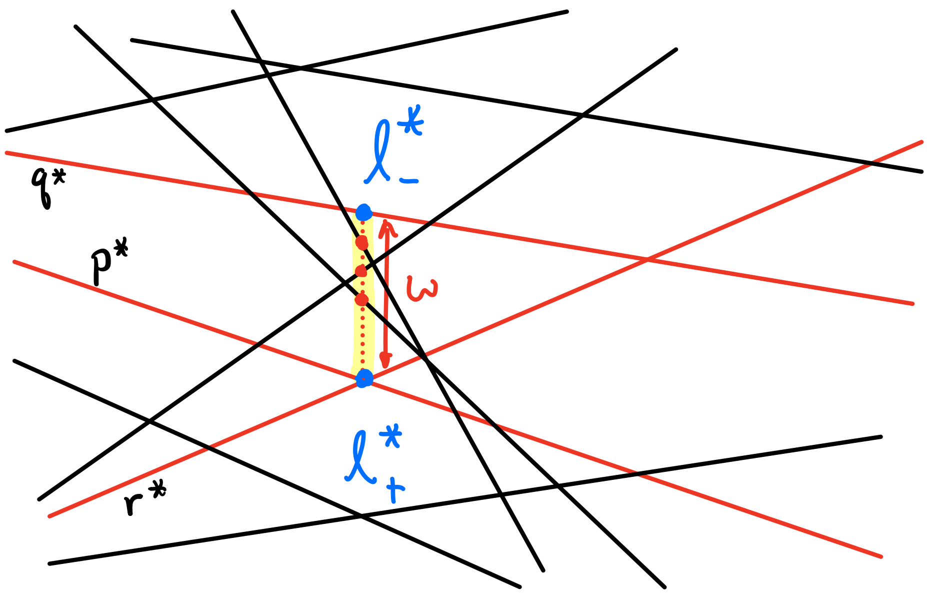
3 pts of P will lie on ℓ_+ + ℓ_-, 2 on one edge + 1 on other

- If 0, 1, or 2 can make width smaller
- If 4 or more – not gen'l position

vertical line segment

Dual form:

- ℓ_+^* + ℓ_-^* are pts (a, b_+) + (a, b_-)
- vertical distance $b_- - b_+$
- k lines of P^* pass through or between these pts



Local optimality:

3 lines will pass through ℓ_-^* + ℓ_+^* with 2 on one side + one on other

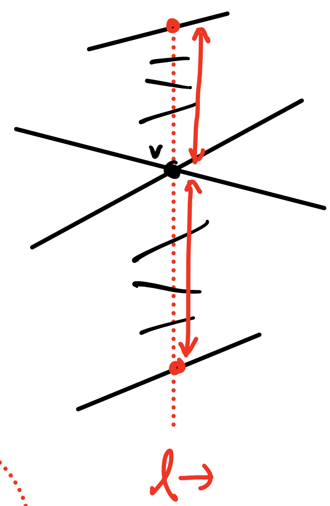
Narrowest-Corridor(P, k):

(1) $P^* \leftarrow$ dual lines of P

(2) Plane sweep through P^*.

(3) On arriving at each vertex v, compute vertical distance to lines k-2 above + k-2 below

(4) Return smallest such distance



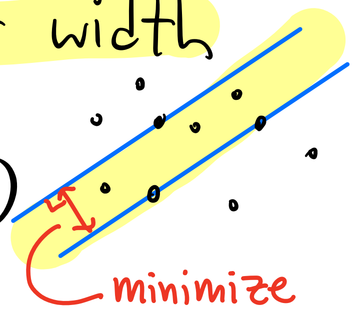
Correctness: (Argued above)

Can access in $O(1)$ time since sweep line can be stored in array

Time: $O(n^2 \log n)$ time + $O(n)$ space

↳ can reduce to $O(n^2)$ by topol. plane sweep.

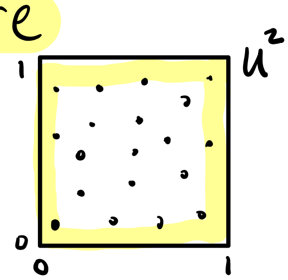
Aside: It is easy to generalize this to minimize perpendicular width (Just apply a correction factor when computing widths)



Halfplane Discrepancy:

Let $U^2 = [0,1]^2$ denote the unit square

Given n pts $P = \{p_1, \ldots, p_n\} \subset U^2$, how close is P to being uniformly distributed over U^2?



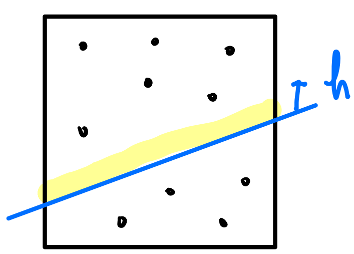
Idea:

For any halfplane h, let

$$\mu(h) = \text{area}(h \cap U^2) \qquad [0 \le \mu(h) \le 1]$$

the fraction of P in h

$$\mu_P(h) = |h \cap P| / |P| \qquad [0 \le \mu_P(h) \le 1]$$



$\mu(h) = 2/3 = 0.666\ldots$

$\mu_P(h) = 6/10 = 0.6$

If P is uniformly distrib., we expect

$$\mu(h) \approx \mu_P(h) \quad \forall h$$

To measure how uniform is P, define:

$$\Delta(P) = \max_h |\mu(h) - \mu_P(h)|$$

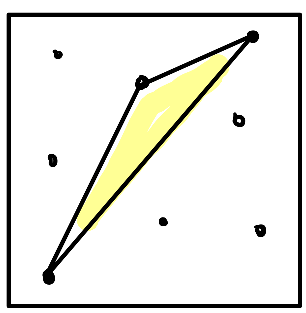
$[0 < \Delta(P) \leq 1]$

can't be perfect

Called the halfplane discrepancy of P

Questions:

- Given $P \subset U^2$, what is $\Delta(P)$?
- How low can $\Delta(P)$ be for any set of size n?
- How to generate optimally uniform set P_{OPT} of a given size n? ($\Delta(P_{OPT})$ is min. possible)
- Other measures of discrepancy?
  - Triangle discrepancy
  - Heilbronn's Triangle Problem: Given any set of n pts P in U^2, how large can the min area triangle be?



Conj: $O(1/n^2)$

Open for a century!

Computing $\Delta(P)$ for a set $P \subset U^2$.

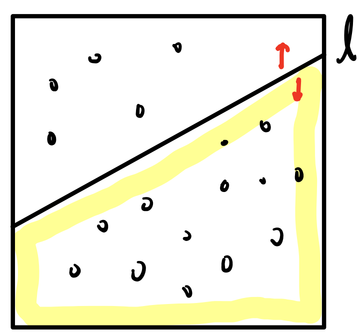
Key: Identify $O(n^2)$ candidates for halfplane that maximizes discrepancy.
- Compute discrepancy for each
- Return the max

Lemma: Given pt set P, let h be halfplane of max discrepancy. Let ℓ be h's bounding line. Either:
(i) ℓ passes through pt $p_i \in P$, and p_i is midpoint of $\ell \cap U^2$
(ii) ℓ passes through two pts of P.

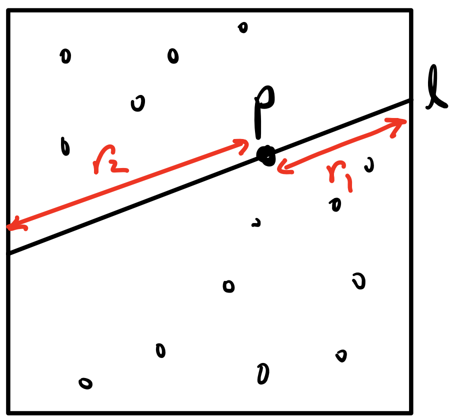
Proof:

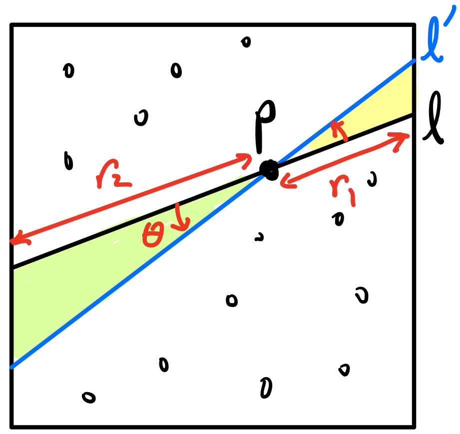
Approach: Consider any line ℓ. We'll show unless it satisfies (i) or (ii) we can perturb it to increase discrepancy.

Case 1: ℓ passes through no pt of P – perturbing ℓ up or down increases discrepancy.



Case 2: ℓ passes through a pt $p \in P$, but p is not midpt of $\ell \cap U^2$





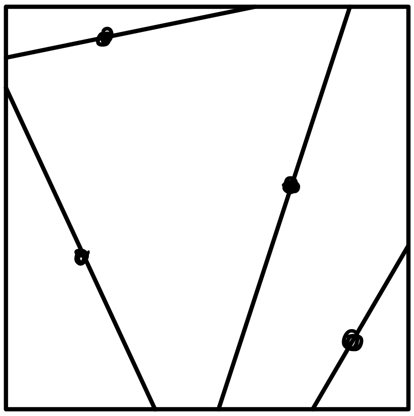
p splits $\ell \cap U^2$ into two segments of lengths $r_1 + r_2$. Since p is not midpt, may assume w.l.o.g. $r_2 > r_1$

If we rotate ℓ by small angle θ about p we increase/decrease area by $\sim$

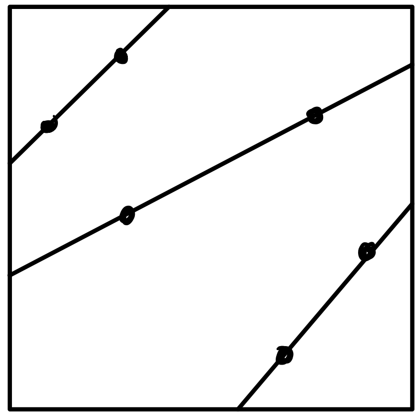
$$r_2^2 \cdot \theta - r_1^2 \theta = (r_2^2 - r_1^2)\theta > 0$$

Some small rotation will increase discrepancy. $\square$

Type (i)



Type (ii)



Computing $\Delta(\underline{P})$:

Type (i):

- for each $p_i \in \underline{P}$, compute lines ℓ s.t. p_i on midpt of $\ell \cap U^2$
  - Count no. of pts on either side of ℓ

$\rightarrow$ n pts ; $O(1)$ lines each ; $O(n)$ time to count $\Rightarrow O(n^2)$ time

Type (ii):

- Dualize $\underline{P}$ to $\underline{P}^*$
- Perform plane sweep of arrangement $\mathcal{A}(\underline{P}^*)$
- For each vertex of arrangement maintain no. of lines above + below on sweep line
- Compute discrepancy in $O(1)$ time for each vertex

$\rightarrow O(n^2)$ vertices

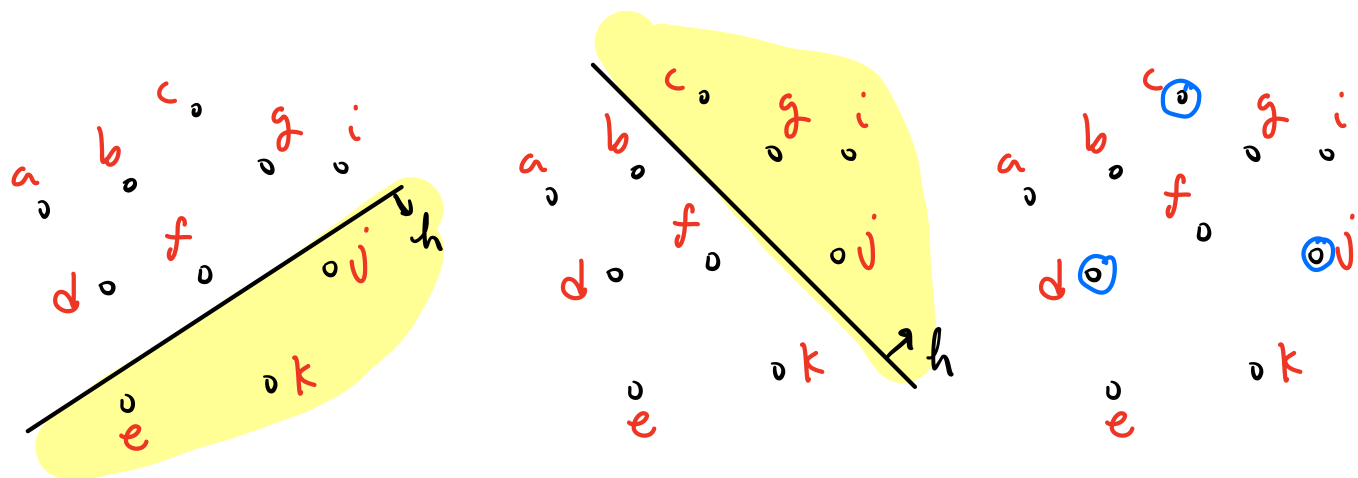
Can maintain counts in $O(1)$ time

$\Rightarrow O(n^2 \log n)$ time + $O(n)$ space

$O(n^2)$ by topol plane sweep

Computing k-sets:

Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$ and integer k, $1 \le k \le n-1$, a **k-set** is a k-element subset of P of the form $P \cap h$, for some halfplane h.



$\{e, k, j\}$ is a 3-set

$\{c, g, i, j\}$ is a 4-set

$\{c, d, j\}$ is not a 3-set

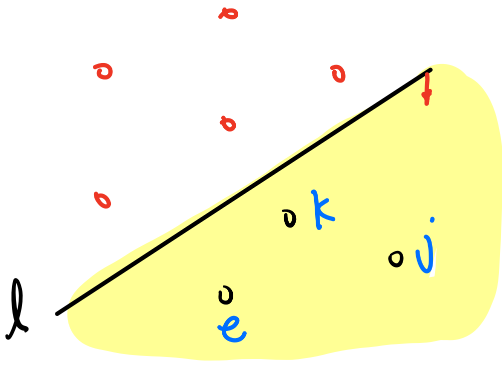
Problem: Given P and k, enumerate all k-sets of P.

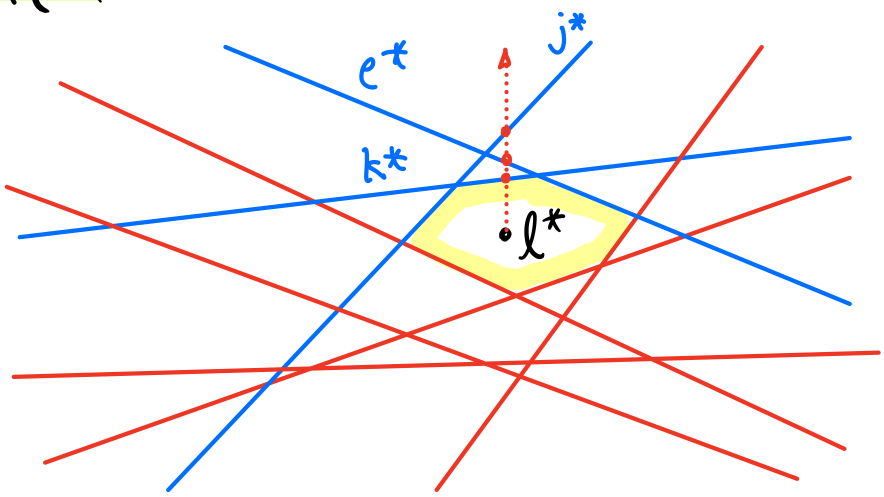
How many? Naive $\le \binom{n}{k} = O(n^k)$

Better $\le \binom{n}{2}$ (see below)

Best theoretic bounds: $O(n \log k) \,..\, O(n k^{1/3})$

Dual equivalent?





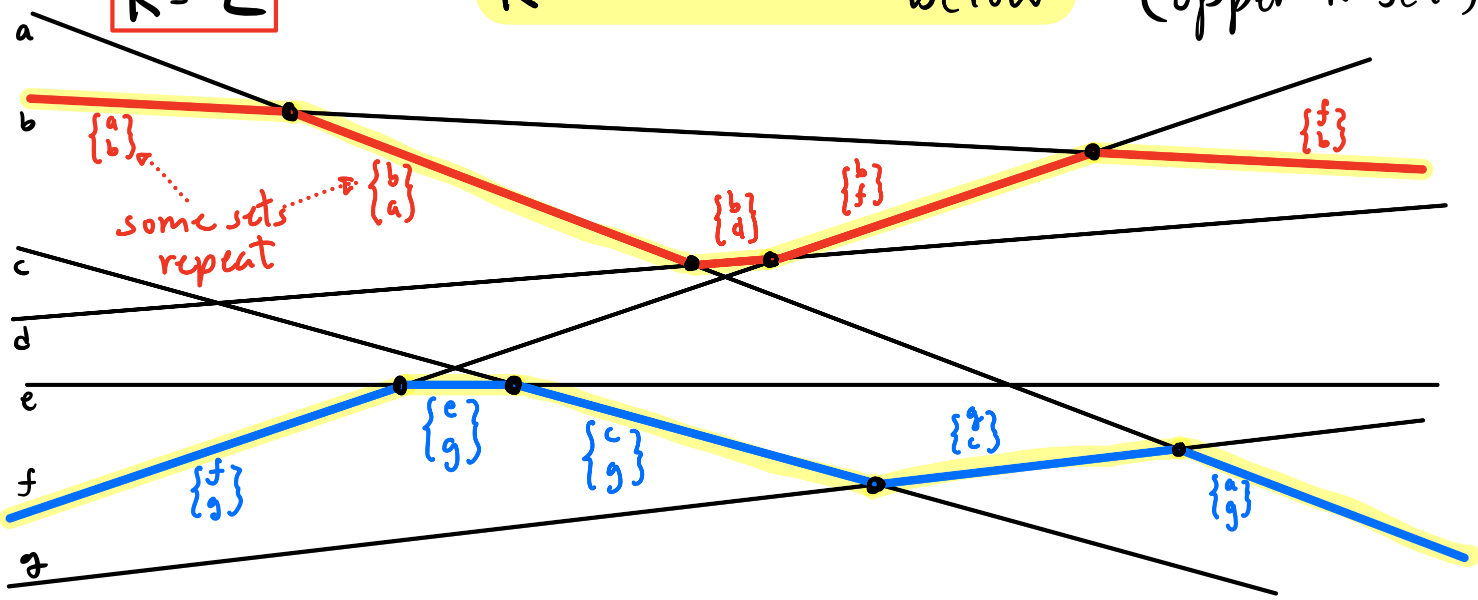
By order reversal:

k-pts of P lie below ℓ $\iff$ k-lines of P^* pass above ℓ^*

Approach:

- Traverse the arrangement $A(P^*)$
- Identify all edges with
  - k lines on or above (lower k-set)
  - k " " " below (upper k-set)

$\boxed{\begin{matrix} n=7 \\ k=2 \end{matrix}}$



Level: Given an arrangement of n lines $A(L)$, for $1 \leq k \leq n$, define level k, $\mathcal{L}_k$, to be set of pts in $A(L)$ with

$\leq k-1$ lines (strictly) above
$\leq n-k$ lines (strictly) below

In above fi gure, we have shown $\mathcal{L}_2$ and $\mathcal{L}_6$

Obs: By applying plane sweep through $A(L)$, we can construct all levels in time $O(n^2)$

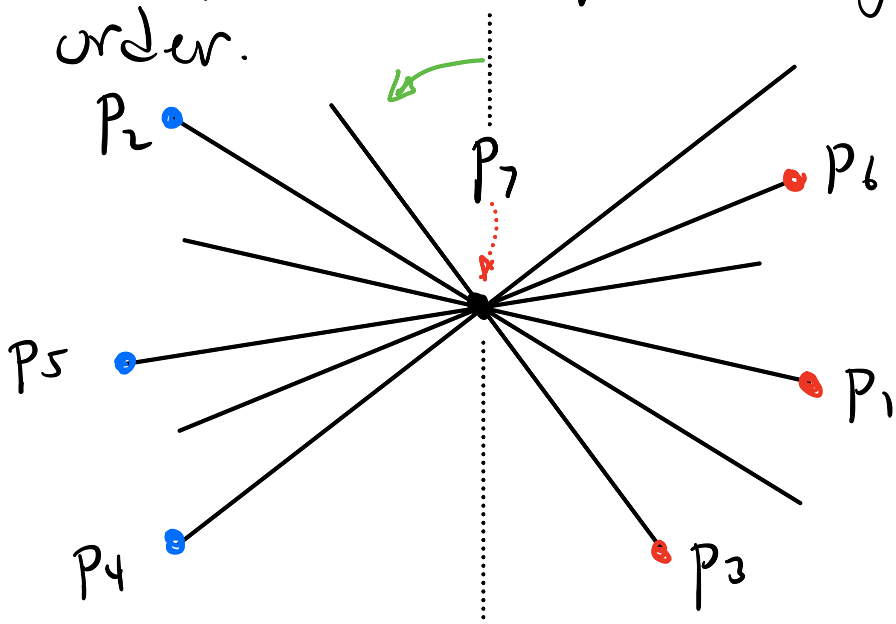
$\Rightarrow$ Can identify all k-sets of P in time $O(n^2)$ by sweeping $A(P^*)$ + extracting levels $\mathcal{L}_k$ + $\mathcal{L}_{n-k+1}$

Note: To actually list the sets adds addional k factor, total $O(k \cdot n^2)$

Avoid duplicates? Exercise

Sorting angular sequences:

Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$, for each p_i, sort the remaining $n-1$ pts around p_i in angular order.



$$\begin{bmatrix} 2 \\ 5 \\ 4 \\ 3 \\ 1 \\ 6 \end{bmatrix}$$

Naive: $O(n(n \log n)) = O(n^2 \log n)$
Sort angles for each point

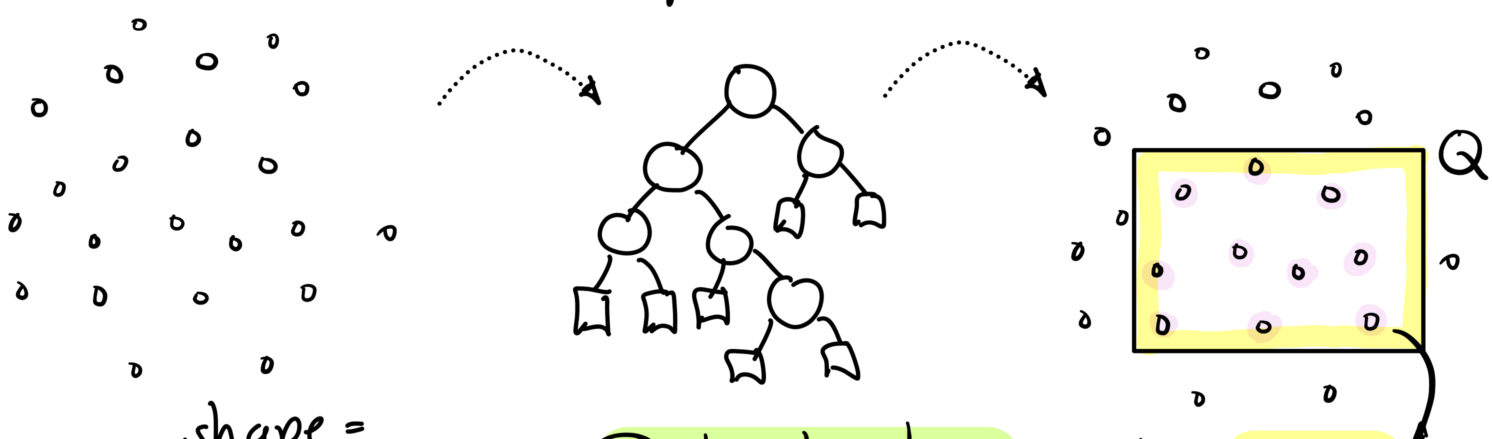
Better: $O(n^2)$ using arrangements.

[See lect. notes for details]

CMSC 754 - Computational Geometry
Lecture 14 - Orthogonal Range Search + kd-Trees

Range Searching: (Data structure problem)

- Given a **point set** $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$
- Given a **class of shapes** (e.g. rectangles, balls, triangles, halfspaces)
- Build a **data structure** so that:
  - Given any **query region Q** from the class, quickly identify the points of P in Q



P + shape = rectangles

Data structure

Ans: 10 pts of P in Q

- **Points** (data structure) is (relatively) **static**
- **Queries** must be answered **fast!** (sublinear time)

What types of Queries?

- **Emptiness**: Any pts of P in Q?
- **Counting**: How many? $|P \cap Q|$
- **Weighted count**: Each $p \in P$ has weight $w(p)$. Return total weight

$$\sum_{p \in P \cap Q} w(p)$$

- **Semigroup weight**: Any commutative + associative function of wts:

Eg. max-query: $\max_{p \in P \cap Q} w(p)$

- **Reporting**: List the pts of $P \cap Q$
- **Top-k**: List just the highest k pts of $P \cap Q$ based on weights

Complexity Bounds:

Space: Total space needed to store points + data structure

Query time: Time needed to answer a query

Construction time: Time to build structure
Common: (Space bound)$\cdot O(\log n)$

"**Gold standard**": $O(n)$ space
$O(\log n)$ query time
$O(n \log n)$ constr. time

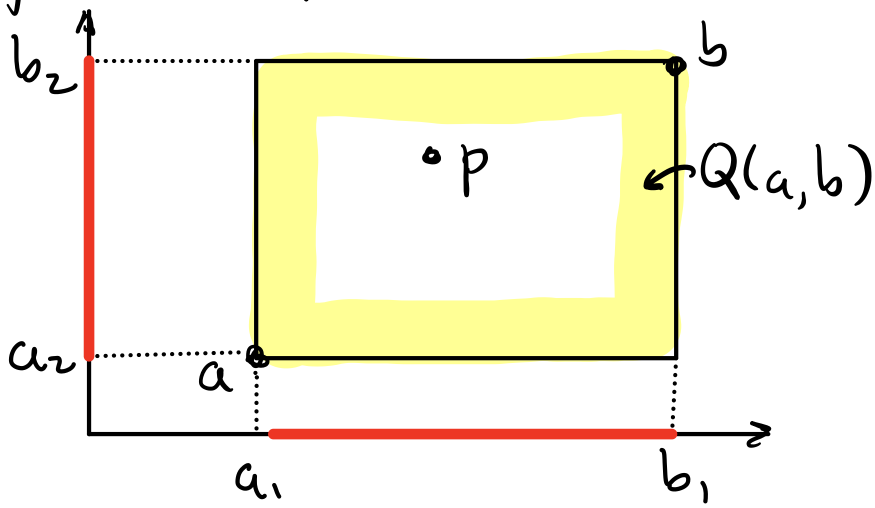
Many geometric structures are **inferior** w.r.t. **space**: $O(n \log^2 n)$
$O(n \log^d n)$ in $\mathbb{R}^d$
$O(n^2)$

or **Query time**:
$O(\log^2 n)$
$O(\sqrt{n})$
$O(n^{1-1/d})$ in $\mathbb{R}^d$

Orthogonal Range Queries:

Query region is axis-aligned rectangle

Eg. Given pts $a, b \in \mathbb{R}^d$ s.t. $a_i < b_i \; \forall i$



Query rectangle is product of intervals:

$$Q(a,b) = \{ p \in \mathbb{R}^d \mid a_i \leq p_i \leq b_i \}$$

$$= [a_1, b_1] \times \dots \times [a_d, b_d]$$

Common in database queries:

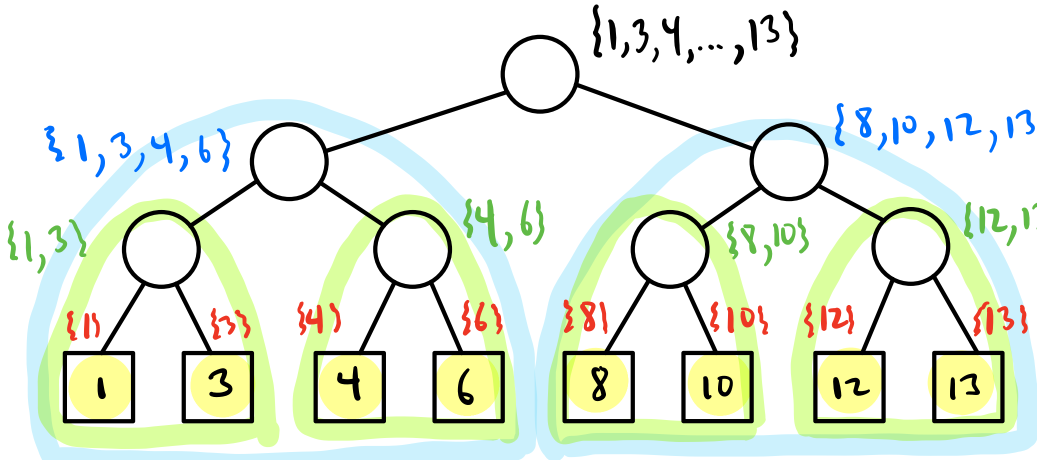
How many patients with age $\in [25, 35]$
weight $\in [100, 200]$
blood pressure $\in [80, 120]$

General approach to answering range queries:

- Too slow to count pts one by one
- Too much space to precompute answer to every possible query

- Canonical subsets:
  Carefully select an (ideally small) collection of subsets of P so that the answer to any query can be formed as (disjoint) union of a small number of subsets.

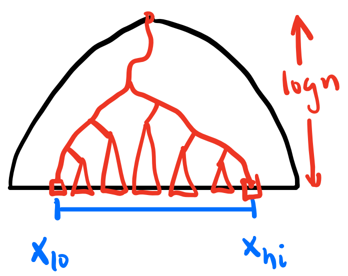
Example: 1-dimensional range query
$P = p_1 < p_2 < \dots < p_n$ in $\mathbb{R}$
- Store P as leaves of a balanced tree
- Leaves of each subtree form canonical set



- The answer to any 1-dim range query can be expressed as the disjoint union of $O(\log n)$ canonical subsets.

- Example: $Q = [x_{lo}, x_{hi}] = [2, 23]$

  $P \cap Q = \{3\} \cup \{4,7\} \cup \{9,12,14,15\} \cup \{17,20\} \cup \{22\}$



  - Cover the range with maximal subtrees
  - Take union of the assoc. canonical subsets
  - $O(\log n)$ subtrees always suffice.
  - $O(n)$ nodes $\Rightarrow$ $O(n)$ canon. subsets

Compose the Answer to Query from Subsets:

Counting query: Node stores # of leaves

Weighted count: Node stores total weight of leaves

Max query: Node stores max of all weights in leaves

...

Can answer queries in $O(\log n)$ time by combining subtree results (assuming you can identify the canon. subsets for query + precompute info.)

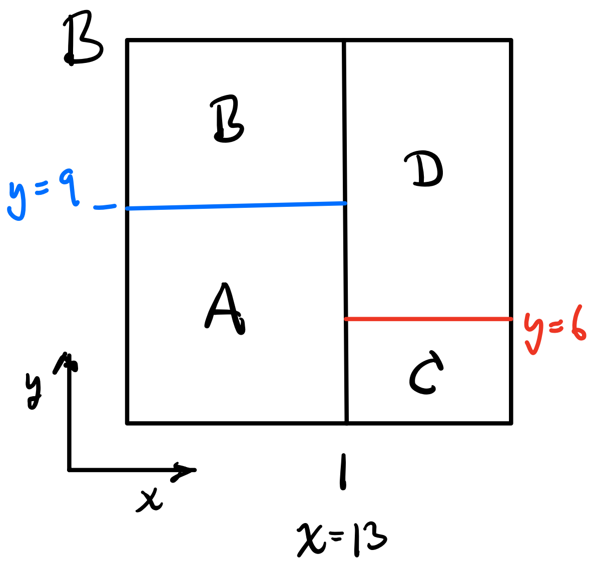
Kd-Trees: A natural generalization of 1-d trees to higher dim

1-d tree, 2-d tree, ..., k-d tree

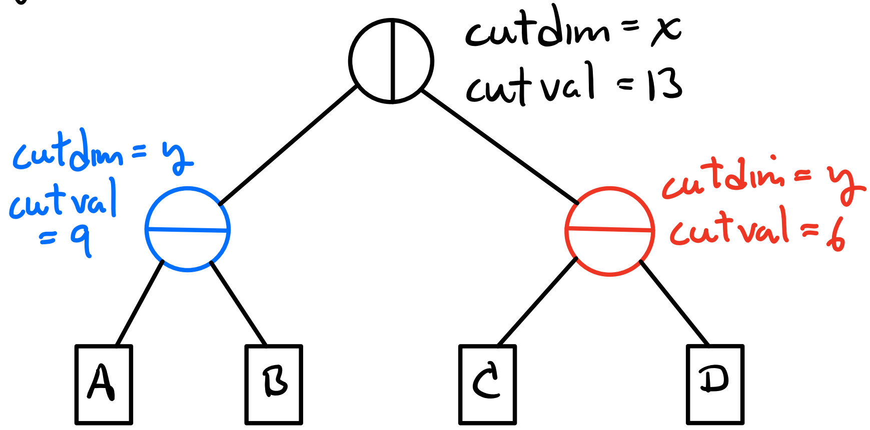
Jon Bentley (1975)

Numerous variants - we present one

- Assume have large **bounding box** B containing P
- **Recursively split space** by axis-orthogonal hyperplane
  - **cutting dimension**: which axis
  - **cutting value**: where to cut



Spatial subdivision

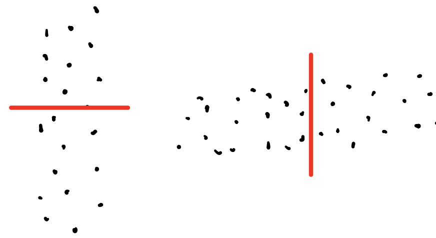


Tree structure

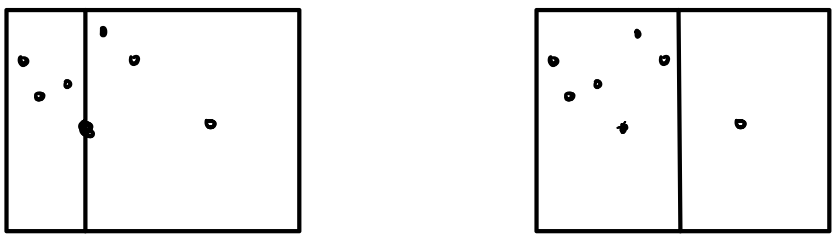
Cell: Each tree node represents a rectangular region

Design choices:

- Where are points stored?
  - internal nodes (used for splitting)
  - external nodes (leaves)
    ↳ Permits more flexibility in where to split
- How is cutting dim chosen?
  - alternate: x, y, x, y, … or x, y, z, x, y, z, …
  - select based on point distribution



- How is cutting value chosen?
  - median (balanced height)
  - mid pt (geom. balanced)



Our structure:

- Points stored at leaves (external nodes)
- Alternate splitting axes
- Split at median

Construction:

Tree can be built in $O(n \log n)$ time

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

find median splitting coord

recursively build subtrees

$$= O(n \log n)$$

Slight improvement: Presort the points d times into d lists - one for each coordinate + cross-link entries

- Faster in practice

Space: $O(n)$

- n leaves (one per point)
- $(n-1)$ internal nodes
- $O(1)$ info per node

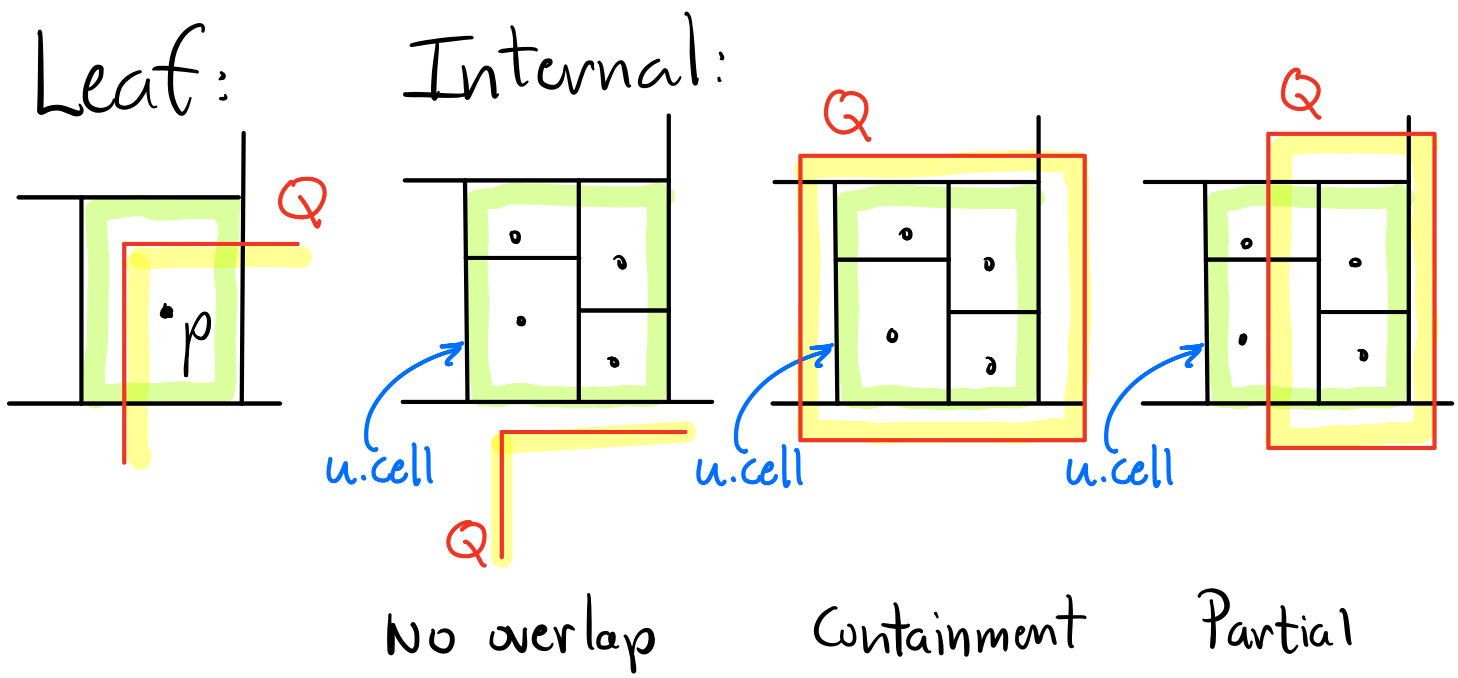
Range Search:

Key: If node's cell does not overlap Q → Don't visit

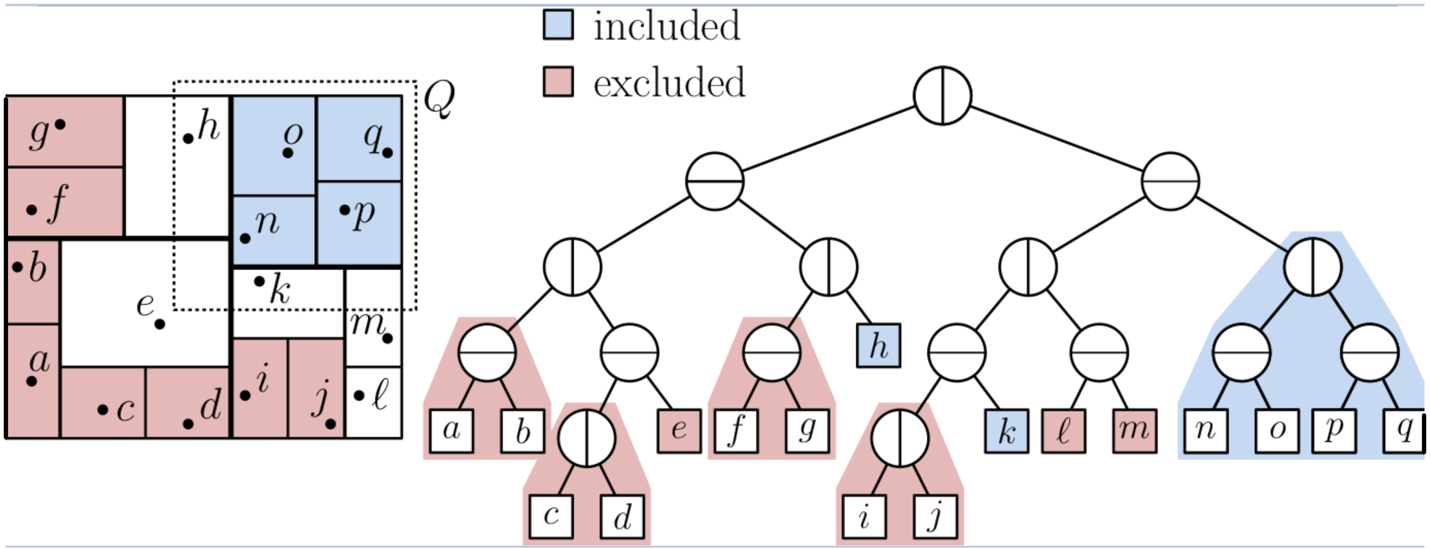
If node's cell completely in Q → count all its pts

Algorithm: Weighted range count in kd-tree

```
range-count ( Rect Q , KdNode u )
    if ( u is leaf )
        if ( u.point ∈ Q ) return u.point.weight
        else return 0
    else ( u is internal )
        if ( u.cell ∩ Q = ∅ )
            return 0   ( no overlap )
        else if ( u.cell ⊆ Q )
            return u.weight   ( total weight )
        else
            return range-count ( Q , u.left )
                 + range-count ( Q , u.right )
```



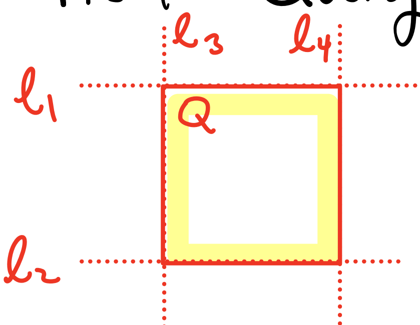
Example:



Query Time:

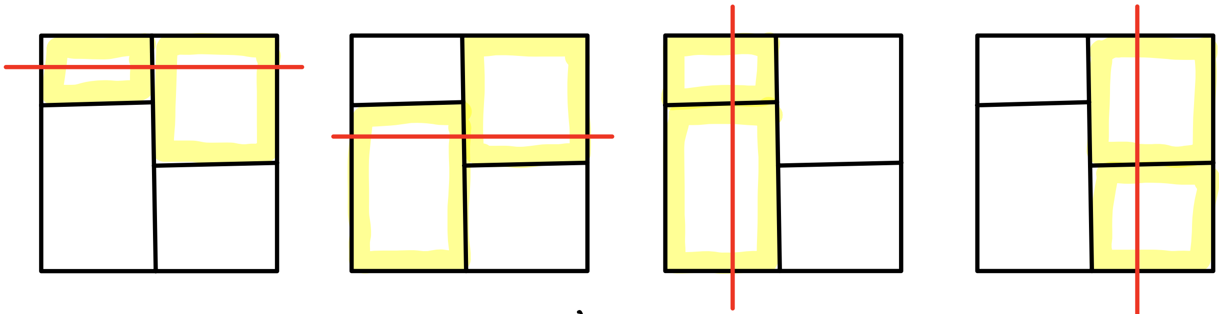
Thm: Given a height-balanced kd-tree in $\mathbb{R}^2$ using alternating splitting axes, orthog. counting queries can be answered in $O(\sqrt{n})$ time.
[Reporting queries in time $O(k + \sqrt{n})$, where k = # of points reported]

Proof: Query rectangle bounded by 4 lines



We'll show that each line stabs $\leq \sqrt{n}$ cells of tree $\Rightarrow O(4 \cdot \sqrt{n})$

Key: Because we alternate cutting dim for every 2 levels of tree, any axis parallel line can stab at most 2 out of 4 grandchild cells



Since we use balanced splitting

parent	n pts
child	$n/2$ pts
grandchild	$n/4$ pts

$\Rightarrow$ Query time:

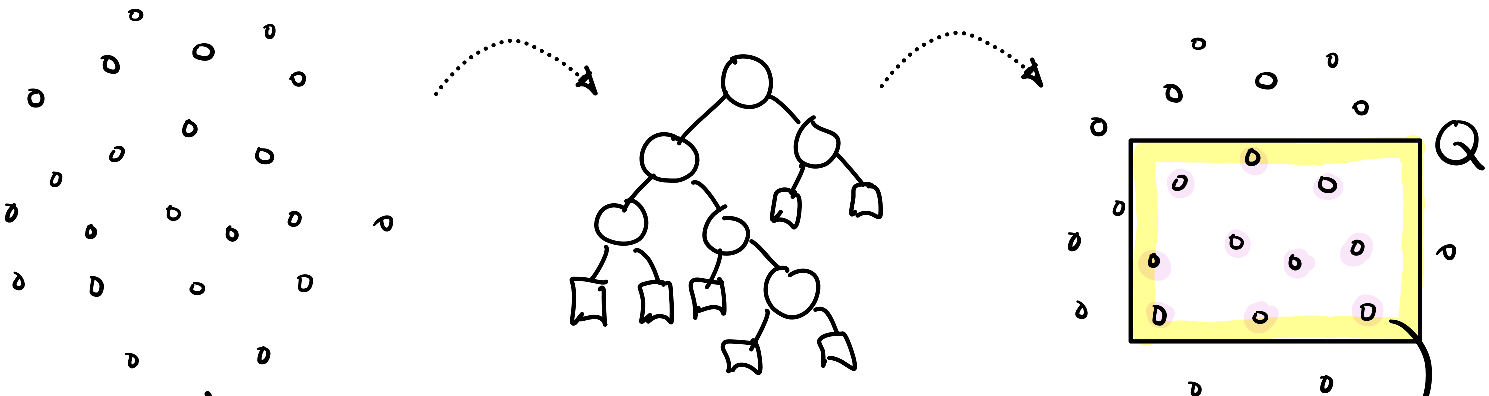
$$T(n) = \underbrace{2T(n/4)}_{\text{recurse on 2 of 4 grandchildren}} + \underbrace{1}_{\text{constant time per cell}}$$

$$= O(\sqrt{n})$$ [see lect. notes for details]

CMSC 754 - Computational Geometry
Lecture 15 - Orthogonal Range Trees

Recall: Range Search:

Given a set of n pts $P = \{p_1, \ldots, p_d\} \subseteq \mathbb{R}^d$, and class of shapes (range space) preprocess P to answer range queries: Given shape Q, count/report the pts in $P \cap Q$.



P + shape = rectangles

Data structure

Ans: 10 pts of P in Q

Last lecture: kd-trees

$O(n)$ space / $O(n \log n)$ build time

$O(\sqrt{n})$ query time (in $\mathbb{R}^2$)

$O(n^{1-1/d})$ in $\mathbb{R}^d$

Today: Orthogonal Range Trees

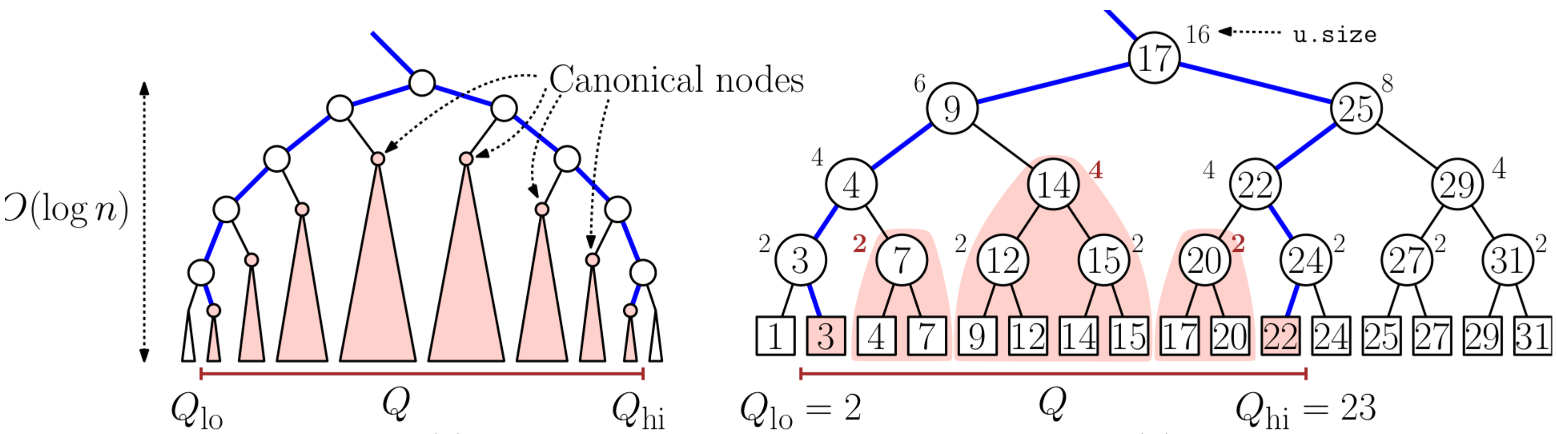
→ $O(\log^d n)$ query time

+ Layered Data Structures

→ $O(n \log^{d-1} n)$ space

1-Dimensional Range Tree: (Review)

- Given set of scalars: $\underline{P} = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$
- Store as leaves in balanced search tree → $O(n)$ space
  → $O(n \log n)$ construct. time
- Each node u stores num. of leaves: u.size
- Given query interval $Q = [Q_{lo}, Q_{hi}]$
  - Identify $O(\log n)$ maximal subtrees that cover Q
  - Add up sizes for all these nodes



Query answer = 1 + 2 + 4 + 2 + 1 = 10

Range counting algorithm:

Node u:

- $u.\text{point}$: point p_i (if u is leaf)
- $u.x$: split value (if u internal)
- $u.\text{size}$: # leaves (if u internal)
- $u.\text{left}$, $u.\text{right}$: children

---

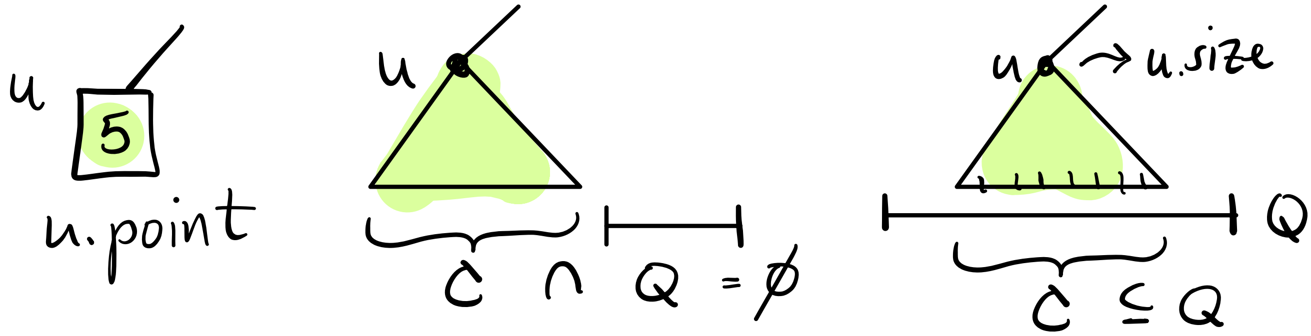
range1Dx(Node u, Range Q, Interval $C = [x_0, x_1]$)

if (u is leaf)
  return $\begin{cases} 1 & \text{if } u.\text{point} \in Q \\ 0 & \text{o.w.} \end{cases}$

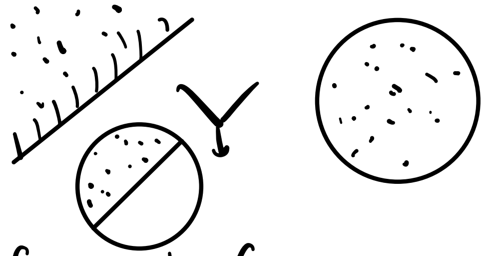
else if ($C \cap Q = \emptyset$) (no overlap)
  return 0

else if ($C \subseteq Q$) (contained)
  return $u.\text{size}$

else (recurse)
  return range1Dx($u.\text{left}$, Q, $[x_0, u.x]$)
    + range1Dx($u.\text{right}$, Q, $[u.x, x_1]$)



Multi-Layered Structures:



Suppose your ranges are formed from composing multiple (independent) queries:

Eg. Find all patients of

- age between 25..35 : Q_1
- weight $\leq$ 200 lbs : Q_2
- blood pressure $\geq$ 100 : Q_3

Idea: Design a datastructure for each query type + "merge them"

How to merge?

- Build range structure for age for P

  $\Rightarrow$ Canonical subsets: $P_1, P_2, \ldots, P_m$

- For each P_i, build a range structure for weight

  $\Rightarrow$ Canonical subsets: $P_{i1}, P_{i2}, \ldots$

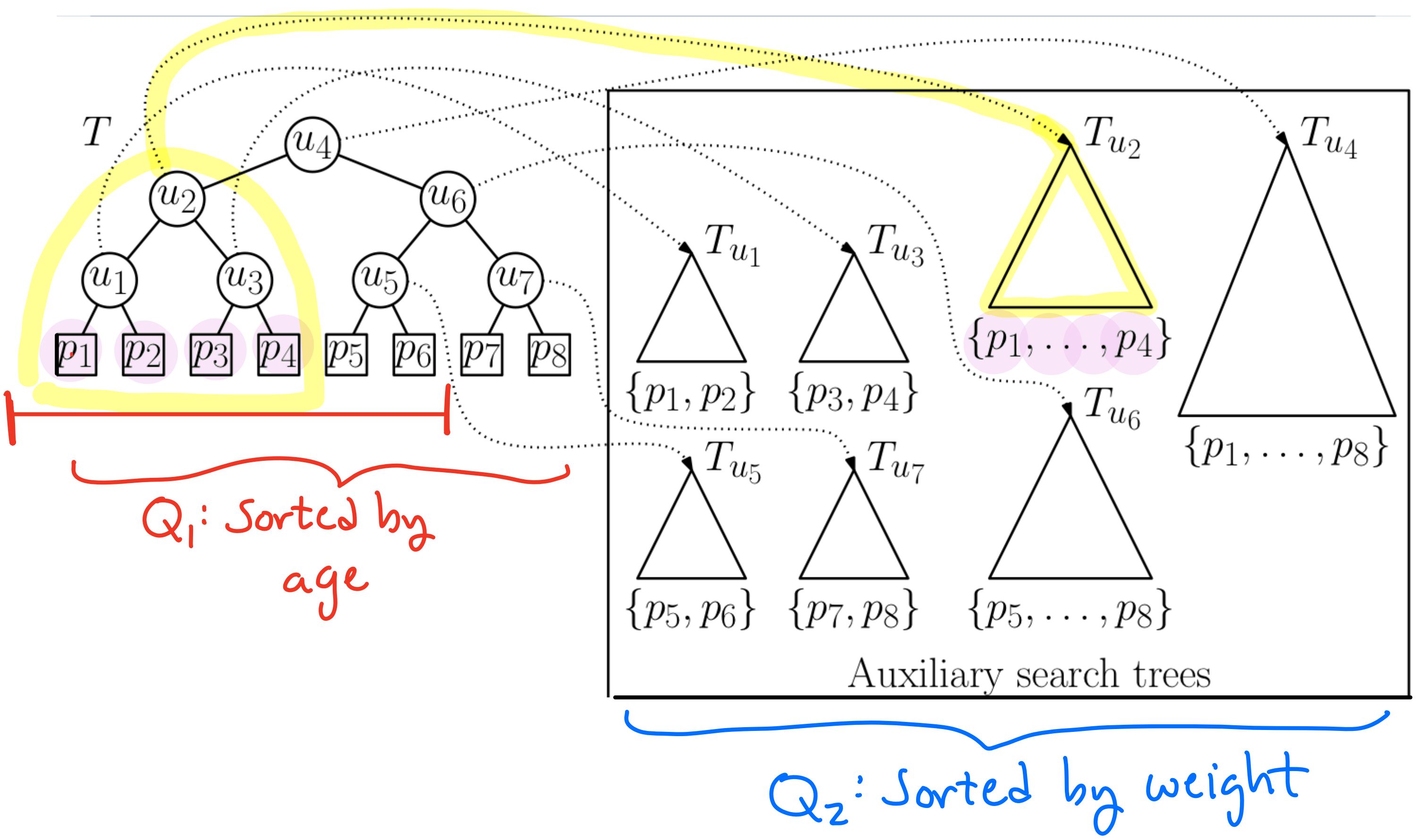
- For each P_{ij}, build range structure for blood pressure

  ⋮

Multi-Layered Search Tree:

- Store data in leaves of tree
- Each node's canonical subset consist of it's leaves
- For each node, build a search tree for its canonical subset
  called its auxiliary tree

Example:



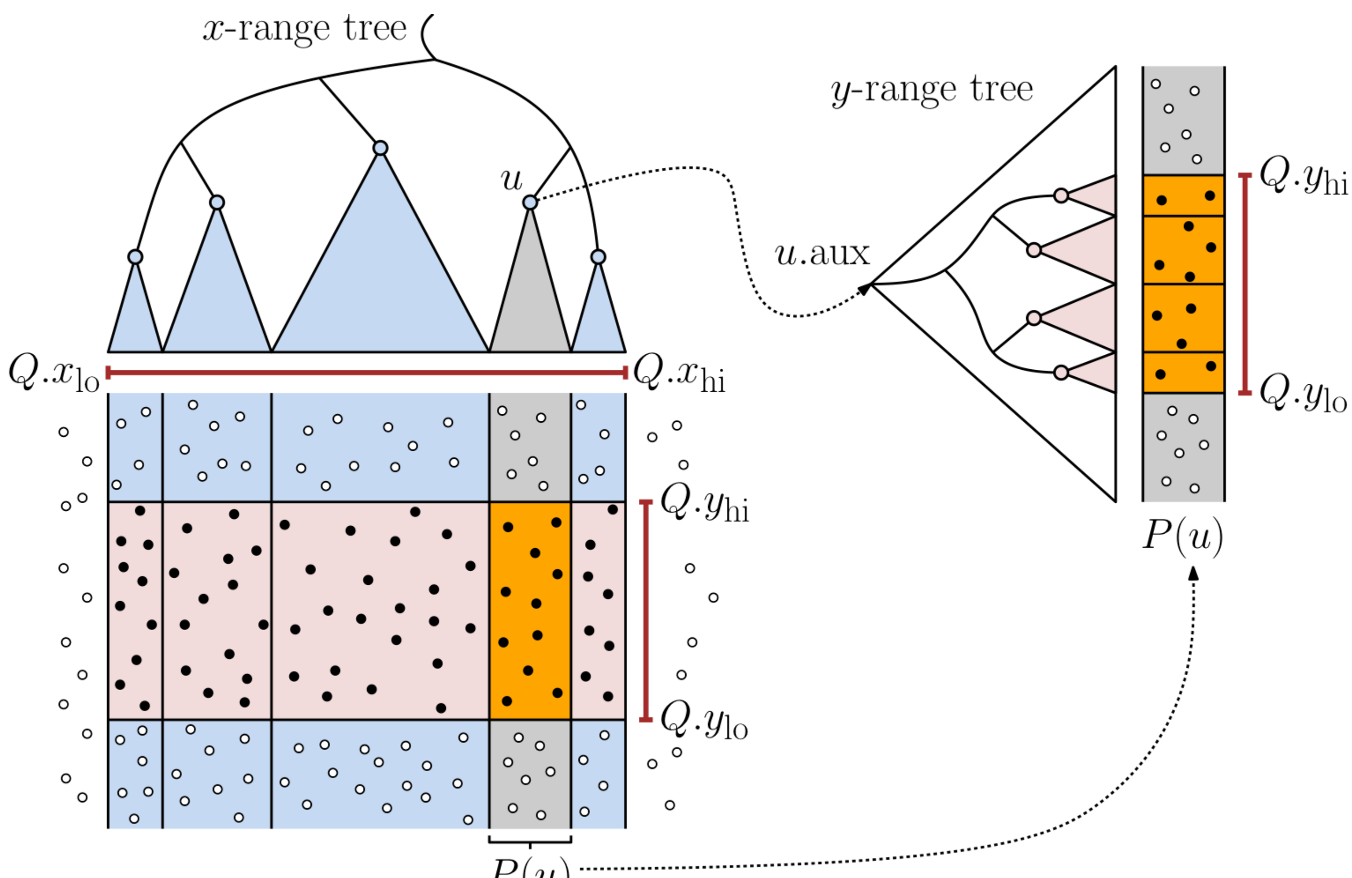
Orthogonal (2-d) Range Tree:

- Given points $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$
- Build a 1-d range tree for P based on x only (data in leaves)

Main tree

- For each internal node u, let $P(u)$ be points in its leaves (canon. subset)
  - Build a 1-d range tree for $P(u)$ sorted by y-coords.

Aux. tree for u



To process query $Q = [Q_{lo}, Q_{hi}]$
$= [Q_{lo.x}, Q_{hi.x}] \times [Q_{lo.y}, Q_{hi.y}]$

- Apply 1-d search in main tree with query $[Q_{lo.x}, Q_{hi.x}]$ to identify $O(\log n)$ maximal subtrees
- For each root u of one of these max. subtrees apply 1-d search in u.aux with query $[Q_{lo.y}, Q_{hi.y}]$
- Return overall sum

```
range2D(Node u, Range Q, Interval C = [x0, x1])
  if (u is leaf)
    return { 1 if u.point ∈ Q
           { 0 o.w.
  else if (Q.x ∩ C = ∅)          (no x overlap)
    return 0
  else if (C ⊆ Q.x)              (containment in x)
    return range1Dy(u.aux, Q, [-∞, +∞])
                         search aux. tree
  else                 (recurse)
    return range2D(u.left, Q, [x0, u.x])
         + range2D(u.right, Q, [u.x, x1])
```

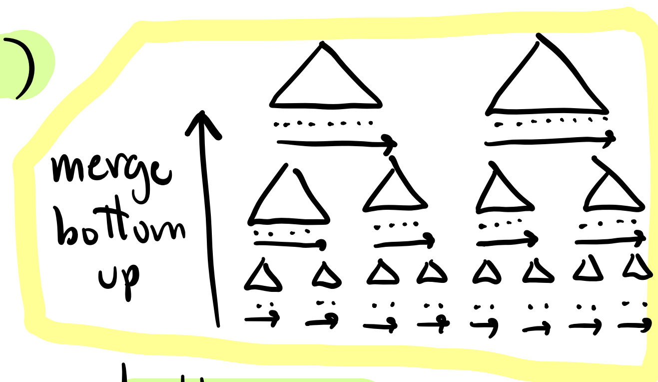
Space + Preprocessing Time:

- Since each node stores $O(1)$ data, total
  space = size of main tree + total size of aux. trees
- A tree with m leaves has size $O(m)$

$$\text{Space} = \underbrace{n}_{\text{main tree}} + \underbrace{\sum_u |P(u)|}_{u.\text{aux tree}}$$

- Main tree's height is $O(\log n)$
- Each leaf contributes a point to $u.\text{aux}$ for each of its ancestors
  - ⇒ Each point appears in $O(\log n)$ aux. trees
  - ⇒ $\sum_u |P(u)| = O(n \log n)$

⇒ Total space is $O(n \log n)$



Construction time:

Naive: $O(n \log^2 n)$

Better: Build aux trees bottom-up
- Two child sets can be merged in linear time

⇒ $O(n \log n)$

Query Time:

Main tree: $O(\log n)$ time

→ Identifies $O(\log n)$ maximal subtrees

- each has $\leq n$ points
- each searchable in $O(\log n)$ time

⇒ Total time $= O(\log n) \cdot O(\log n)$

$= O(\log^2 n)$

Thm: Using orthogonal range trees, 2-dim orthog. range (counting) queries can be answered in:

$O(n \log n)$ space

$O(n \log n)$ build time

$O(\log^2 n)$ query time → $+k$ for reporting

Thm: Using orthogonal range trees, d-dim orthog. range (counting) queries can be answered in:

$O(n \log^{d-1} n)$ space

$O(n \log^{d-1} n)$ build time

$O(\log^d n)$ query time → $+k$ for reporting

Can we do better?

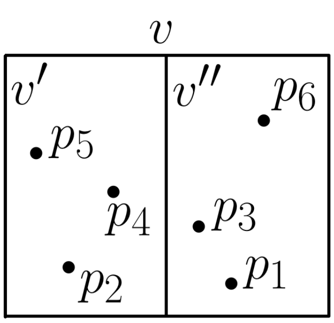
You can shave off a log n factor for query times – Cascading Search

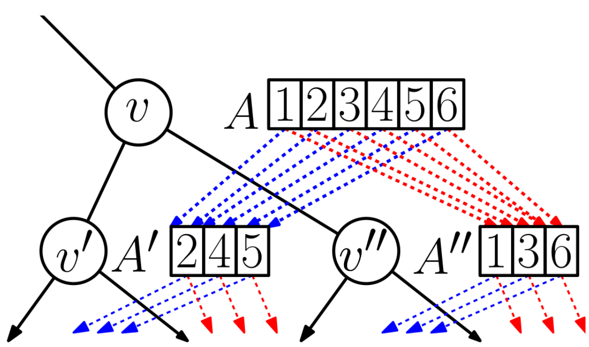
2-dim: $O(\log^2 n) \to O(\log n)$
d-dim: $O(\log^d n) \to O(\log^{d-1} n)$

(See latex notes)

Idea:

- Final aux trees can be stored as sorted arrays (trees not needed)
- Always searching for same values: Q.lo.y Q.hi.y
- Can exploit knowledge of answer in one array to find answer in another, without doing search from scratch.





CMSC 754 - Computational Geometry
Lecture 16 - Well-Separated Pair Decompositions

Geometric Approximations:

- Useful when exact computation is too costly
- Geometric inputs are "measurements" and often are uncertain.

So approximate solutions are fine.

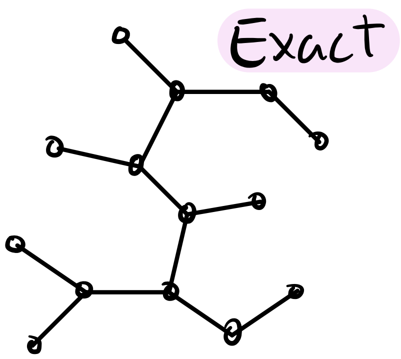
Examples:

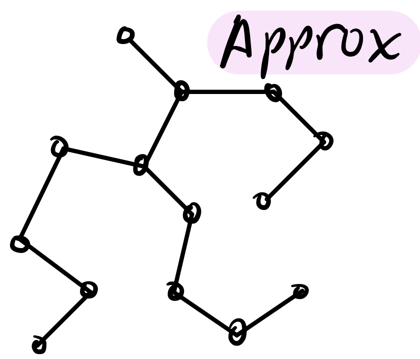
Euclidean MST of pt set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$

Exact: $O(n \log n)$ in $\mathbb{R}^2$

$O(n^{2-4/d})$ in $\mathbb{R}^d$ [Nearly quadratic]

Approx: Given $\varepsilon > 0$, compute a spanning tree of weight $\leq (1+\varepsilon) \cdot EMST(P)$





Convex Hull of a set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$

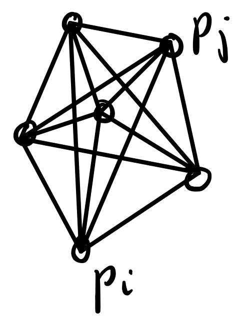
Exact: $O(n \log n)$ in $\mathbb{R}^2$

$O(n^{\lfloor d/2 \rfloor})$ in $\mathbb{R}^d$

Approx: Compute a subset $P' \subseteq P$ s.t. $\mathrm{conv}(P)$ and $\mathrm{conv}(P')$ are very similar

Exact | Approx

Well-Separated Pair Decomposition:

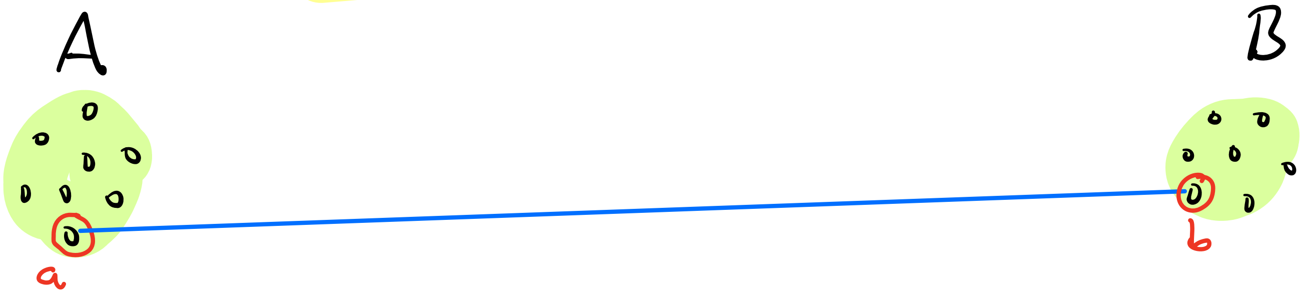


Given set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$, the **Euclidean graph** is complete graph on P, where $w(p_i, p_j) = \|p_i - p_j\|$

- Has $\binom{n}{2} = O(n^2)$ edges

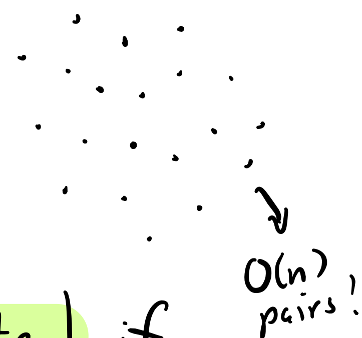
- Can we **encode** this using a structure of size $O(n)$?

Intuition: If two point clusters $A, B \subseteq P$ well separated, we can represent many edges of $A \times B$ using a single edge connecting a representative $a \in A$ + $b \in B$

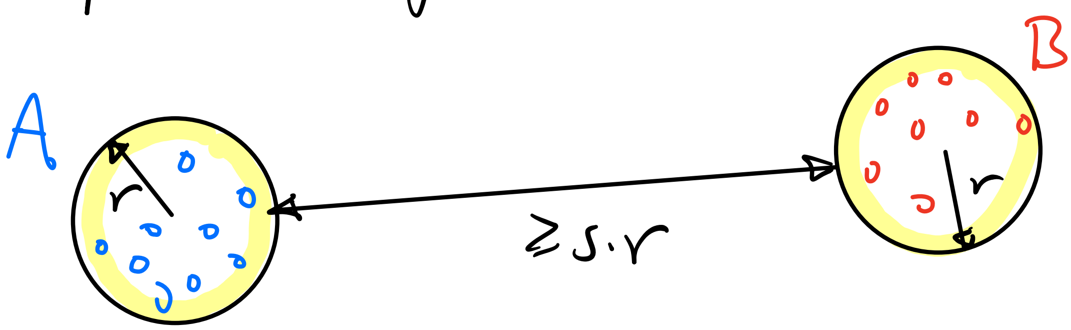


If we do this for all well-separated clusters, how many edges do we need?

Def: Given $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ and scalar $s > 0$



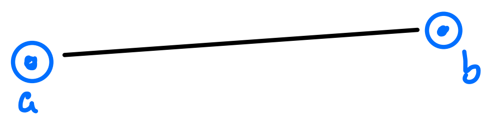
- Two sets $A, B \subseteq P$ are s-well separated if $A + B$ can be enclosed in balls of some radius r, s.t. these balls are separated by distance $\geq s \cdot r$



Obs:

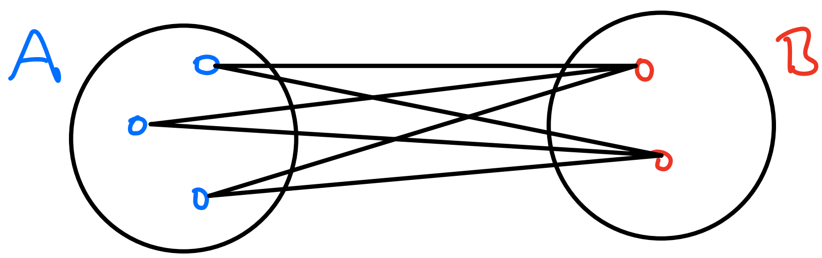
- If A + B are s-well separated, they are s'-well separated for any $0 < s' \leq s$

- Two singleton sets $A = \{a\}$, $B = \{b\}$ are s-well separated for any $s > 0$. $(a \neq b)$



Def: Given sets A, B, define

$$A \otimes B = \{\{a,b\} \mid a \in A,\ b \in B,\ a \neq b\}$$



Obs: $P \otimes P$ = set of all $\binom{n}{2}$ pairs of P.

Def: Given P + $s > 0$, an **s-well separated pair decomposition** of P (**s-WSPD**) is collection of pairs

$$\{\{A_1, B_1\}, \ldots \{A_m, B_m\}\}$$

such that:

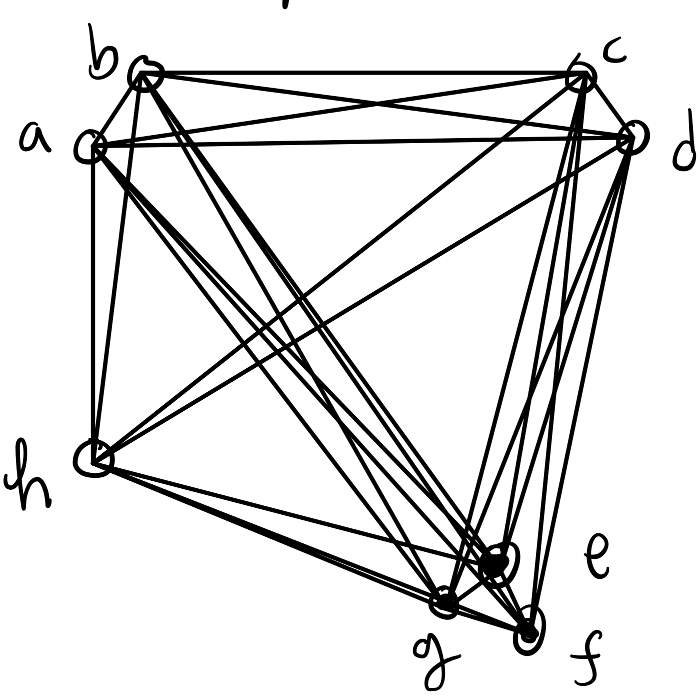
(1) $A_i, B_i \subseteq P$ for $1 \le i \le m$

(2) $A_i \cap B_i \neq \emptyset$ " " (**disjoint**)

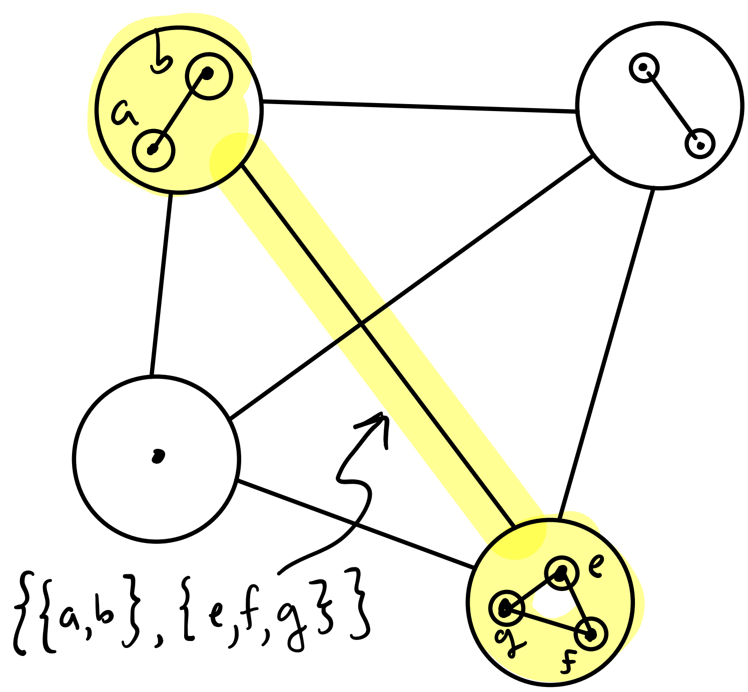
(3) $\bigcup_{i=1}^{m} A_i \otimes B_i = P \otimes P$ (**cover**)

(4) A_i + B_i are **s-well separated** for $1 \le i \le m$

28 pairs



11 well-sep pairs



Obs: For any $s > 0$ there is always a trivial s-WSPD consisting of $\binom{n}{2}$ singleton pairs.

Can we do better?

d is constant: hidden d^d

Yes! $\rightarrow$ $O(s^d \cdot n)$ pairs

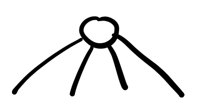
If s, d constants: $O(n)$!

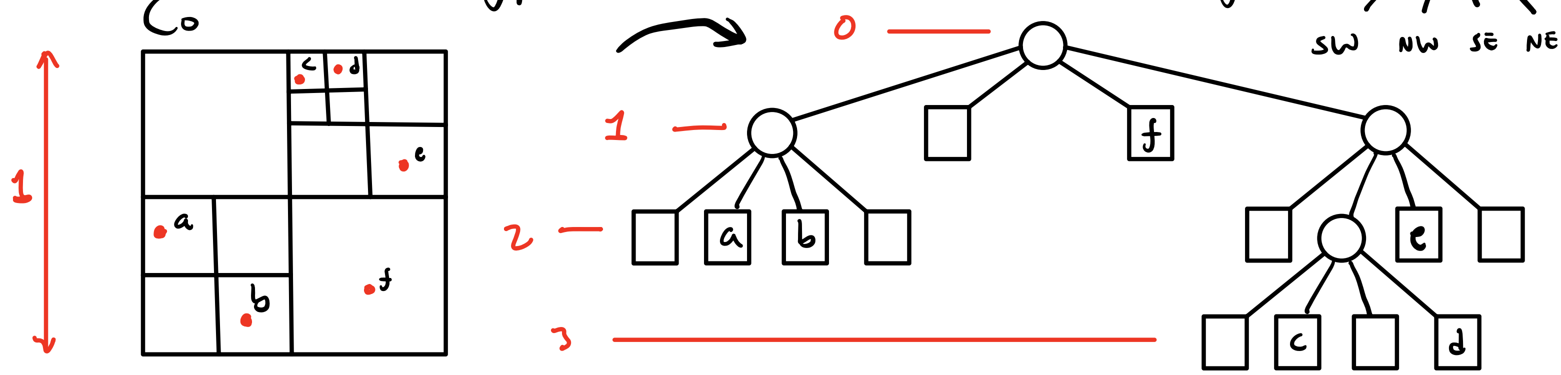
Can compute in time:
$O(n \log n + s^d n)$

Quadtrees:

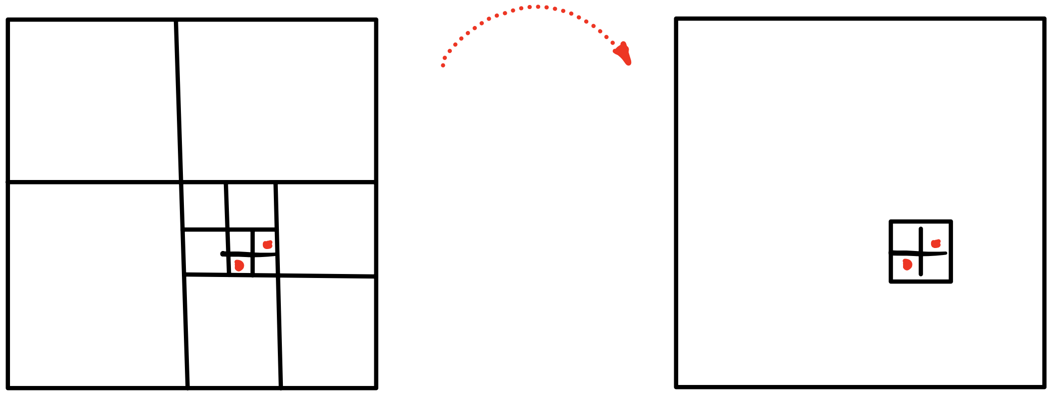
A tree storing P based on recursive subdiv. into hypercubes.

- Let C_0 be a bounding hypercube for P
- While a cell of subdivision has 2 or more pts of P, split it into 2^d hypercubes of half side length





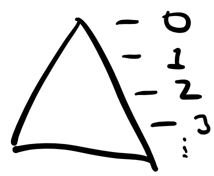
Note: A quadtree may have more than $O(n)$ nodes, but we can reduce storage to $O(n)$ by path compression. (see latex notes)



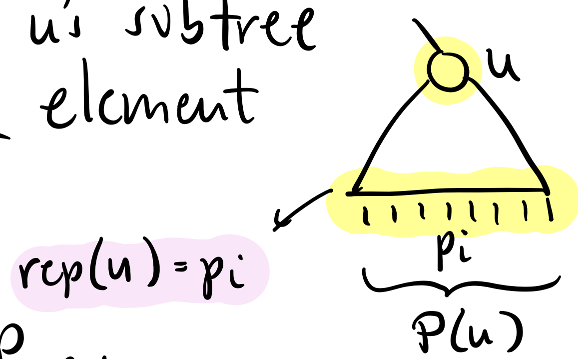
> **Thm:** Given a set of pts $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ can construct a (compressed) quadtree of space $O(n)$ in $O(n \log n)$ time.

Additional information (provided by construction)

Given node u in tree:



- $\text{level}(u)$ = level of u in tree
- $P(u)$ = set of pts in u's subtree
- $\text{rep}(u)$ = an arbitrary element of $P(u)$



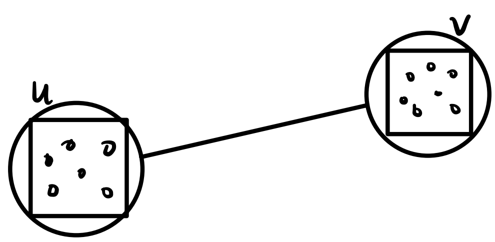
We will represent each WSP as pair of nodes $\{u, v\}$. Actual pair is $\{P(u), P(v)\}$

Constructing the WSPD:

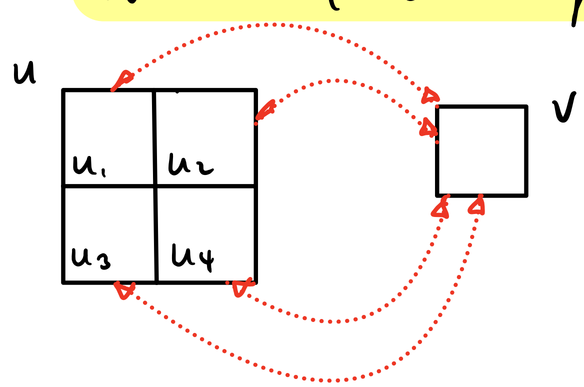
Given P + $s > 0$:
- Build quadtree for P → Let u_0 = root
- Invoke: ws-pairs(u_0, u_0, s)

```
ws-pairs(Node u, Node v, Scalar s) {
    if(u + v are both leaves + u==v) return ∅
  ← if(rep(u) or rep(v) is empty) return ∅
    else if (u + v are s-well sep)
        return {u,v}        // wsp = {P(u), P(v)}
    else      // not w.s.
        if (level(u) > level(v))
            swap u ↔ v    // u is not deeper than v
        let u_1, ..., u_k be u's children
        return ∪_{i=1}^{k} ws-pairs(u_i, v, s)
}
```

Cases: u + v are well sep



u + v not well-sep



Analysis: We'll show $O(s^d \cdot n)$ pairs generated

- **Assume:** Quadtree is not compressed (simpler)
  $s \geq 1$ (else just use $s' = \max(1, s)$)

① **Terminal / Non-Terminal:**

- To count no. of WSP's, we'll count no. of calls to ws-pairs
- A call is:
  - terminal: makes no recursive calls
  - non-terminal: otherwise
- It suffices to count just no. of non-terminal calls (each generates at most $2^d = O(1)$ term. calls)

② **Charging:** We'll count no. of non-term calls by charging each to node of tree.

Preview:
- Each node receives $O(s^d)$ charges
- $O(n)$ nodes in tree

$\Rightarrow O(s^d \cdot n)$ total charges

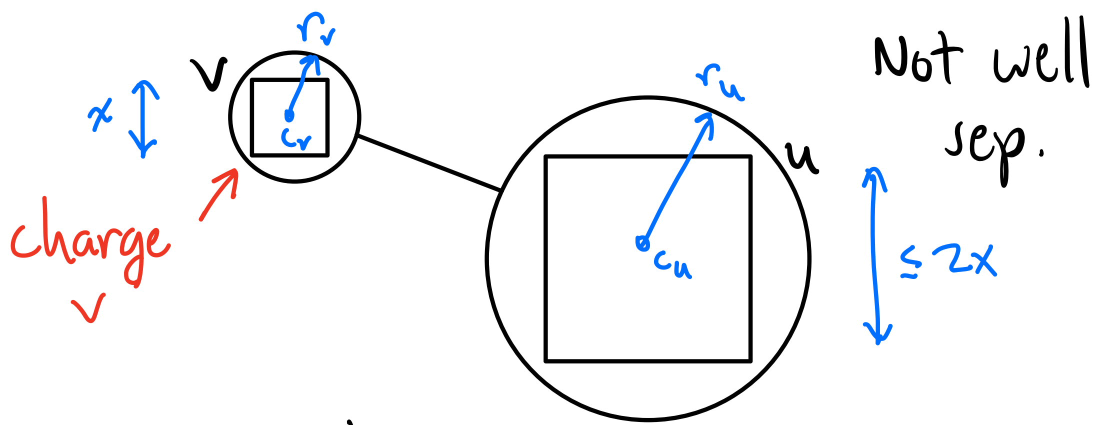
③ Who gets charged?

Let ws-pairs(u, v, s) be non-term call

$\Rightarrow$ u, v not well sep.

$\rightarrow$ Assume (w.l.o.g.) $lev(u) \leq lev(v)$

$\rightarrow$ We will charge v
(smaller node is charged)



- Let x be side length of v's cell
- We always split larger cell first
  $\Rightarrow$ u's side length $\leq 2x$

- Let r_v = radius of ball enclosing v's cell
  + r_u = " " " " u's cell
  $\Rightarrow$ $r_u \leq 2 r_v$
  and $r_v = x\sqrt{d}/2$
- Let c_u, c_v be centers of u + v's cells

This call is non-term

$\Rightarrow$ u, v not well separated

$\Rightarrow$ Distance between balls is

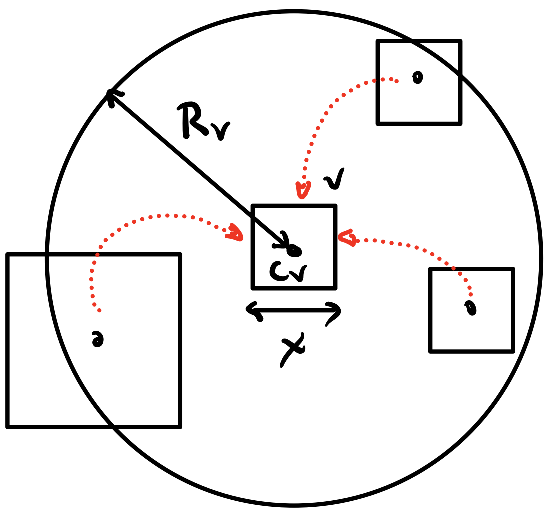
$$< s \cdot \max(r_u, r_v) \le s \cdot r_u \le s(2 \cdot r_v) = s \cdot x \cdot \sqrt{d}$$

$\Rightarrow$ Distance between centers

$$\|c_u - c_v\| \le r_v + r_u + s x \sqrt{d}$$
$$\le x\sqrt{d}/2 + x\sqrt{d} + s x \sqrt{d}$$
$$= \left(\tfrac{1}{2} + 1 + s\right) x \sqrt{d}$$
$$< 3 s x \sqrt{d} \quad (\text{since } s \ge 1)$$

Def: $R_v = 3 s x \sqrt{d}$

$$\boxed{\begin{array}{l}\textbf{Summary: } \text{A node } v \text{ of side length } x \\ \text{is charged by nodes } u \text{ of} \\ \text{side length } x \text{ or } 2x \text{ whose} \\ \text{cell centers lie within a ball} \\ \text{of radius } R_v = 3 s x \sqrt{d} \text{ of } c_v.\end{array}}$$



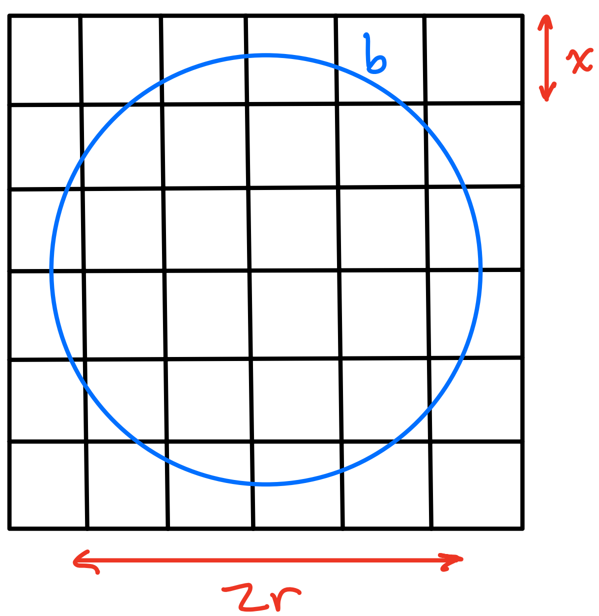
How many such nodes can there be?

Packing Lemma: Given a ball b of radius r in $\mathbb{R}^d$ + any collection X of disjoint quadtree cells of side length $\geq x$ that overlap b, then

$$|X| \leq \left(1 + \left\lceil \frac{2r}{x} \right\rceil\right)^d \leq O\left(\max\left(2, \frac{r}{x}\right)^d\right)$$

Proof: To maximize no. of cells, assume they are as small as possible $\Rightarrow x$

These cells form a grid of side length x that overlaps b



No. of grid squares of side length x overlapping an interval of length $2r$ is

$\leq 1 + \left\lceil \frac{2r}{x} \right\rceil$

$\Rightarrow$ Total: $\left(1 + \left\lceil \frac{2r}{x} \right\rceil\right)^d$

$\square$

Returning to WSPD analysis:

- No. of charges to v $\leq$

  No. of nodes of side length $\geq x$ overlapping a ball of radius $R_v = 3sx\sqrt{d}$

- By Packing Lemma, no. of nodes

$$\leq \left(1 + \left\lceil \frac{2R_v}{x} \right\rceil\right)^d$$

$$\leq \left(1 + \left\lceil \frac{6sx\sqrt{d}}{x} \right\rceil\right)^d$$

$$\leq \left(2 + 6s\sqrt{d}\right)^d$$

$$\leq O(s^d)$$ since $s \geq 1$, d is constant

So, each node charged $O(s^d)$ times

→ $O(n)$ nodes in quadtree

→ $O(n \cdot s^d)$ non-term calls to ws-pairs

→ $O(n \cdot s^d)$ pairs generated

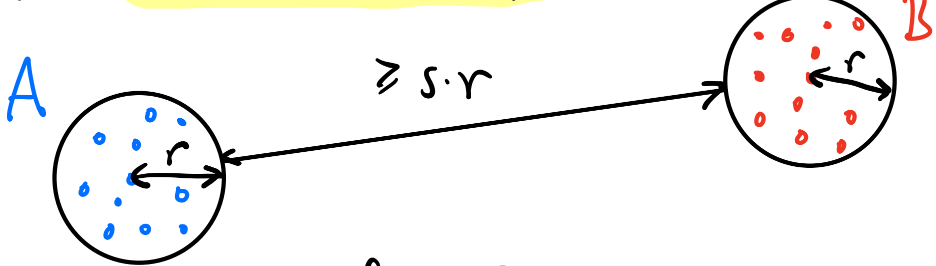
whew!!

Theorem: Given a point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ (d is constant) and $s \geq 1$, in $O(n \log n + s^d n)$ time, can build an s-WSPD for P of size $O(s^d \cdot n)$

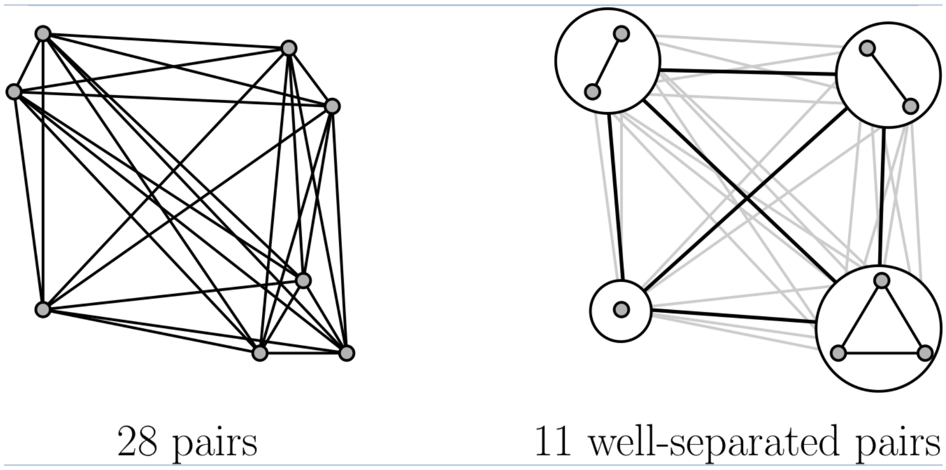
CMSC 754 - Computational Geometry
Lecture 17: Applications of WSPDs

Review of WSPDs:

- Given a point set $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^d$ (d a fixed constant) and separation factor $s > 0$, two sets A + B are s-well separated if they can be contained in two balls of some radius r s.t. the distance between these balls is $\geq s \cdot r$

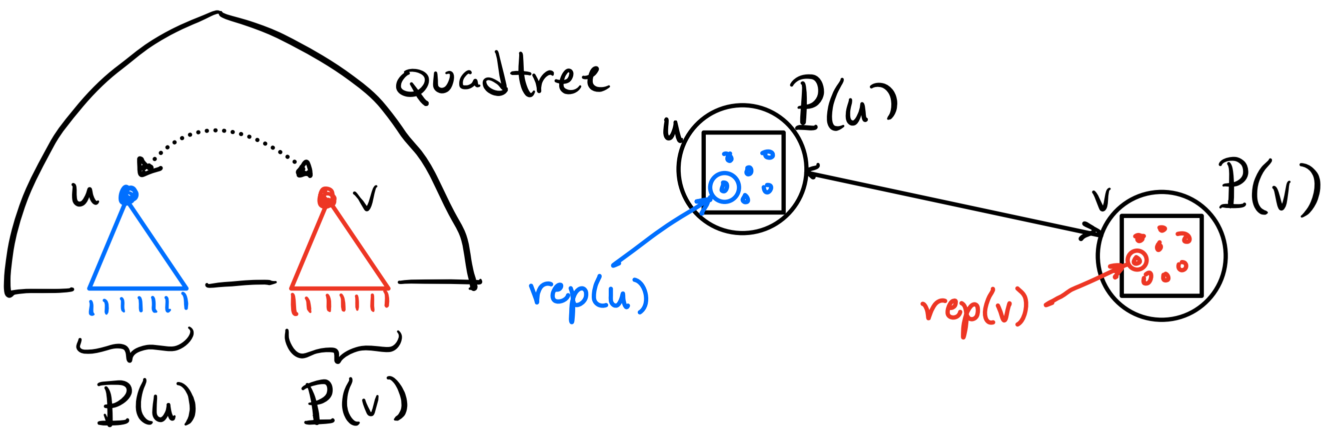


- An s-WSPD for P is a collection: $\{\{A_1, B_1\}, \ldots, \{A_m, B_m\}\}$ such that
  - $A_i, B_i \subseteq P$
  - $A_i \cap B_i = \emptyset$ (disjoint)
  - $\bigcup_i A_i \otimes B_i = P \otimes P$ (cover all pairs)
  - A_i + B_i are s-well separated
- Given P + $s \geq 1$, in time $O(n \log n + s^d \cdot n)$ we can construct an s-WSPD for P of size $O(s^d \cdot n)$.



- Construction is based on d-dim quadtree

- Given nodes u, v in tree let
  P(u) - points in u's subtree
  rep(u) - an arbitrary pt of P(u)
  (u's representative)

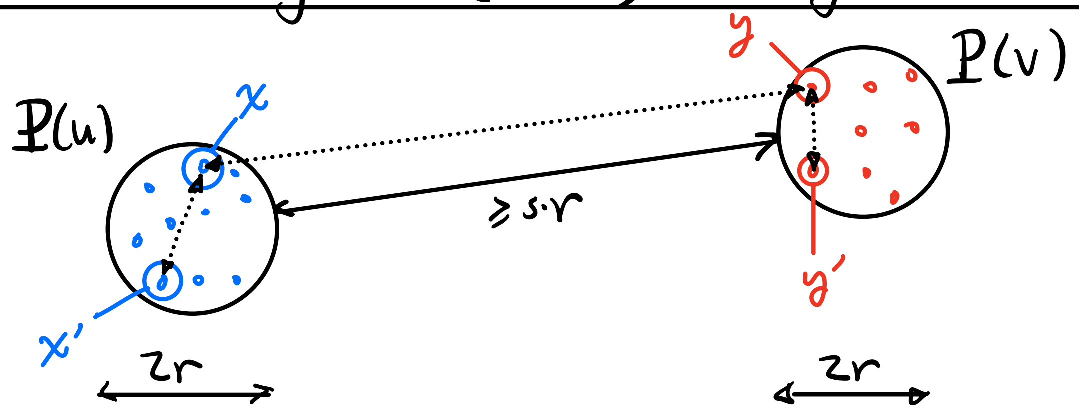


The WSP {P(u), P(v)} is represented by the pair {u, v}

Utility Lemma: Given an s-WSP $\{P(u), P(v)\}$ and $x, x' \in P(u)$ + $y, y' \in P(v)$:

(i) $\|x - x'\| \le 2/s \cdot \|x - y\|$

(ii) $\|x' - y'\| \le (1 + 4/s) \cdot \|x - y\|$



Intuition: (i) Same side closer than cross side

(ii) Cross side dists similar

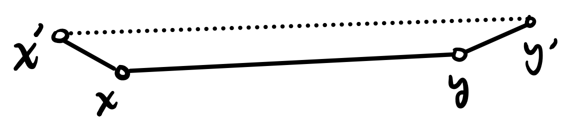
Proof: (i) $\|x - x'\| \le 2 \cdot r$

$$= 2 \cdot r \frac{s \cdot r}{s \cdot r} \le \frac{2 \cdot r}{s \cdot r} \|x - y\|$$

$$= \left(\frac{2}{s}\right) \|x - y\| \quad \checkmark$$

(ii) Observe: $\|x - y\| \ge s \cdot r \Rightarrow 4 \cdot r \le \frac{4}{s} \|x - y\|$

By the triangle inequality:



$$\begin{aligned}
\|x' - y'\| &\le \|x' - x\| + \|x - y\| + \|y - y'\| \\
&\le 2r + \|x - y\| + 2r \\
&\le \|x - y\| + 4r \\
&\le \|x - y\| + 4/s \, \|x - y\| \\
&= (1 + 4/s) \|x - y\| \quad \checkmark
\end{aligned}$$

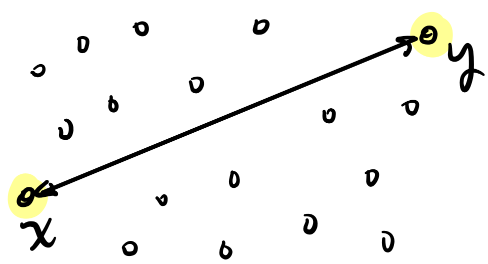
Applications:

- $(1+\varepsilon)$ approx to diameter (farthest pair)
- exact closest pair
- Computing a t-spanner (for any $t>1$)
- $(1+\varepsilon)$ approx to Euclidean MST

$(1+\varepsilon)$ Approx Diameter: in time $O(n\log n + n/\varepsilon^d)$

Given $P=\{p_1,\dots,p_n\}$ in $\mathbb{R}^d$

$$\mathrm{diam}(P) = \max_{x,y\in P} \|x-y\|$$



Exact:

In $\mathbb{R}^2$: Can compute in $O(n\log n)$

[Convex hull + rotating calipers]

$\mathbb{R}^d$: (Nearly)$^?$ quadratic in n

$(1+\varepsilon)$-Approx:

- Set $s = 4/\varepsilon$
- Compute an s-WSPD for P → $O(n\log n + n/\varepsilon^d)$
- for each WSP $\{u,v\}$: → $O(n/\varepsilon^d)$

  $\mathrm{dist}_{u,v} = \|\mathrm{rep}(u) - \mathrm{rep}(v)\|$
- return $\max \mathrm{dist}_{u,v}$ as approx diam

Correctness:

Plan:

① Since reps $\subseteq P$, approx diam $\leq$ diam(P)

② We will show

$*$: $\exists$ WSP u,v s.t.

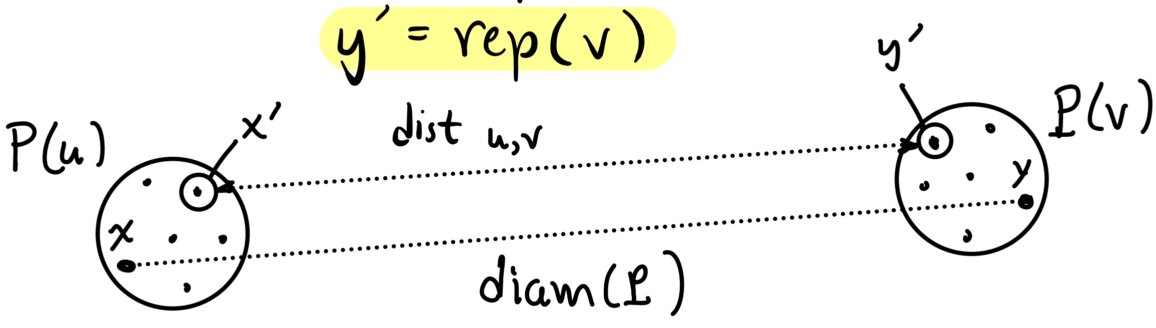
$\text{dist}_{u,v} \geq \text{diam}(P)/(1+\varepsilon)$

$\Rightarrow$ max $\text{dist}_{u,v} \geq \text{diam}(P)/(1+\varepsilon)$

① + ② $\Rightarrow$ $\dfrac{\text{diam}(P)}{1+\varepsilon} \leq$ approx diam $\leq$ diam(P) ✓

Need to show $*$

- Let x, y be diameter pair
- $\exists$ WSP $\{u,v\}$ s.t. $x \in P(u)$ $y \in P(v)$
- Let $x' = \text{rep}(u)$
  $y' = \text{rep}(v)$



By WSPD utility lemma:

$\text{diam}(P) = \|x-y\| \leq \left(1+\frac{4}{s}\right)\|x'-y'\|$

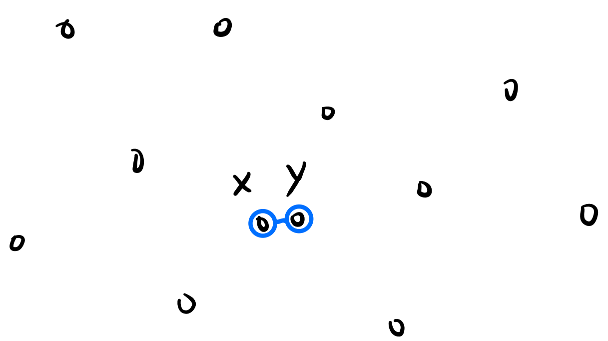
$= (1+\varepsilon)\|x'-y'\|$ $\quad (s = 4/\varepsilon)$

$= (1+\varepsilon)\, d_{u,v}$ $\Rightarrow$ $\text{dist}_{u,v} \geq \text{diam}(P)/(1+\varepsilon)$

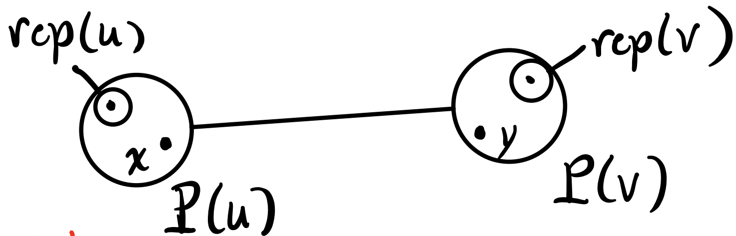
(Exact) Closest Pair: in time $\mathcal{O}(n \log n)$

Given $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ find $x, y \in P$

$$\min_{x,y \in P} \|x - y\|$$



Intuition: Some WSP $\{u,v\}$ must cover the pair $\{x,y\}$



Huh? It looks like x + y not closest!

It must be that $rep(u) = x$
+ $rep(v) = y$

Exact Closest Pair:

- Let $s > 2$ (eg. $s = 2.0001$)
- Build s-WSPD for P
- For each WSP $\{u,v\}$
  $dist_{u,v} = \|rep(u) - rep(v)\|$
- return $\min_{u,v} dist_{u,v}$ as closest dist

$\mathcal{O}(n \log n + 2^d \cdot n) = \mathcal{O}(n \log n)$

Correctness:

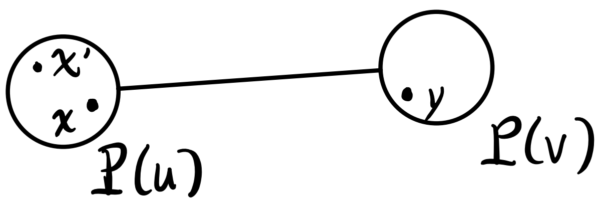
Follow directly from the following lemma:

Lemma: If $s > 0$ + x, y are closest pair in P, then any s-WSPD of P contains the pair $\{\{x\},\{y\}\}$

That is, x + y are singletons in WSPD

Proof:

- Suppose not.
- Let $\{u, v\}$ be WSP with $x \in P(u)$, $y \in P(v)$
- May assume w.l.o.g. that $P(u)$ has another pt x'



- By WSPD Utility Lemma:

$$\|x - x'\| \le \frac{2}{s} \cdot \|x - y\|$$

$$< \|x - y\| \qquad (\text{since } s > 2)$$

$\Rightarrow$ x, y not closest pair

$\rightarrow$ contradiction

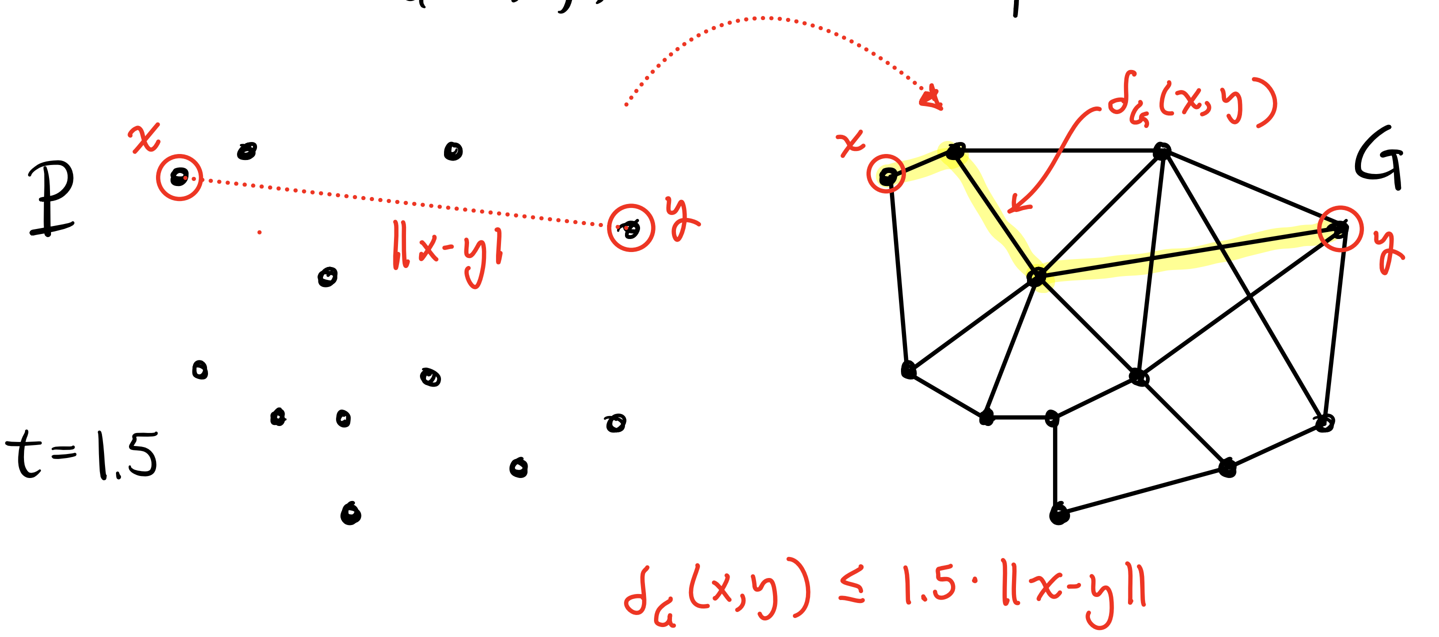
Spanners:

Recall def. of t-spanner (from lect. on Delaunay Tri.)

Given point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ and $t \geq 1$, a t-spanner is a graph on P s.t. $\forall x, y \in P$:

$$\|x-y\| \leq d_G(x,y) \leq t \cdot \|x-y\|$$

where $d_G(x,y)$ is shortest path dist in G



We will show that given $P \subseteq \mathbb{R}^d$ + $t > 1$ can build a $(1+\varepsilon)$-spanner for P in time $O(n \log n + n/\varepsilon^d)$ consisting of $O(n/\varepsilon^d)$ edges

Spanner construction (Given P + $t > 1$)

- Let $s = \dfrac{4(t+1)}{t-1}$
- $G \leftarrow$ graph with vertex set P + no edges
- Build an s-WSPD for P
- for each WSP $\{u, v\}$:
    - add edge $(\text{rep}(u), \text{rep}(v))$ to G
- return G

Time: If $t = 1 + \varepsilon$, $s = O(1/\varepsilon)$ $[0 < \varepsilon < 1]$

$\Rightarrow O(n \log n + n/\varepsilon^d)$

Size: $O(n/\varepsilon^d)$ WSPs $\Rightarrow O(n/\varepsilon^d)$ edges

Correctness:

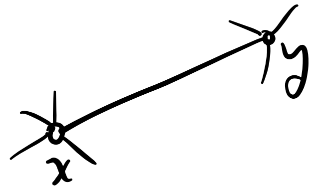
Will show that for all $x, y \in P$

$$\|x-y\| \overset{①}{\leq} \delta_G(x,y) \overset{②}{\leq} t \cdot \|x-y\|$$

① Trivially true since G is a subgraph of complete Euclidean graph

② Rest of the proof...
Induction on num. of edges in path from x to y in G

Basis: Edge (x,y) is in G

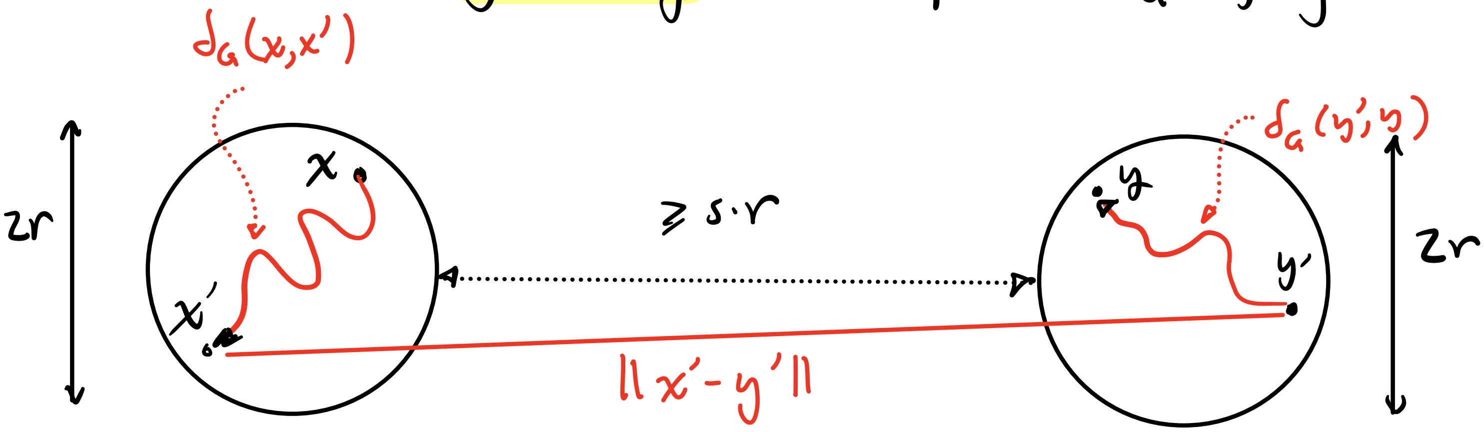


$\Rightarrow \delta_G(x,y) = \|x-y\| \leq t \cdot \|x-y\|$ ✓
(since $t > 1$)

Induction step:

- $\exists$ pair $\{u,v\}$ in WSPD that covers the pair (x,y)
- Let $x' = rep(u)$ $y' = rep(v)$
  (possibly $x' = x$ or $y' = y$)

- To get from x to y in G we can:
  - x to x' $\longrightarrow$ path $\delta_G(x,x')$
  - x' to y' $\longrightarrow$ direct edge: $\|x'-y'\|$
  - y' to y $\longrightarrow$ path $\delta_G(y',y)$



By the induction hyp: $\delta_G(x,x') \leq t \cdot \|x-x'\|$
$\delta_G(y',y) \leq t \cdot \|y'-y\|$

$$\Rightarrow \delta_G(x,y) \le t \cdot \|x-x'\| + \|x'-y'\| + t \cdot \|y'-y\|$$
$$= t(\|x-x'\| + \|y'-y\|) + \|x'-y'\|$$

By WSPD Utility Lemma:

- $\|x-x'\| \le \frac{2}{s}\|x-y\|$
- $\|y'-y\| \le \frac{2}{s}\|x-y\|$
- $\|x'-y'\| \le \left(1+\frac{4}{s}\right)\|x-y\|$

$$\Rightarrow \delta_G(x,y) \le t\left(\frac{2}{s}\|x-y\| + \frac{2}{s}\|x-y\|\right) + \left(1+\frac{4}{s}\right)\|x-y\|$$
$$= \left(t\frac{4}{s} + 1 + \frac{4}{s}\right)\|x-y\|$$
$$= \left(1 + \frac{4(t+1)}{s}\right)\|x-y\|$$
$$= t\|x-y\| \qquad \left(\text{since: } s = \frac{4(t+1)}{t-1}\right)$$

$\square$

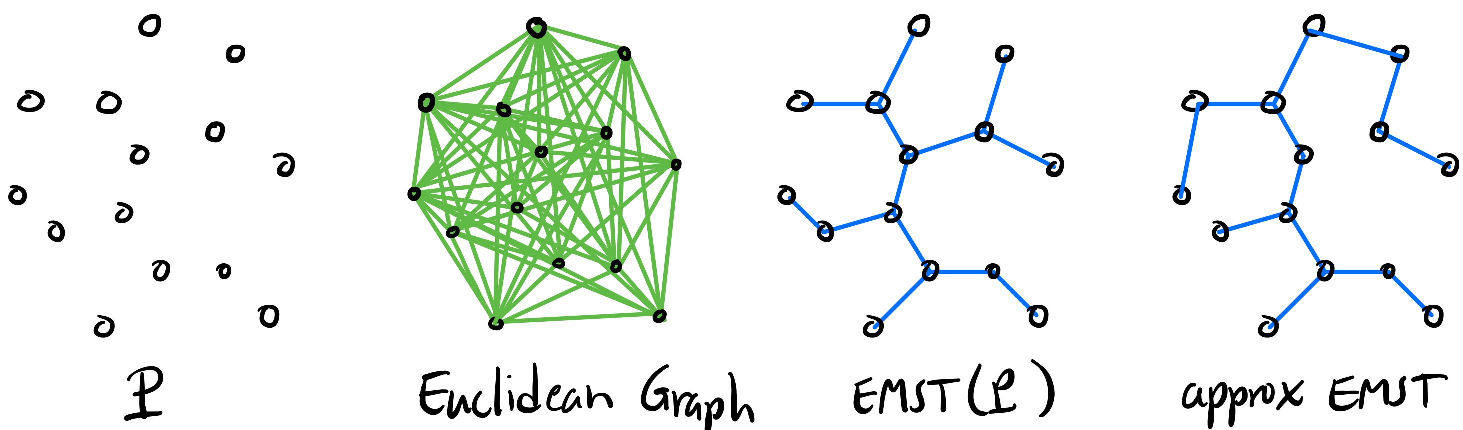
To obtain a $(1+\varepsilon)$-spanner, set $t = 1+\varepsilon$ + apply this construction

Approx. to Euclidean MST

Given a point set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$
define:

$EMST(P)$ = Min. spanning tree of complete Euclidean graph on P (where $w(u,v) = \|u-v\|$)

Let: $emst(P) = \sum_{(x,y)\in EMST(P)} \|x-y\|$
= total weight of $EMST(P)$



A graph H is an $(1+\varepsilon)$-approx EMST if:

(1) H is a spanning tree for P

(2) $w(H) \le (1+\varepsilon)\cdot emst(P)$

where $w(H)$ = total weight of H's edges

We'll show how to compute an $(1+\varepsilon)$-approx EMST in time $O(n\log n + n/\varepsilon^d)$

$\boxed{\begin{array}{l}\text{approx-EMST}(P, \varepsilon) \\ \quad - \; G \leftarrow (1+\varepsilon)\text{-spanner for } P \\ \quad - \; \text{return } MST(G)\end{array}}$

Time: Compute G: $O(n \log n + n/\varepsilon^d)$

Compute $MST(G)$:

- Can compute MST of a graph with v vertices + e edges in time $O(v \log v + e)$
- G has n vertices + n/ε^d edges
- $MST(G)$ takes $O(n \log n + n/\varepsilon^d)$

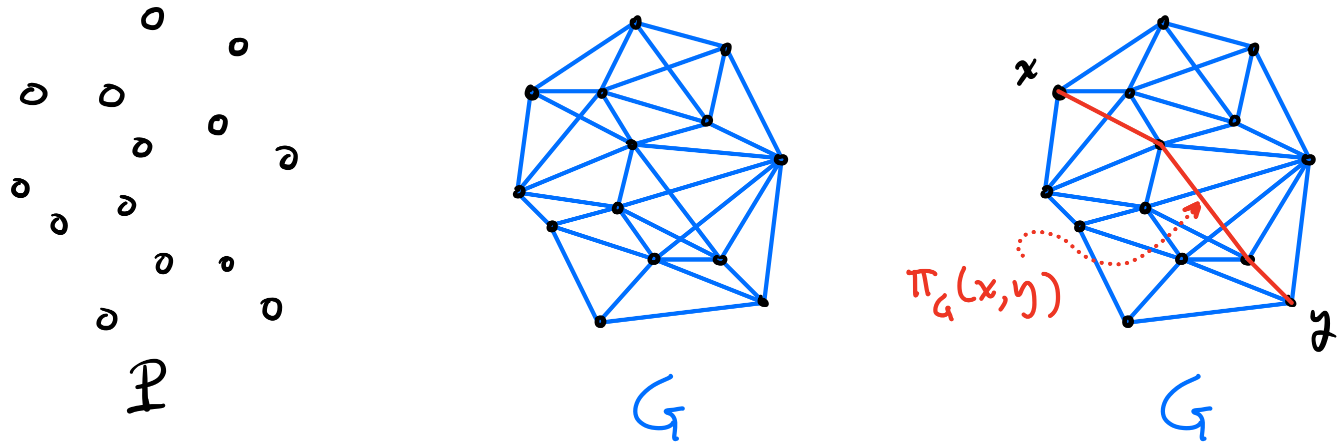
Correctness:

- We'll show that G has a connected subgraph that contains all pts of P ("spans P") and has weight $\leq (1+\varepsilon) \cdot emst(P)$
- If G has a spanning subgraph H of weight W, then the weight of its MST is no larger

For each $x, y \in P$, let $\pi_G(x,y)$ be shortest path from x to y in G.

Let $\delta_G(x,y)$ be length of this path

Know that $\delta_G(x,y) \leq (1+\varepsilon)\,\|x-y\|$



H:

for each $(x,y) \in \mathrm{EMST}(\mathcal{P})$

add the edges of $\pi_G(x,y)$ to H

Obs:

① H is connected and spans all pts of $\mathcal{P}$

② Total weight:

$$\begin{aligned} w(H) &\leq \sum_{(x,y)\in \mathrm{EMST}(\mathcal{P})} \delta_G(x,y) \\ &\leq \sum_{(x,y)\in \mathrm{EMST}(\mathcal{P})} (1+\varepsilon)\cdot\|x-y\| \\ &= (1+\varepsilon) \sum_{(x,y)\in \mathrm{EMST}(\mathcal{P})} \|x-y\| \\ &= (1+\varepsilon)\cdot \mathrm{emst}(\mathcal{P}) \end{aligned}$$

① + ② $\Rightarrow$ $w(MST(G))$

$$\leq w(H) \leq (1+\varepsilon) \cdot emst(P)$$

✓

CMSC 754 - Computational Geometry
Lecture 18: Coresets and Kernels

Approximation by Sampling:

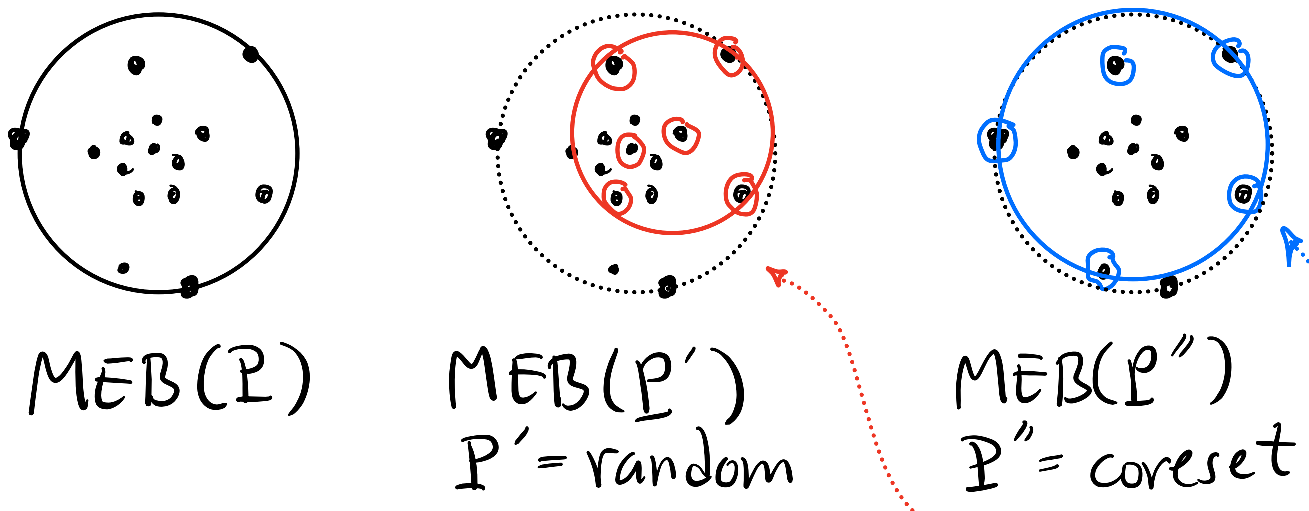
- Running time too slow?
- Maybe your data size is too large!
- Idea:
  - Extract a small subset, $P' \subseteq P$
  - Run solve problem exactly on P'
  - Prove that the answer on P' is "close" to optimal on P.

How to compute P'?

- Depends on your problem
- Random sampling is most common, but not necessarily best

Example: Minimum Enclosing Ball (MEB)

- Given a set $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^d$ compute the Euclidean ball of min. radius enclosing P.



Problem with random sampling:

- MEB(P) depends on points near periphery
- Random sample extracts many irrelevant points.
- Smarter: Use a sampling method that gives priority to peripheral points

Coreset: Let P be input set.

$f^*(P) \to \mathbb{R}$ is our objective function

(eg. $f^*(P)$ = radius of MEB)

Given $\varepsilon > 0$, an ε-coreset is a subset $Q \subseteq P$ s.t.

$$1-\varepsilon \leq \frac{f^*(Q)}{f^*(P)} \leq 1+\varepsilon$$

The opt. soln. for Q is close to opt. for P

Questions:

- For what optimization problems do (small) coresets exist?
- (As a function of $n + \varepsilon$) how small is the coreset?
- How fast can we compute a coreset?

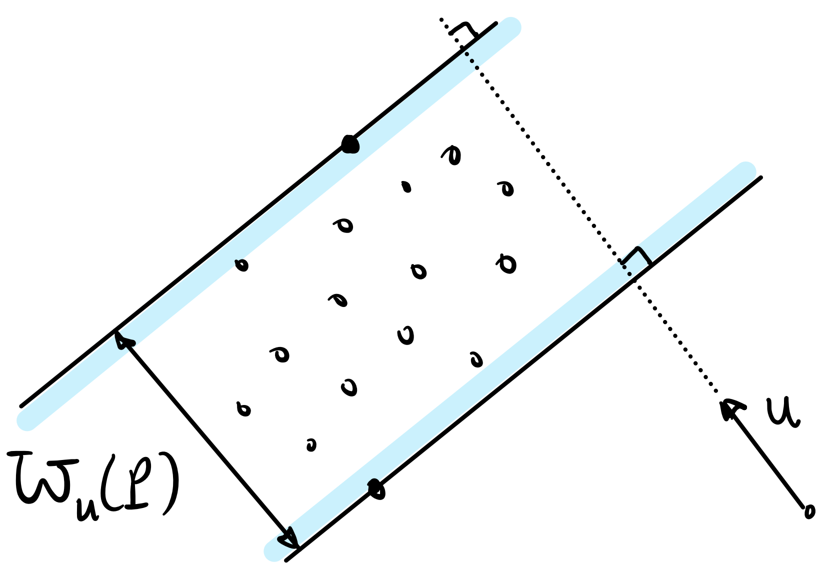
Coreset for Directional Width:

(also called ε-kernel)

- Given a pt set $P \subseteq \mathbb{R}^d$
- Given a unit vector $\vec{u}$

- Directional width of P in direction $\vec{u}$ is:

$$\omega_u(P) = \max_{p \in P} (\vec{p} \cdot \vec{u}) - \min_{p \in P} (\vec{p} \cdot \vec{u})$$



Given $\varepsilon > 0$, an ε-coreset for direc. width (also called ε-kernel) is a subset $R \subseteq P$ s.t.

$\forall$ unit vect. $\vec{u}$:

Trivially true since $R \subseteq P$

$$(1-\varepsilon)\,\omega_u(P) \leq \omega_u(R) \leq \omega_u(P)$$

Getting this is the objective

Aside: When computing approx. lower bounds we sometimes write:

$$(1-\varepsilon)\cdot \text{exact} \leq \text{approx}$$

and other times:

$$\frac{\text{exact}}{1+\varepsilon} \leq \text{approx}$$

Does the form matter?

Not really. If $0 < \varepsilon < 1$, then

$$1-\varepsilon \leq \frac{1}{1+\varepsilon} \leq 1 - \varepsilon/2$$

- Only constant factors are affected

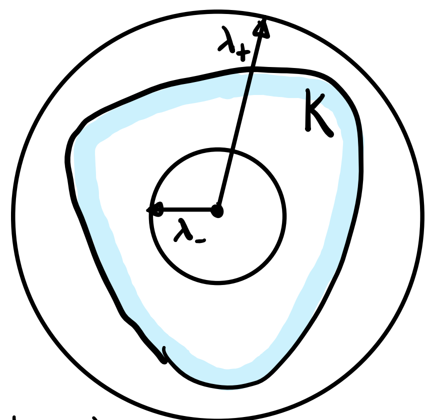
Useful facts:

Chain Property: If X is an ε-kernel for Y
and Y is an ε'-kernel for Z
then X is an $(\varepsilon+\varepsilon')$-kernel for Z

Union Property: If X is an ε-kernel for P,
X' is an ε-kernel for P'
then $X \cup X'$ is an ε-kernel for $P \cup P'$

Canonical Position: We like fat things...

Fat: Given $0 \leq \alpha \leq 1$, a convex body K is
is α-fat if K can be sandwiched
between two concentric balls of radii
$\lambda_- \leq \lambda_+$ where
$\alpha = \lambda_- / \lambda_+$



Canonical Position: Convex body K is in
α-canonical form if it is sandwiched
between balls of radius $\lambda_- = 1/2\alpha$ + $\lambda_+ = 1/2$
centered at the origin.

Why $1/2$? $\Rightarrow$ K's diameter ≤ 1

A point set P is $\left\{\begin{array}{l}\alpha\text{-fat} \\ \\ \alpha\text{-canonical form}\end{array}\right\}$ if $conv(P)$ is.

We can convert any pt set into canonical form.

Affine Transformation: Is a linear transformation (scaling + rotation + shearing) + translation

Lemma: Given any n-element pt. set $P \subseteq \mathbb{R}^d$, there exists an affine transformation T that maps P into $(1/d)$-canonical form

- $R \subseteq P$ is an ε-kernel for P iff $T(R)$ is an ε-kernel for $T(P)$
- T can be computed in $O(n)$ time

Proof makes use of important fact: (1948)

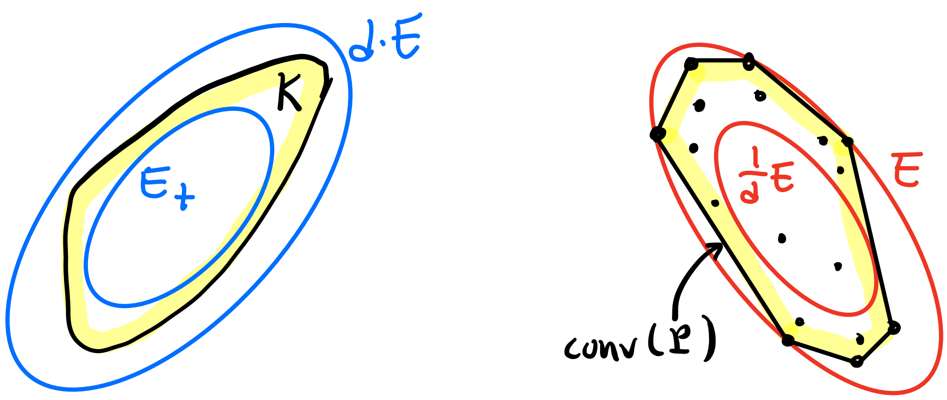
John's Theorem: Given any convex body $K \subseteq \mathbb{R}^d$, let E be max volume ellipsoid contained in K, then

$$E \subseteq K \subseteq d \cdot E$$

where $d \cdot E$ is a factor-d scaling E about its center.

Equiv: Given pt. set P, let E be min vol. ellipsoid containing P, then

$$\tfrac{1}{d} E \subseteq \mathrm{conv}(P) \subseteq E$$



- The ellipsoid is called the John Ellipsoid or Löwner-John Ellipsoid
- Can compute it in $O(n)$ time (randomized incremental)

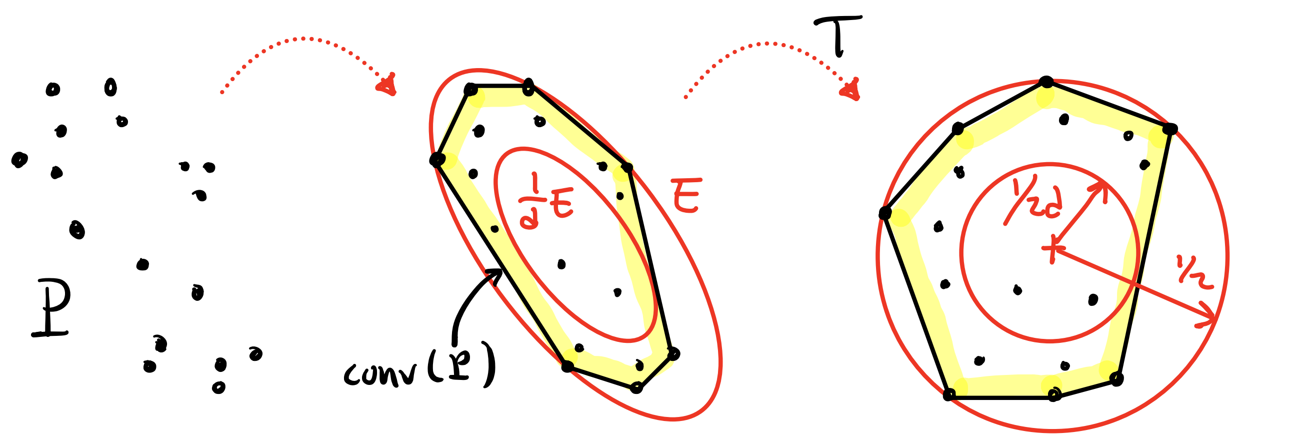
Fact: Given any ellipsoid E, there exists an affine transformation that maps E to a unit ball, centered at origin.

Proof (of canonical form lemma):

① Compute P's outer John ellipsoid E

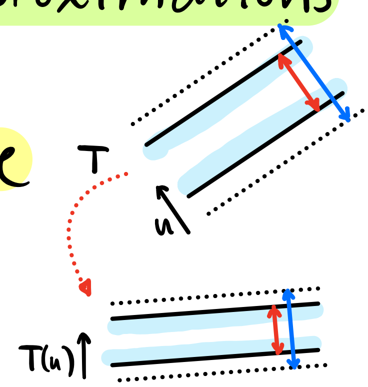
② Find affine transformation mapping E to unit ball centered at origin

③ Scale by ½ → output resulting transformation T



Why are directional width approximations preserved?

- Affine transformations preserve ratios of parallel lengths (Details omitted)



Quick + Dirty Kernel: Simple but not optimal size – $O(1/\varepsilon^d)$

Given $P \subseteq \mathbb{R}^d$ + $\varepsilon > 0$:

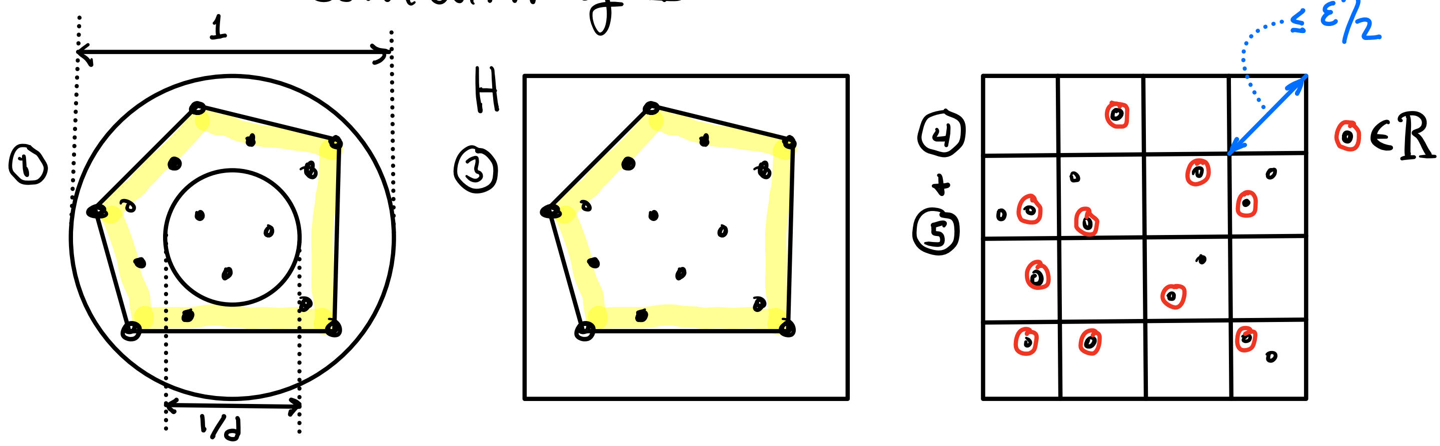
① Map P to $1/d$-canonical position

Note: $\forall u,\ 1/d \leq \omega_u(P) \leq 1$

$\Rightarrow$ absolute error of $\varepsilon/d \Rightarrow$ rel. error $\leq \varepsilon$

② Let $\varepsilon' = \varepsilon/d$

③ Let $H = [-1/2, +1/2]^d$ be unit hypercube containing P



④ Subdivide H into square grid of diameter $\leq \varepsilon'/2$ (equiv., side length $= \varepsilon'/2\sqrt{d}$)

Note: No. of grid cells is $\left(\frac{1}{\varepsilon'/2\sqrt{d}}\right)^d = O(1/\varepsilon^d)$

⑤ $R \leftarrow$ take one pt of P from each occupied cell

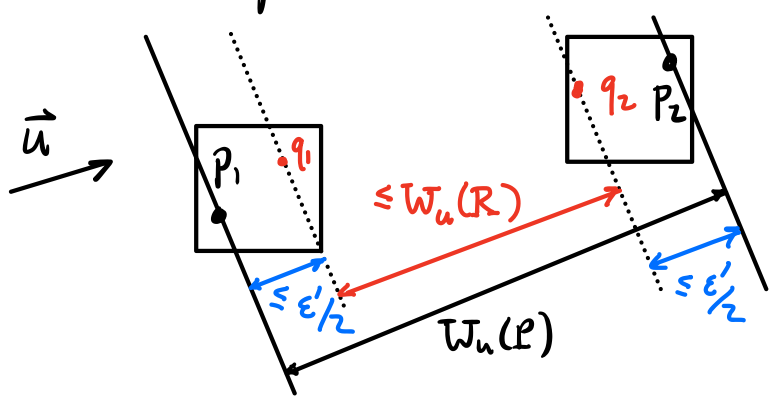
Note: $|R| = O(1/\varepsilon^d)$. Computable in $O(n)$ time

Running time: $O(n + 1/\varepsilon^d)$

- Canonical position – $O(n)$
- Place pts in grid cells – $O(n)$
  [integer division + hashing]
- Output R – $O(1/\varepsilon^d)$

Correctness:

- Given any direction $\vec{u}$, let $p_1, p_2 \in P$ be pts that define $\omega_u(P)$
- Let $q_1, q_2 \in R$ be corresponding representatives from p_1 + p_2's cells



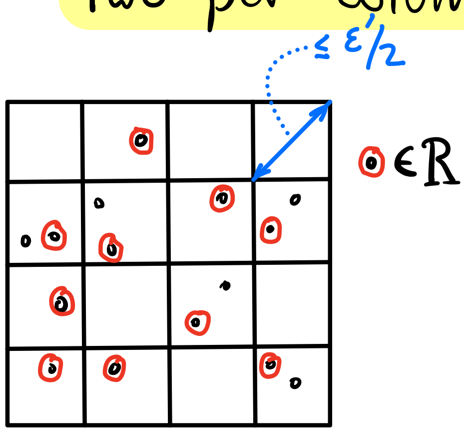
- Since cell diameter $\leq \varepsilon'/2$, it follows that
  $$\omega_u(P) \leq \varepsilon'/2 + \omega_u(R) + \varepsilon'/2$$
  $$= \varepsilon' + \omega_u(R) = \varepsilon/d + \omega_u(R)$$
- By canonical form, $\omega_u(P) \geq 1/d$
  $$\omega_u(P) \leq \varepsilon \cdot \omega_u(P) + \omega_u(R)$$
  $$\Rightarrow (1-\varepsilon)\,\omega_u(P) \leq \omega_u(R) \leq \omega_u(P)$$
  since $R \subseteq P$

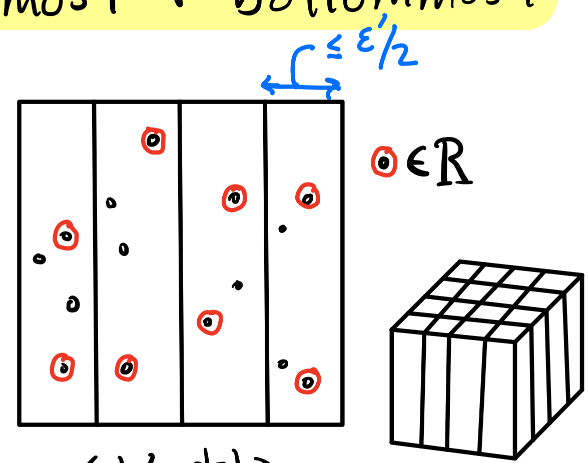
$\Rightarrow$ R is an ε-kernel

$\square$

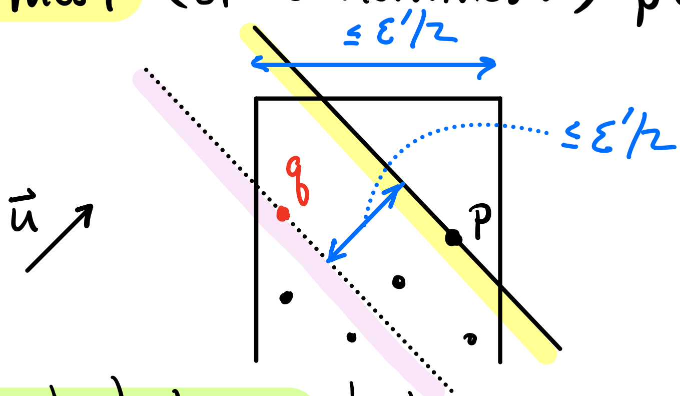
Small Improvement: $\cancel{O(1/\varepsilon^{d})} \to O(1/\varepsilon^{d-1})$

- Quick + dirty's grid includes many internal points → wasteful
- Rather than take:
  - one representative per cell, instead
  - two per column – toppmost + bottommost





- How many? Top grid has $O(1/\varepsilon^{d-1})$ cells
  $|R| = 2 \cdot O(1/\varepsilon^{d-1}) = O(1/\varepsilon^{d-1})$
- Correctness? Let p be extreme pt in direction $\vec{u}$ + let $q \in R$ be topmost (or bottommost) pt in column



Directional distance betw. q + p is $\leq \varepsilon/2$

... remaining details omitted

Big Improvement – ε-kernel of size $O(1/\varepsilon^{\frac{d-1}{2}})$
[optimal in the worst case]

Construction based on idea discovered (independently) by Dudley + Bronsteyn + Ivanov (~1974)

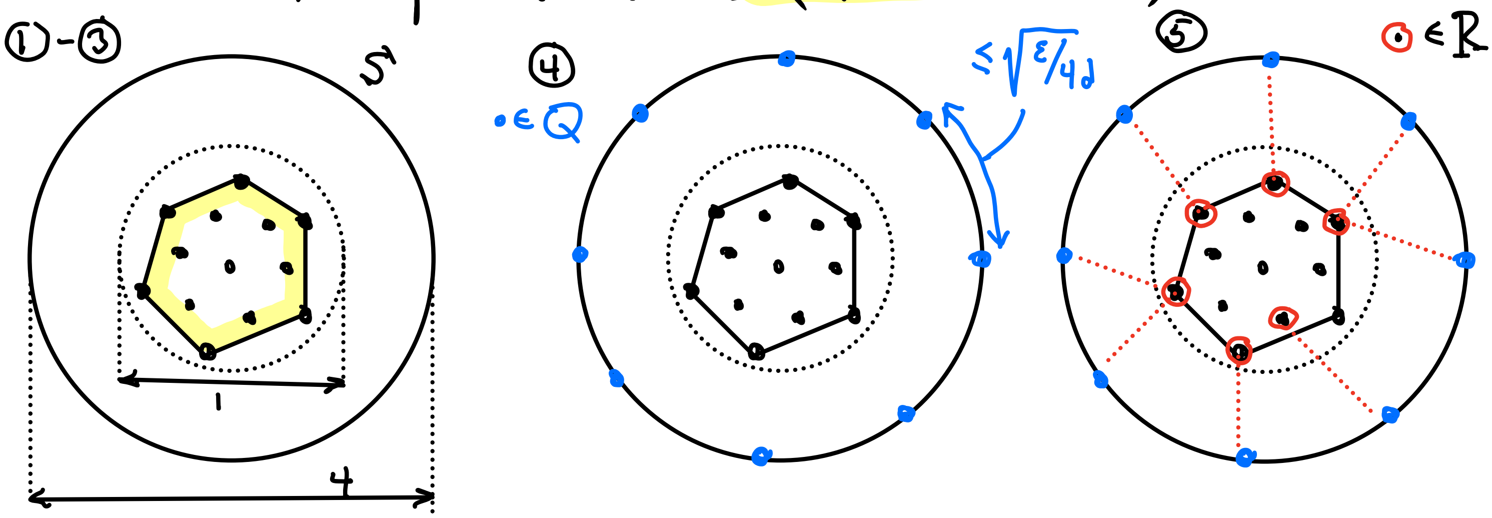
① Map P to $1/d$-canonical position

Note: $\forall u$, $1/d \leq w_u(P) \leq 1$
$\Rightarrow$ absolute error of $\varepsilon/d \Rightarrow$ rel. error $\leq \varepsilon$

② Let $\varepsilon' = \varepsilon/d$

③ Let S = sphere of radius 2 centered at origin, let $\delta = \sqrt{\varepsilon/4d}$

④ Let Q be a set of points on S s.t. any point of S is within distance δ of some pt of Q. (Q is "δ-dense")



⑤ For each $q \in Q$, let $nn(q) \in P$ be its closest pt.
Return: $R = \bigcup_{q \in Q} nn(q)$

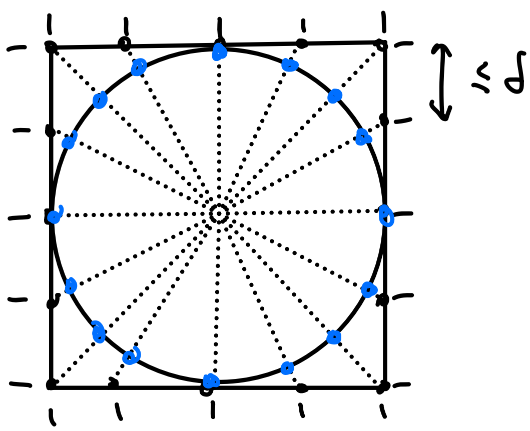
Size: $|R| \le |Q|$

- Claim that $|Q| = O((1/\sqrt{\varepsilon})^{d-1}) = O(1/\varepsilon^{\frac{d-1}{2}})$
- **Intuition:** Each $q \in Q$ covers a spherical cap of radius $\delta \cong \sqrt{\varepsilon}$
  - Such a cap has surface area $\approx \delta^{d-1} \approx \sqrt{\varepsilon}^{\,d-1} \approx \varepsilon^{(d-1)/2}$
  - S has constant radius $\Rightarrow$ constant area
  - No. caps needed to cover S $\approx \text{const}/\varepsilon^{(d-1)/2} = O(1/\varepsilon^{(d-1)/2})$

$\Rightarrow |R| = O(1/\varepsilon^{(d-1)/2})$

Running Time:

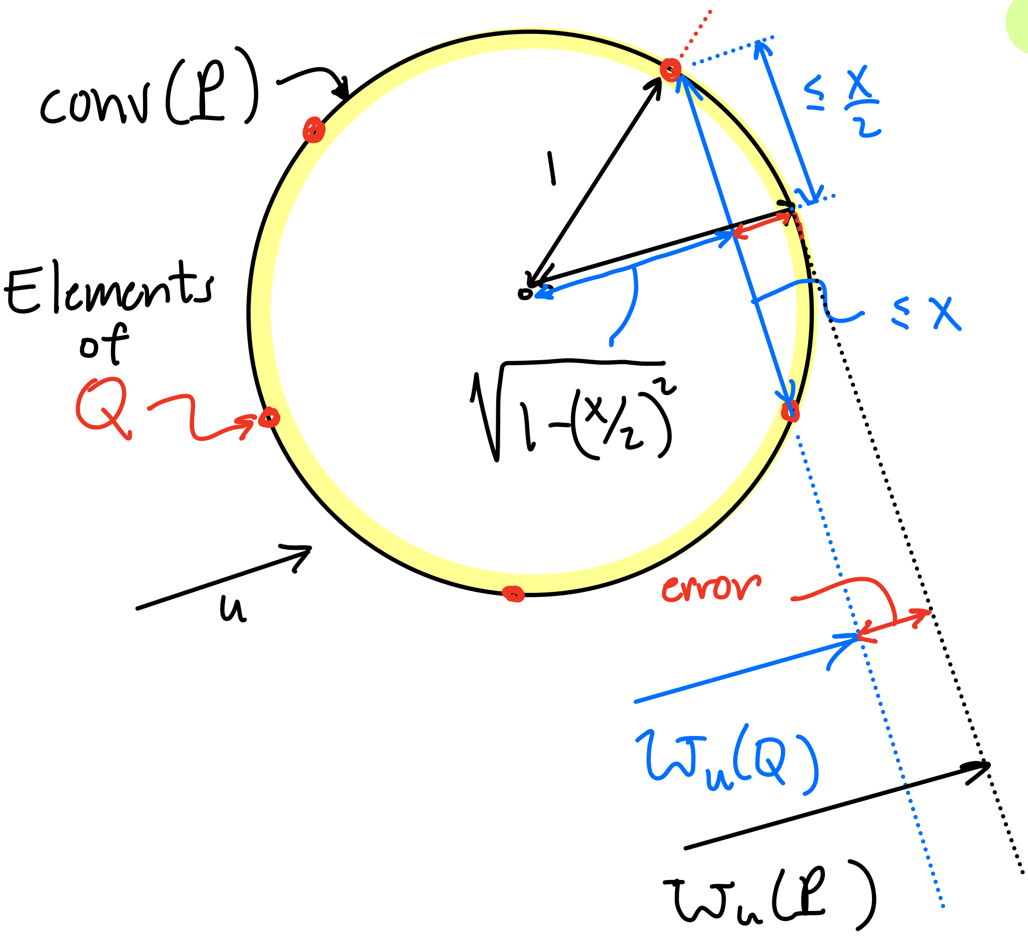
- Canonical position: $O(n)$
- Computing δ-dense Q
  $O(|Q|) = O(1/\varepsilon^{(d-1)/2})$
  How? Enclose S in a hypercube
  Cover hypercube with grid $\sim \delta$
  Project onto S
- Compute $nn(q)$
  - Suffices to use approx nn
  - $O(\text{poly}(1/\varepsilon) \cdot \log n)$



Correctness: (Complex — See latex notes)

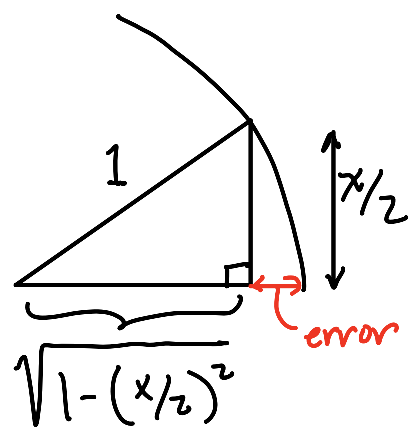
We'll consider a simpler question:

Why $\sqrt{\varepsilon}$ spacing? Simple case: Suppose **conv($\mathcal{P}$)** is a **unit ball**



To generate a large error select u to hit conv($\mathcal{P}$) midway between pts of Q.

By **Pythagorean Thm:**



$$\text{error} \leq 1 - \sqrt{1-(x/2)^2}$$

we want

$$\text{error} \leq \varepsilon$$

That is, want x s.t.

$$1 - \sqrt{1 - (x/2)^2} \le \varepsilon$$

Solving for x, we have:

$$\Leftrightarrow \quad 1 - (x/2)^2 \geq (1-\varepsilon)^2 = 1 - 2\varepsilon + \varepsilon^2$$

if $\varepsilon \le 1$, then $\varepsilon^2 \le \varepsilon \Rightarrow 1 - 2\varepsilon + \varepsilon^2 \le 1 - \varepsilon$

$$\Leftarrow \quad 1 - (x/2)^2 \geq 1 - \varepsilon$$

$$\Leftrightarrow \quad x/2 \le \sqrt{\varepsilon}$$

$$x \le 2\sqrt{\varepsilon}$$

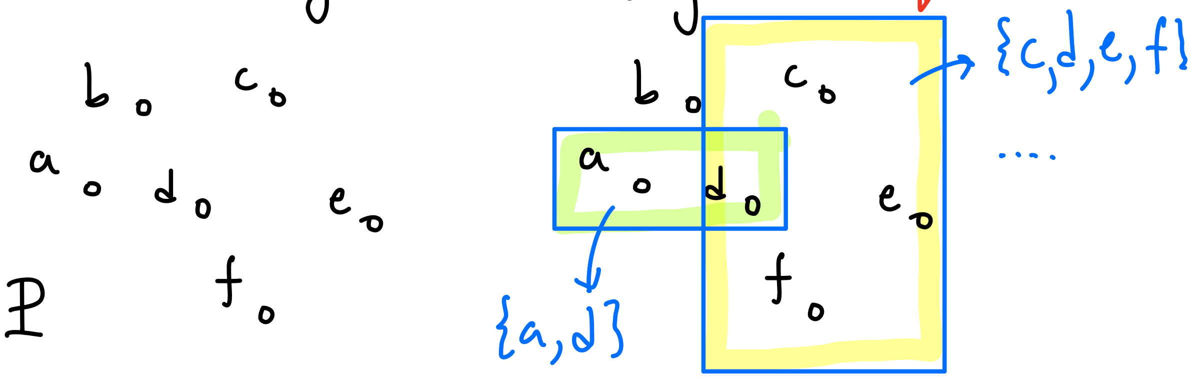
- This explains why spacing $\sim \sqrt{\varepsilon}$ is the right thing to do

- Notice this is tight up to constant factors.

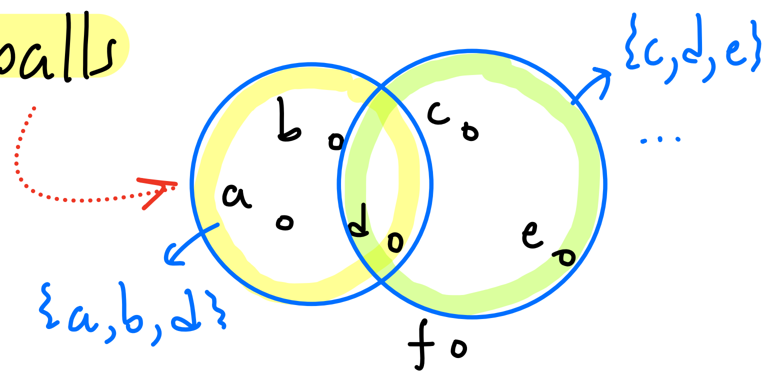
CMSC 754 - Computational Geometry
Lecture 19 - Sampling + VC-Dimension

Geometric Set Systems:

- Many problems involve sets of points that are defined by geometric objects
- Example: Given a set $P \subseteq \mathbb{R}^d$, consider all subsets of P contained in:
    - axis-aligned rectangles



    - Euclidean balls



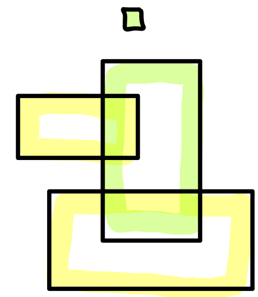
Range Space:

Given a set P, let 2^P denote the power set of P, consisting of all subsets of P $(|2^P| = 2^{|P|})$

Range space is a pair $(X, \mathcal{R})$ where:

X - **domain** (a set)
$\mathcal{R}$ - **ranges** - a subset of 2^X

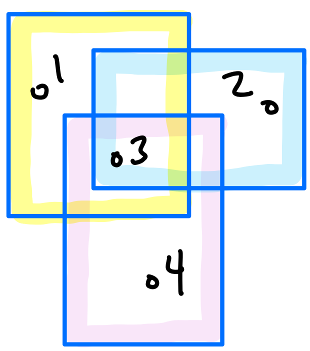
Eg. $X = \mathbb{R}^2$
$\mathcal{R}$ = set of all axis-aligned rectangles
(each is an infinite set)



Restriction: Given $P \subseteq X$, define

$$\mathcal{R}_{|P} = \{ P \cap Q \mid Q \in \mathcal{R} \}$$

the **restriction of $\mathcal{R}$ to P**



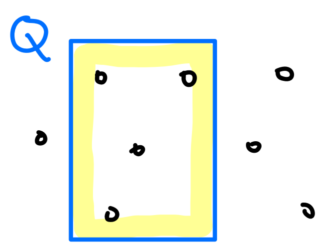
$\mathcal{R}_{|P} = \emptyset, \{1\}, \dots, \{4\}, \dots \{1,2,3,4\}\}$

But not: $\{1,4\}$ or $\{1,2,4\}$

Range space $(X, \mathcal{R})$ is **discrete** if $|X|$ finite

Given a discrete range space $(P, \mathcal{R})$
and any $Q \in \mathcal{R}$ define Q's **measure**

$$\mu(Q) = \frac{|Q \cap P|}{|P|}$$



$\mu(Q) = \frac{4}{8} = \frac{1}{2}$

Sampling: Rather than deal with entire point set (may be huge) we would like a "good" sample.

Given $S \subseteq P$ (presumably $|S| \ll |P|$) define

$$\hat{\mu}_S(Q) = \frac{|Q \cap S|}{|S|}$$

(When S is clear, we write $\hat{\mu}(Q)$)

How good is S as a sample?

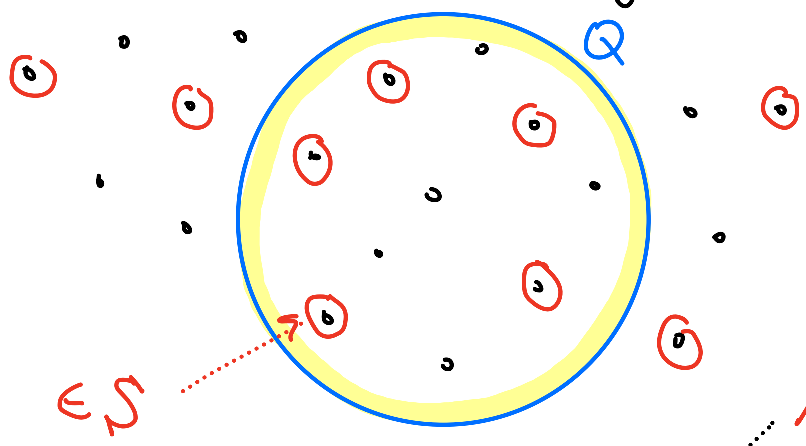
Given a discrete range space (P, R) + $\varepsilon > 0$

$\boxed{\textbf{$\varepsilon$-sample:} \ S \subseteq P \text{ is an } \varepsilon\text{-sample if } |\mu(Q) - \hat{\mu}(Q)| \le \varepsilon \quad \forall Q \in R}$

$\boxed{\textbf{$\varepsilon$-net:} \ S \subseteq P \text{ is an } \varepsilon\text{-net if } \mu(Q) \geq \varepsilon \Rightarrow S \cap Q \neq \emptyset \quad \forall Q \in R}$

Intuition:

- S is an **ε-sample** if it captures roughly the **same proportion** of elements for any range



$$\mu(Q) = \frac{10}{20} = 0.5$$

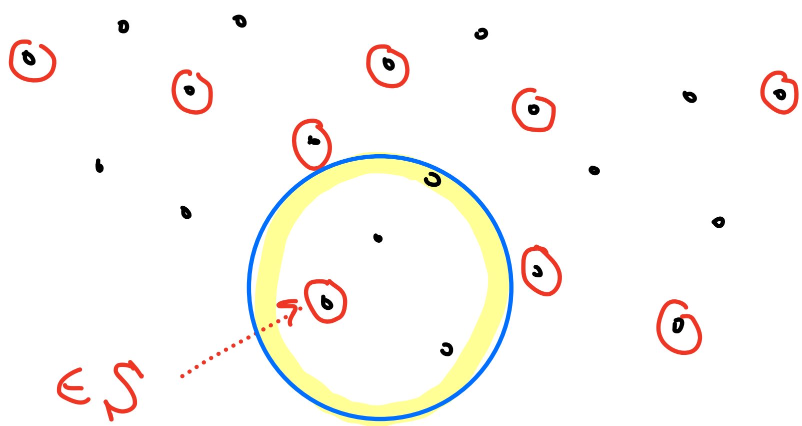
$$\hat{\mu}(Q) = \frac{5}{8} = 0.625$$

$$|\mu(Q) - \hat{\mu}(Q)| = 0.125 = 1/8$$

If this holds for all ranges in $\mathcal{R}$
S is a $1/8$-sample.

- A range Q is **ε-heavy** if $\mu(Q) \geq \varepsilon$

An **ε-net hits all ε-heavy ranges**

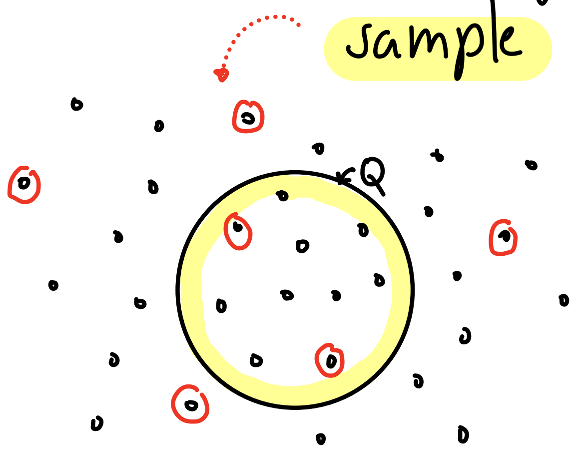


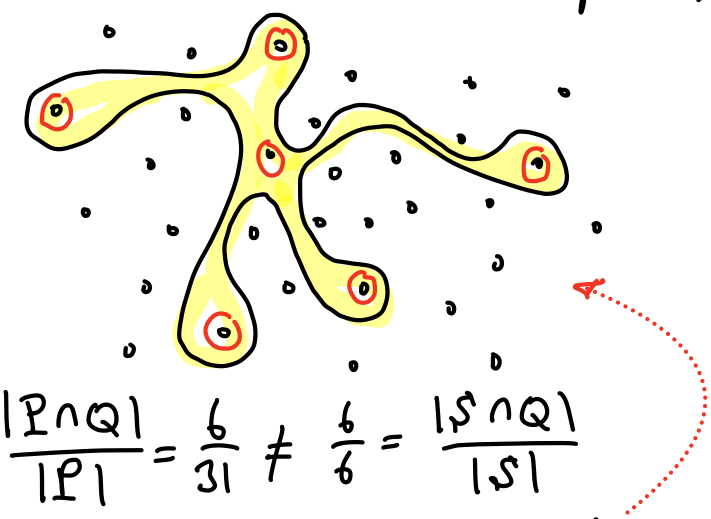
$\varepsilon = 1/5$

Any range that contains $\geq \varepsilon \cdot |P| = 4$ pts must hit a pt of S

How to construct ε-nets + ε-samples?

Intuition: Any sufficiently large random sample should work (with some prob.)





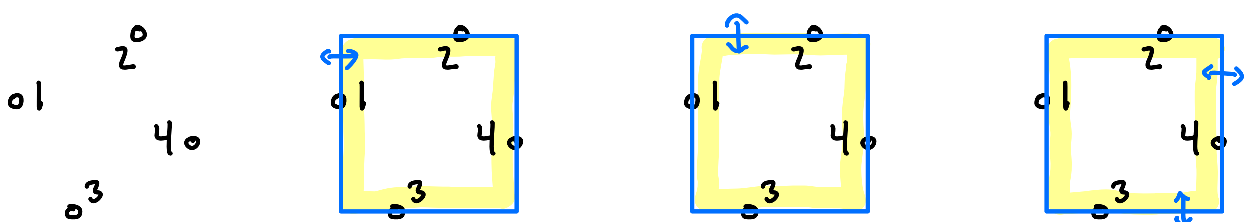
$$\frac{|P\cap Q|}{|P|} = \frac{10}{31} \approx \frac{2}{6} = \frac{|S\cap Q|}{|S|} \qquad \frac{|P\cap Q|}{|P|} = \frac{6}{31} \neq \frac{6}{6} = \frac{|S\cap Q|}{|S|}$$

But this fails if we allow very wild range shapes.
How to formally forbid such ranges?

VC-Dimension:

Shattering: A range space $(X, \mathcal{R})$ shatters a pt set P if $\mathcal{R}_{|P} = 2^P$ (contains all subsets of P)

E.g. Axis-aligned rectangles shatter the pt set below:

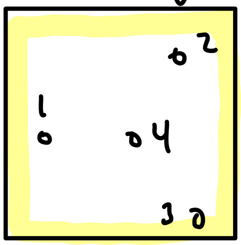


Can include or exclude 1

Can include or exclude 2

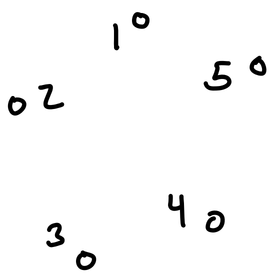
Same for 3 + 4

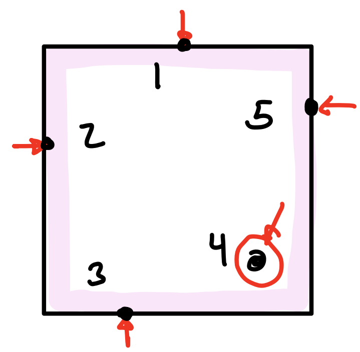
But they can't shatter everything:



Any rect. containg 1,2,3 must contain 4

... and they can never shatter a set of ≥ 5





Any rect that contains the 1,2,3,5 must contain 4

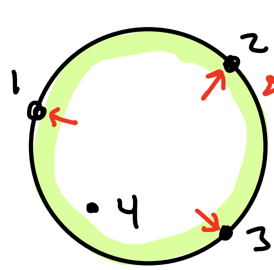
Def: The VC-dimension of a range space $(X, \mathcal{R})$ is the size of the largest pt set shattered by $\mathcal{R}$.

("VC" – Vapnik-Chervonenkis - 1971)

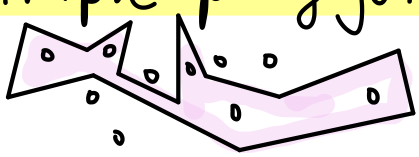
Examples:

→ VC-dim of axis-aligned rects in $\mathbb{R}^2 = 4$

→ VC-dim of Euclidean disks in $\mathbb{R}^2 = 3$



→ VC-dim of simple polygons in $\mathbb{R}^2 = \infty$



Intuitively: Range spaces of constant VC-dim have a constant num. of degrees of freedom

Sauer's Lemma: If $(X, \mathcal{R})$ is a range space of VC-dim d in $|X| = n$, then

$$|\mathcal{R}| = O(n^d)$$

More precisely:

$$|\mathcal{R}| \leq \Phi_d(n)$$

where:

$$\Phi_d(n) = \binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{d}$$

Observe: Φ satisfies the recurrence:

$$\Phi_d(n) = \Phi_d(n-1) + \Phi_{d-1}(n-1)$$

(Exercise)

Proof: (of Sauer's Lemma) Induction on $d + n$.

Basis: $n = 0$ or $d = 0$ - trivial $\mathcal{R} = \{\emptyset\}$

Step: Fix any $x \in X$

Consider two new range spaces: over $X \setminus \{x\}$

$\mathcal{R}_x = \{ Q \setminus \{x\} : Q \cup \{x\} \in \mathcal{R} \text{ + } Q \setminus \{x\} \in \mathcal{R} \}$

↪ Pairs that differ only on x

$\mathcal{R} \setminus \{x\} = \{ Q \setminus \{x\} : Q \in \mathcal{R} \}$

↪ Just remove x

Example: $X = \{1, 2, 3, 4\}$ let $x = 4$

Suppose $\mathcal{R}$ has:		$\mathcal{R}_x$ has:
$\{2,3\}$ + $\{2,3,4\}$	$\longrightarrow$	$\{2,3\}$
$\{1\}$ + $\{1,4\}$	$\longrightarrow$	$\{1\}$
$\{\}$ + $\{4\}$	$\longrightarrow$	$\{\}$

and $\mathcal{R}$ has: $\{1,3\}$ but not $\{1,3,4\}$

$\{2,4\}$ but not $\{2\}$

Then: $\mathcal{R}_x = \{ \{\}, \{1\}, \{2,3\} \}$

$\mathcal{R} \setminus \{x\} = \{ \{\}, \{1\}, \{2,3\}, \{1,3\}, \{2\} \}$

Observe:

- $|\mathcal{R}| = |\mathcal{R}_x| + |\mathcal{R} \setminus \{x\}|$
- $\mathcal{R}_x$ has VC-dim $d-1$
- Both over domain of size $n-1$

$\Rightarrow |\mathcal{R}| \leqslant \Phi_{d-1}(n-1) + \Phi_d(n-1) = \Phi_d(n)$ $\square$

Recall:

Given a discrete range space (P, R) + $\varepsilon > 0$

ε-sample: $S \subseteq P$ is an **ε-sample** if

$$|\mu(Q) - \hat{\mu}(Q)| \leq \varepsilon \quad \forall Q \in R$$

ε-net: $S \subseteq P$ is an **ε-net** if

$$\mu(Q) \geq \varepsilon \Rightarrow S \cap Q \neq \emptyset \quad \forall Q \in R$$

Range spaces of low VC-dimension have ε-samples + ε-nets of small size:

ε-Sample Theorem: Given range space (X, R) of VC-dim d, let P be finite subset of X. There exists constant c s.t. with probability $\geq 1 - \varphi$, a random sample of P of size $\geq$

$$\frac{c}{\varepsilon^2}\left(d \cdot \log \frac{d}{\varepsilon} + \log \frac{1}{\varphi}\right)$$

is an ε-sample for (P, R).

ε-Net Theorem: Given range space (X, R) of VC-dim d, let P be finite subset of X. There exists constant c s.t. with probability $\geq 1-\varphi$, a random sample of P of size $\geq$

$$\frac{c}{\varepsilon}\left(d \log \frac{1}{\varepsilon} + \log \frac{1}{\varphi}\right)$$

is an ε-net for (P, R).

Too many parameters! :(

tl;dr : - Constant VC-dim
- Constant prob. of success

Size of ε-sample is $O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right)$

ε-net is $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$

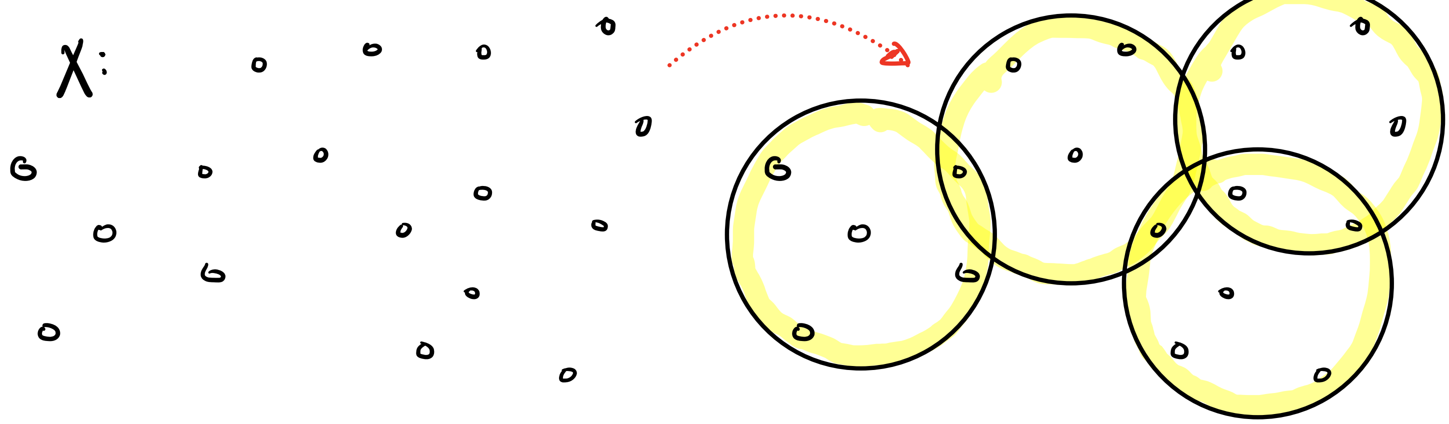
Proofs? See Har-Peled's book

Application: Geometric Set Cover

Given a pt set X + a collection of sets $\mathcal{R}$ over X, a **cover** is a collection of sets from $\mathcal{R}$ that contain every pt of X

E.g. X is a set of n pts in $\mathbb{R}^d$
$\mathcal{R}$ = set of all unit Euclidean balls in $\mathbb{R}^d$

X:



Set cover Problem: Given X and $\mathcal{R}$, find the smallest cover of X

- Set cover is NP-hard
- No known constant factor approximation
- Simple greedy algorithm computes a cover of size $\leq (\ln |X|) \cdot \text{opt}$

Select set that covers the most uncovered pts

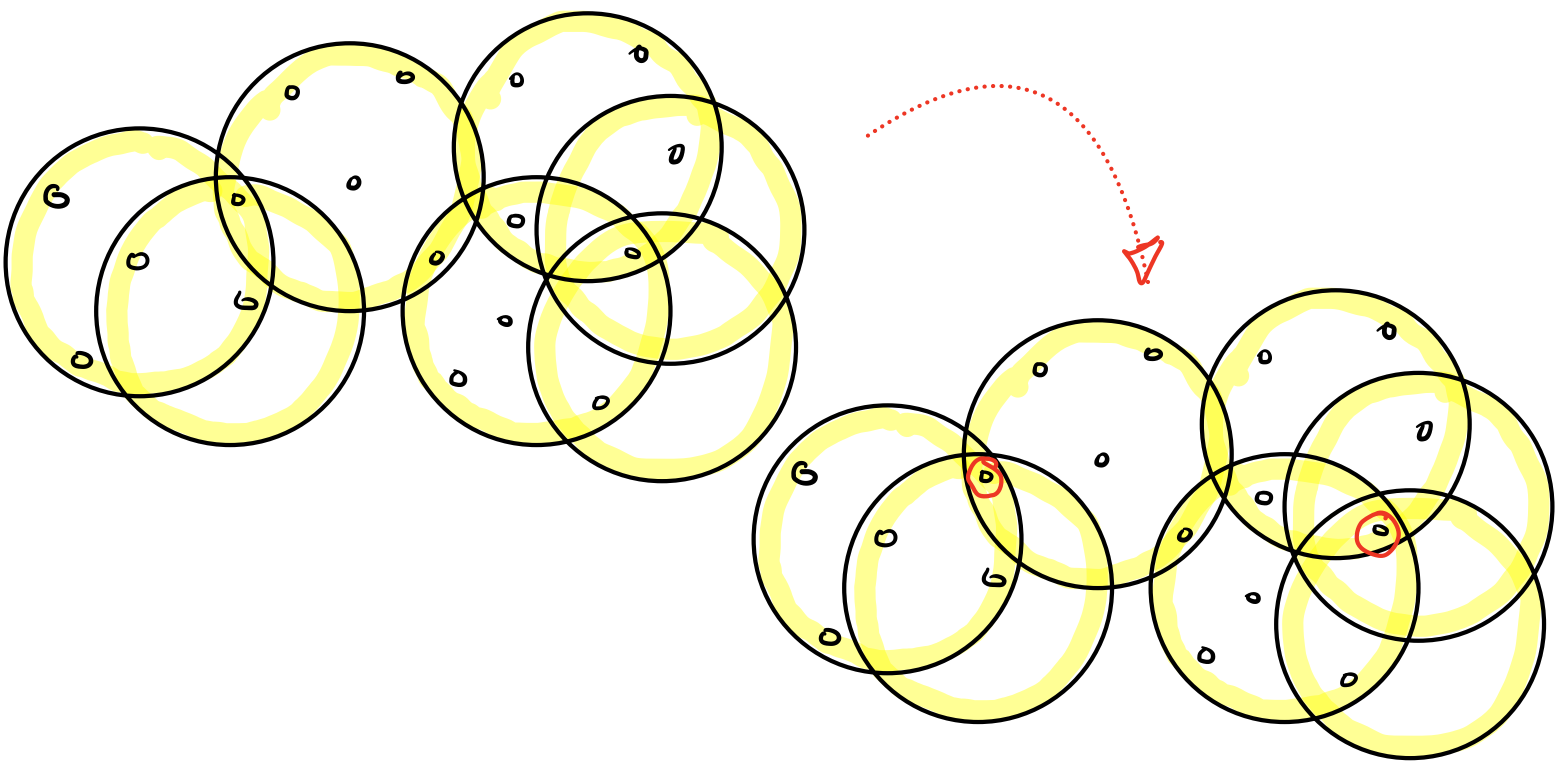
We'll show that if $(X, \mathcal{R})$ is a set system of constant VC-dimension, it is possible to compute an approx. solution of size

$\leq (\log k) \cdot opt$

where k is number of sets in opt. cover
(Note $k < |X|$, so this is always better)

Set cover $\longleftrightarrow$ Hitting Set Duality

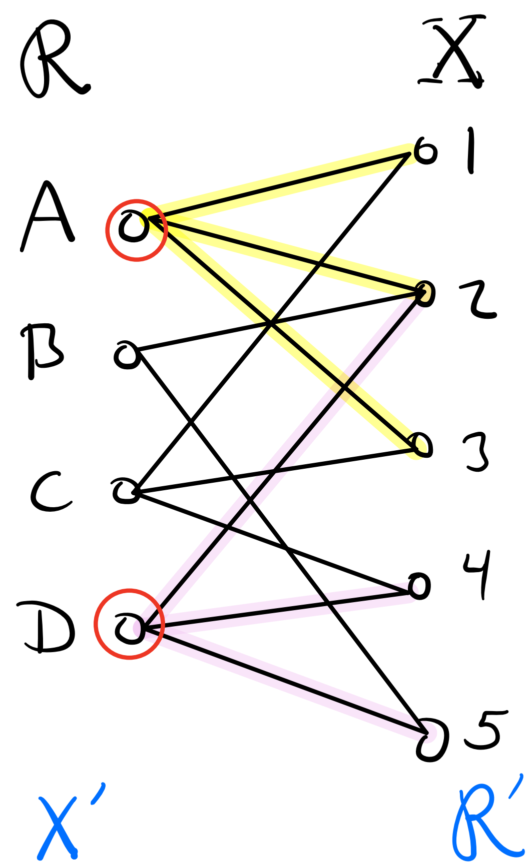
Hitting Set: Given a collection of sets $\mathcal{R}$ over some domain X, a hitting set is a subset of X such that every set of $\mathcal{R}$ contains at least one of them.



Set cover + hitting set are the same problem in disguise

E.g. $A = \{1,2,3\}$ $B = \{2,5\}$
$C = \{1,3,4\}$ $D = \{2,4,5\}$

Cover: $\{A, D\}$



Let's reinterpret: sets $\longrightarrow X'$; pts $\longrightarrow R'$

1: $\{A,C\}$ 2: $\{A,B,D\}$ 3: $\{A,C\}$ 4: $\{C,D\}$ 5: $\{B,D\}$

Hitting set: $\{A, D\}$

Obs: (X, R) has set cover of size k iff (X', R') has hitting set of size k

Theorem: Given a set system $(X, \mathcal{R})$ of constant VC-dimension, in polynomial time it is possible to compute a hitting set of size $O(k^* \log k^*)$ where k^* = size of optimal hitting set.

Note: A set has constant VC-dim iff its dual has constant VC-dim.

Iterative Reweighting:

Weighted ε-Nets: Given a set system $(X, \mathcal{R})$ where each $x \in X$ has a positive weight $w(x)$. Let $w(X)$ be total weight:

$$w(X) = \sum_{x \in X} w(x)$$

A set $S \subseteq X$ is an ε-net if

$$\forall Q \subseteq \mathcal{R} \text{ if } \frac{w(Q \cap P)}{w(P)} \geq \varepsilon \text{ then } Q \cap S \neq \emptyset$$

Standard ε-net ≡ all pts have $w(x) = 1$

Weighted sampling:

ε-Net Theorem still holds, but rather than random sample of size $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ sample each point with probability proportionate to its weight to get a set of this size.

Iterative Reweighting:

- Guess the size k of opt hitting set (binary search to get best k)
- Set all weights to 1
- Repeat:
  - $S \leftarrow$ weighted ε-net of X
  - Is this a hitting set? yes $\rightarrow$ success
  - No? Find any set $Q \subseteq R$ not hit + double weights of all $x \in Q$
- Too many iterations? Fail $\rightarrow$ try larger k

Intuition: If we fail to hit we double weights of unhit object — more likely to hit next time.

Why it works: Critical items (in opt. solution) increase in weight rapidly - eventually they are all selected.

Algorithm: Given $(X, \mathcal{R})$

for $k = 1, 2, 4, \ldots, 2^i, \ldots$ until success

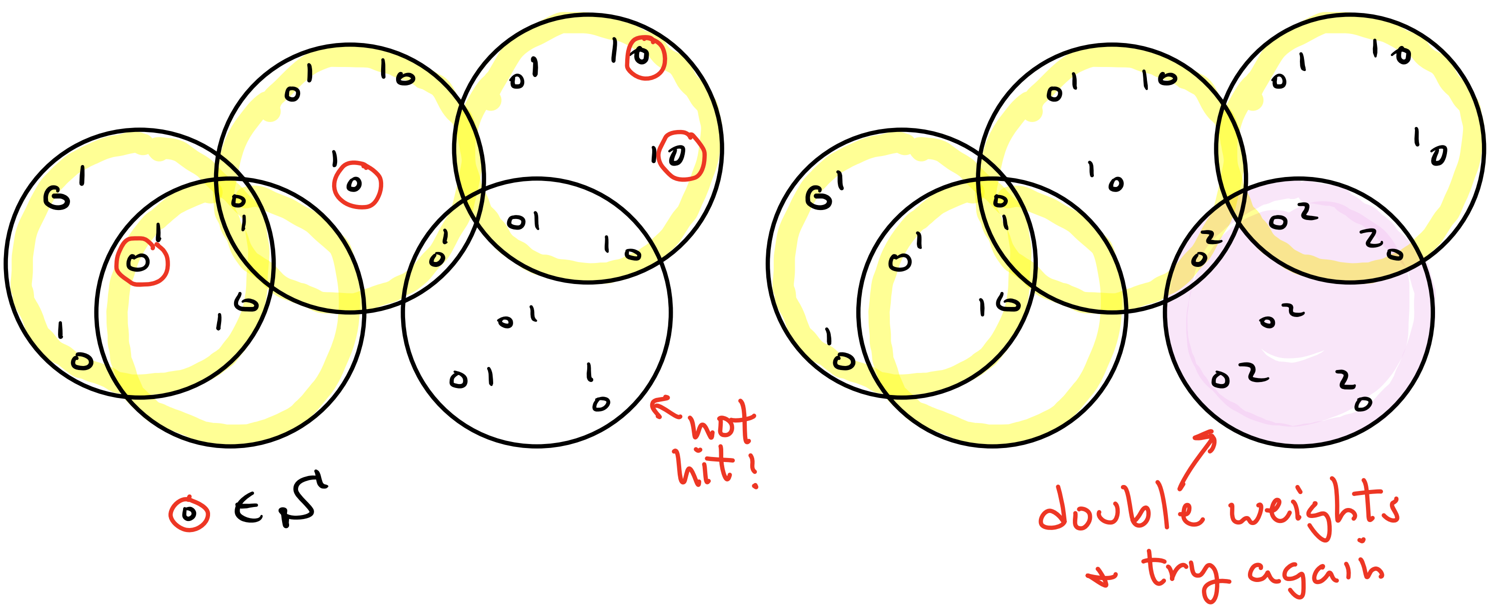
// Guess that $\exists$ hitting set of size k

- $\forall x \in X$ set $w(x) \leftarrow 1$
- Set $\varepsilon \leftarrow 1/4k$

(for suitable const. c)

- Repeat until success or $2k \cdot \lg n/k$ iterations
  - $S \leftarrow$ wgt ε-net of size $c \cdot k \cdot \log k$
  - are all sets of $\mathcal{R}$ hit by S?
    - yes $\rightarrow$ return with success!
    - no $\rightarrow$ find any set $Q \in \mathcal{R}$ not hit

      $\forall x \in Q$, $w(x) \leftarrow 2 \cdot w(x)$



Why this works? Assume k is correct

- Since opt hitting set hits all sets, at least one point of opt doubles in weight
- Weight of opt hitting set grows exponentially fast
- Total weight of pt set grows much more slowly
- Soon, opt hitting set's weight is so high we must sample it.

Lemma: If $(X, \mathcal{R})$ has hitting set of size k, then the repeat-loop has success within $2k \cdot \lg n/k$ iterations. ($\lg \equiv \log_2$)

Proof: Let $n = |X|$ $m = |\mathcal{R}|$

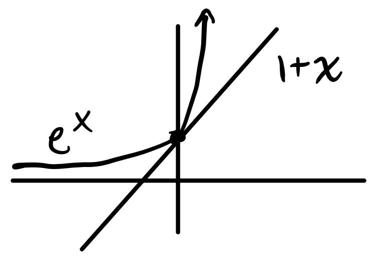
- Let H be hitting set of size k
  $\varpi_i(X)$ = total weight after i^{th} iteration
  $\varpi_i(H)$ = weight of H " " "
- Note: $\varpi_0(X) = |X| = n$

- Since S is an ε-net, if we fail to hit a set Q, then $w_i(Q) < \varepsilon \varpi_i(X)$

$$\Rightarrow \omega_i(X) = \omega_{i-1}(X) + \omega_{i-1}(Q)$$
$$\leq \omega_{i-1}(X) + \varepsilon \cdot \omega_{i-1}(X)$$
$$= (1+\varepsilon)\,\omega_{i-1}(X)$$

$$\Rightarrow \omega_i(X) \leq (1+\varepsilon)^2\,\omega_{i-2}(X)$$
$$\leq (1+\varepsilon)^3\,\omega_{i-3}(X)$$
$$\vdots$$
$$\leq (1+\varepsilon)^i\,\omega_0(X) = (1+\varepsilon)^i \cdot n$$

Fact: $1+x \leq e^x$



$$\Rightarrow \omega_i(X) \leq n \cdot e^{i \cdot \varepsilon}$$

Since H hits all sets, it hits Q

$\Rightarrow$ in each (unsucessful) iteration, at least one element of H doubles

$\Rightarrow$ growth rate of $\omega_i(H)$ is slowest if all its members double at same rate (Jensen's Ineq.)

$\Rightarrow$ After i^{th} iteration, each of the k elements of H doubled i/k times

$$\Rightarrow \omega_i(H) \geq k \cdot 2^{i/k}$$

Since $H \subseteq X$, we know $W_i(H) \le W_i(X)$

$$\Rightarrow \quad k \cdot 2^{i/k} \le n \cdot e^{i \cdot \varepsilon}$$

Recall, we set $\varepsilon \leftarrow 1/4k$

$$\Rightarrow \quad k \cdot 2^{i/k} \le n \cdot e^{i/4k}$$

lg e < 2

$$\Rightarrow \quad \lg k + \frac{i}{k} \le \lg n + \frac{i}{4k} \lg e$$
$$\le \lg n + \frac{i}{2k}$$
$$\Rightarrow \frac{i}{k} - \frac{i}{2k} = \frac{i}{2k} \le \lg n - \lg k = \lg \frac{n}{k}$$
$$\Rightarrow \quad \text{No. of iterations } i \le 2k \cdot \lg \frac{n}{k}$$

(If we exceed this number, we know $|H| > k$, and we fail) $\square$

Total time:

$$\left(2k \cdot \log \frac{n}{k}\right) \cdot \left[\left(k \cdot \log k\right) + m \cdot k\right]$$

since $k \le n$

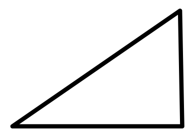
$$= O\left(n^2 \cdot m \cdot \log n\right)$$

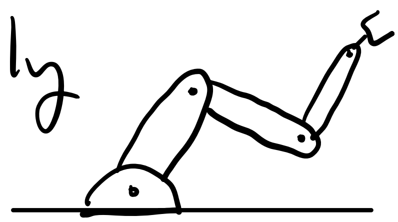
CMSC 754 - Computational Geometry
Lecture 20 - Motion Planning

Motion Planning:

Given a robot (with constraints on how it can move), a set of obstacles, and a start + target configurations for the robot, is there a collision-free motion plan?

Robot: May be rigid object or linked/hinged assembly

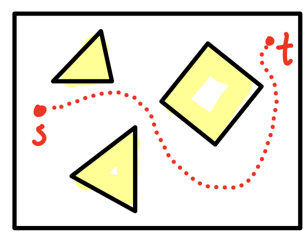




Motion constraints:

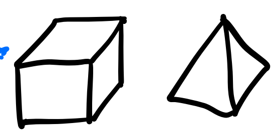
- Translation
- Rotation
- Speed/Acceleration limits

⋮



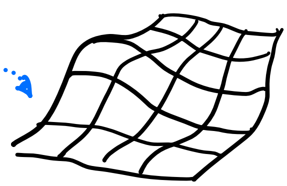
Obstacles: Polygons in 2-D

Polyhedra in 3-D



Curved objects

Terrains



We'll mostly consider the simplest scenario:

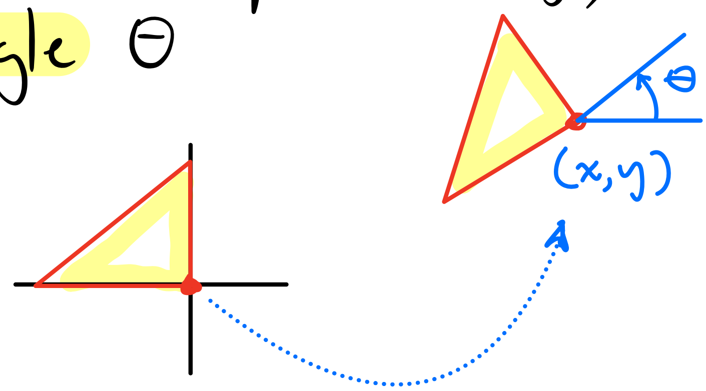
- **Space** – $\mathbb{R}^2$
- **Robot** – (convex) polygon
- **Motion** – translation only
- **Obstacles** – collection of nonoverlapping convex polygons

Configuration: A set of parameters that uniquely specifies the robot's position

E.g. **Rigid in 2-D**

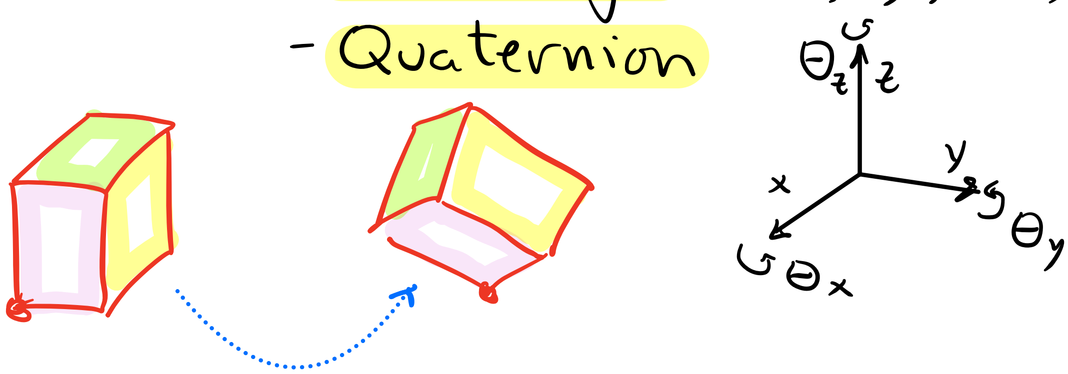
- location of reference point (x, y)
- rotation angle θ

Reference position:

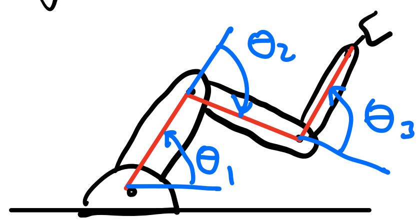


Rigid in 3-D

- location (x, y, z)
- rotation
  - Euler angles $(\theta_x, \theta_y, \theta_z)$
  - Quaternion



Linked/Hinged: Joint angles
$(\theta_1, \theta_2, \theta_3)$



Motion Planning in Config. Space:

- Rather than moving a robot amidst obstacles

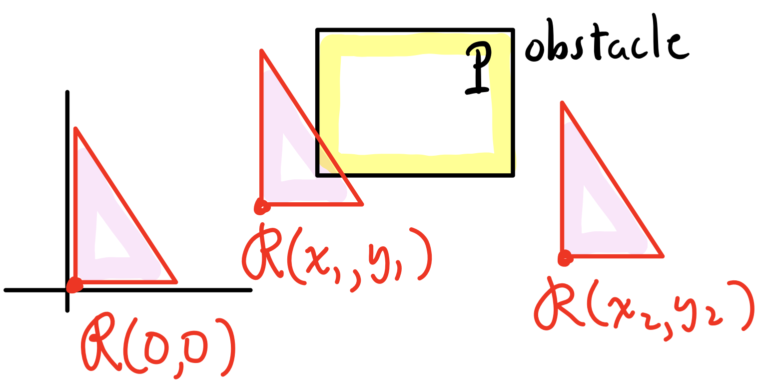
– instead –

- Move a point in the robot's configuration space

Need to distinguish between:

free configuration - robot does not collide
forbidden configuration - robot collides

E.g. Translation only (configuration = location of ref. point)



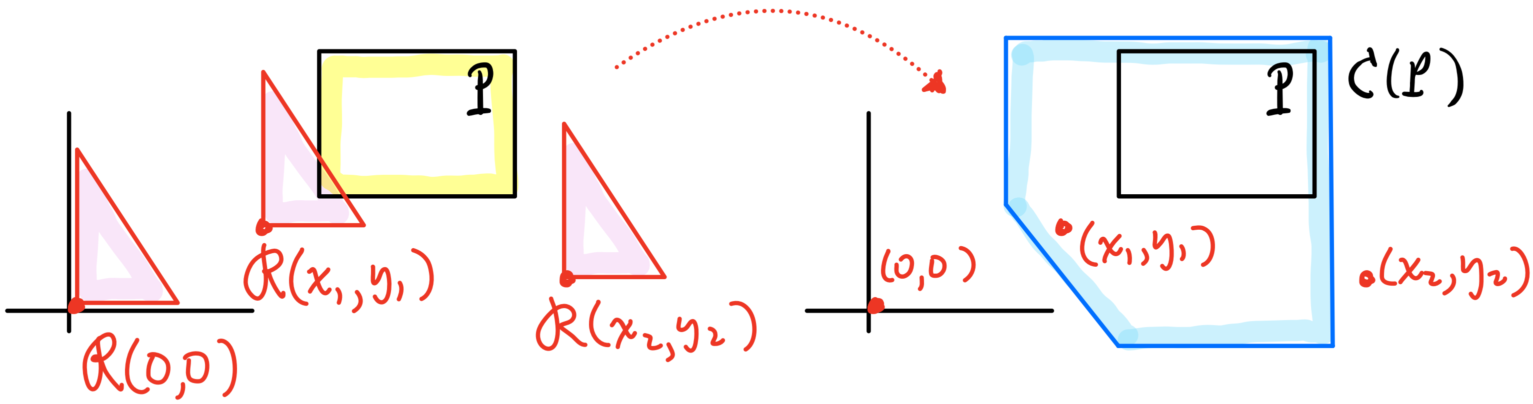
(x_1, y_1) - forbidden
$\mathcal{R}(x_1, y_1) \cap \mathcal{P} \neq \emptyset$

(x_2, y_2) - free
$\mathcal{R}(x_2, y_2) \cap \mathcal{P} = \emptyset$

Configuration Obstacle (or C-Obstacle)

Given robot $\mathcal{R}$, config vector v, obstacle P the C-obstacle for P is:

$$C_{\mathcal{R}}(P) = \{ v \mid \mathcal{R}(v) \cap P \neq \emptyset \}$$



C-Obstacles for Translation - Minkowski Sum

The easiest C-obstacles are for translational motion.

Def: Given $P, Q, S \subseteq \mathbb{R}^d$ + $\alpha \in \mathbb{R}$

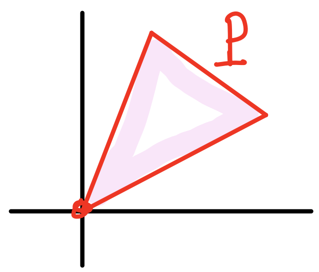
$$P \oplus Q = \{ p + q : p \in P, q \in Q \}$$

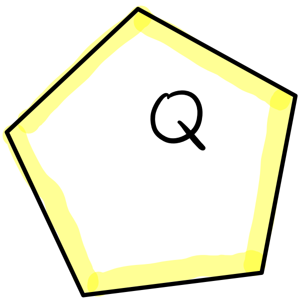
Minkowski Sum

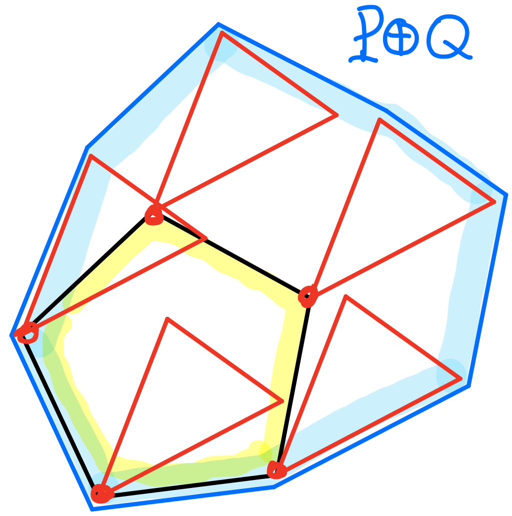
$$\alpha P = \{ \alpha \cdot p : p \in P \}$$

$$-P = \{ -p : p \in P \}$$

Intuition: $P \oplus Q$ - Place P so its ref. pt. is at origin
- Sweep P's ref pt around Q + see what's swept out







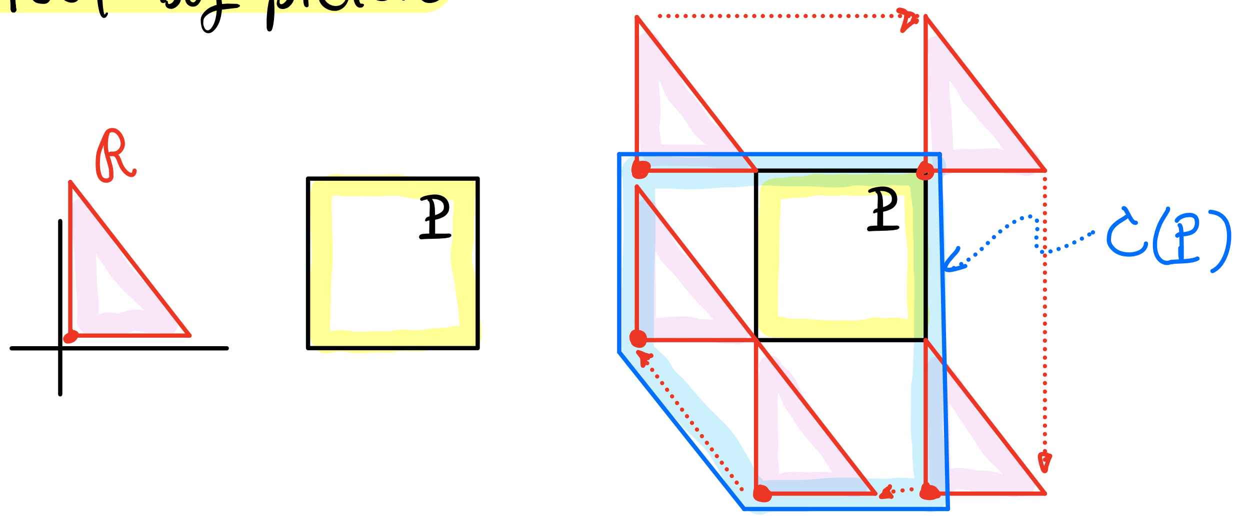
Lemma: Given a translating robot R + obstacle P:

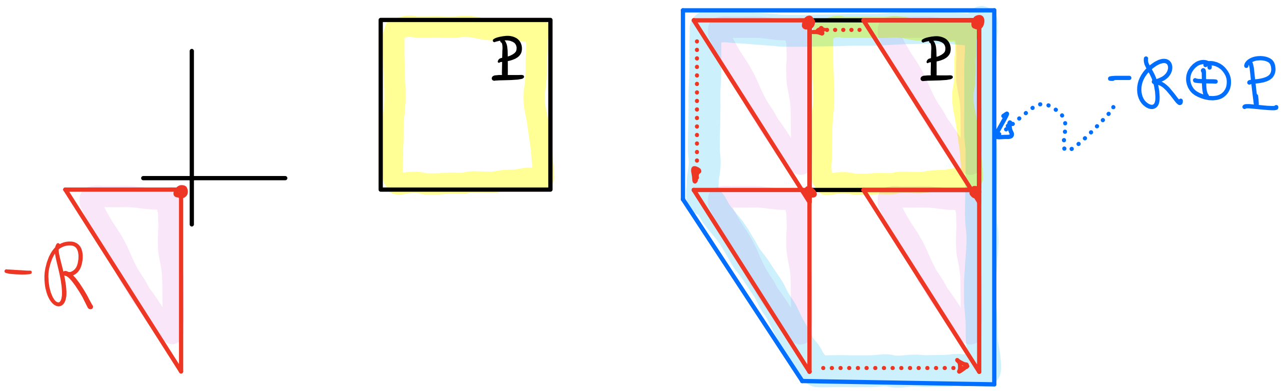
$$\boxed{C_R(P) = P \oplus (-R)}$$

Proof: For any translation vector t

$$
\begin{aligned}
t \in C(P) &\iff R(t) \text{ collides with } P \\
&\iff R+t \cap P \neq \emptyset \\
&\iff \exists r \in R,\ p \in P \quad r+t = p \\
&\iff \quad '' \qquad '' \qquad t = p - r \\
&\iff t \in P \oplus (-R)
\end{aligned}
$$

Proof by picture:

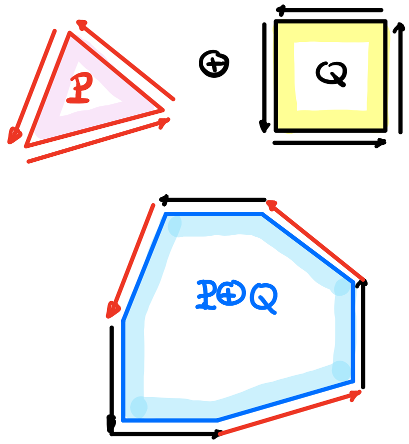




Computing the Minkowski Sum:

If P is a convex m-gon
Q is a convex n-gon
can compute $P \oplus Q$ in time $O(m+n)$

- Direct edges CCW (vectors)
- Sort them by angle
- Join them tail to head



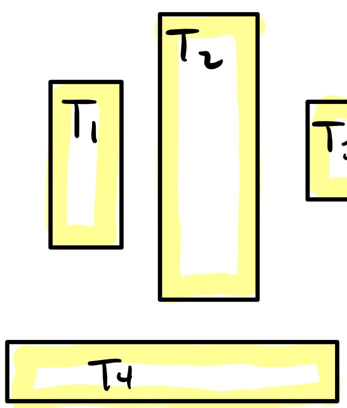
Complexity of C-Obstacles:

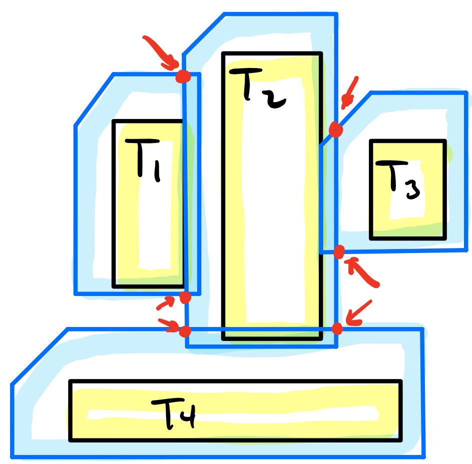
- Suppose we have an m-sided convex robot R and a collection of disjoint convex obstacles $T_1, \dots, T_k$. Let n_i = num. of sides in T_i. Let $n = \sum n_i$

- What is total size of config. obstacles?

$$\bigcup_{i=1}^{k} C_R(T_i) = \bigcup_{i=1}^{k} T_i \oplus (-R)$$

- Although T_i's are disjoint, $C_R(T_i)$ may overlap

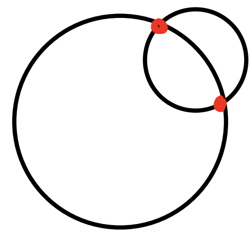




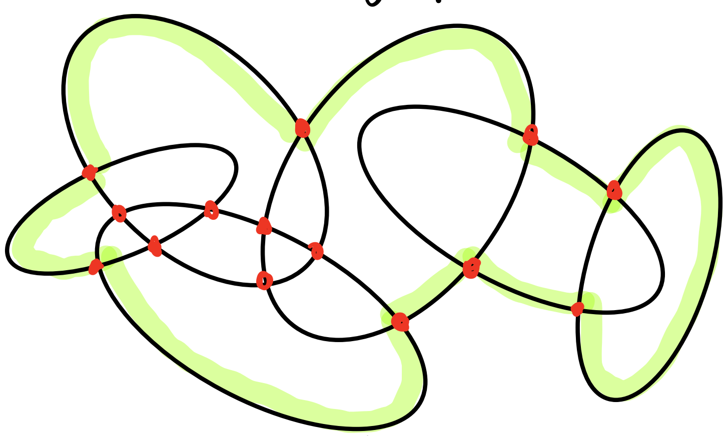
- Points of boundary overlaps create additional vertices - How many? $O(n)$ $O(n^2)$?

Pseudodisks:

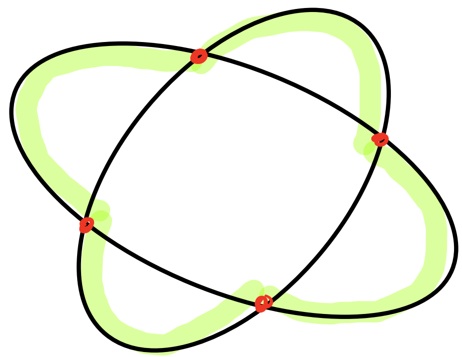
- The boundaries of two circular disks intersect at most twice.



- A collection of convex objects $\{o_1, \dots, o_k\}$ is a collection of pseudodisks if the boundaries of any pair intersect at most twice.



Collection of pseudodisks



Not pseudodisks

Lemma: Given a set $T_1, \dots, T_k$ of disjoint convex bodies in $\mathbb{R}^2$ and convex R

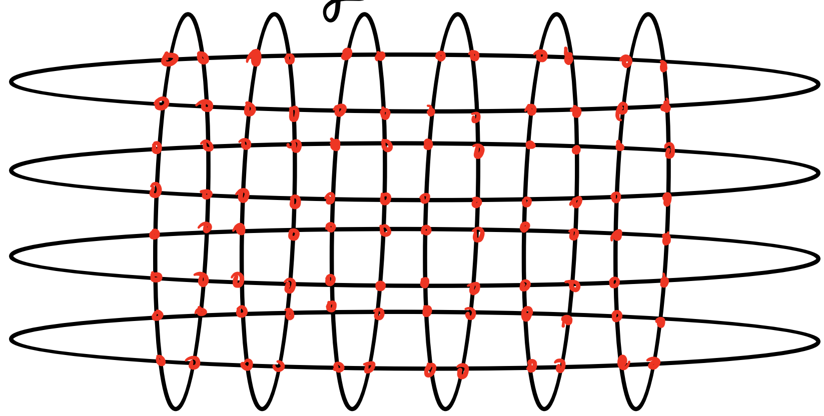
$$\{C_R(T_1), \dots, C_R(T_k)\} = \{T_1 \oplus (-R), \dots, T_k \oplus (-R)\}$$

is a collection of pseudodisks.

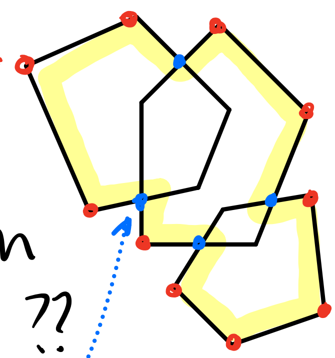
Proof: See latex notes

Theorem: Given a collection of pseudodisks with a total of n vertices, their union has a total of $O(n)$ vertices.

In general, union may have $O(n^2)$ vertices

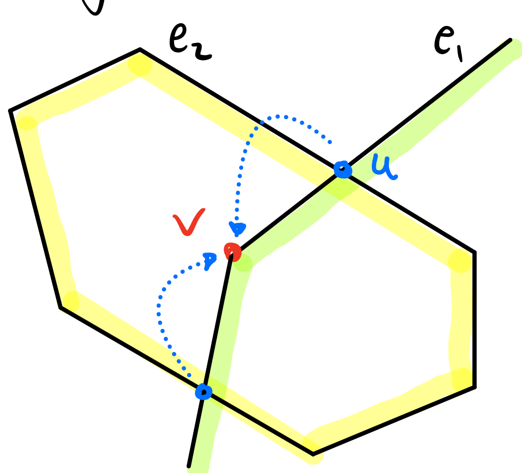


Vertex types:



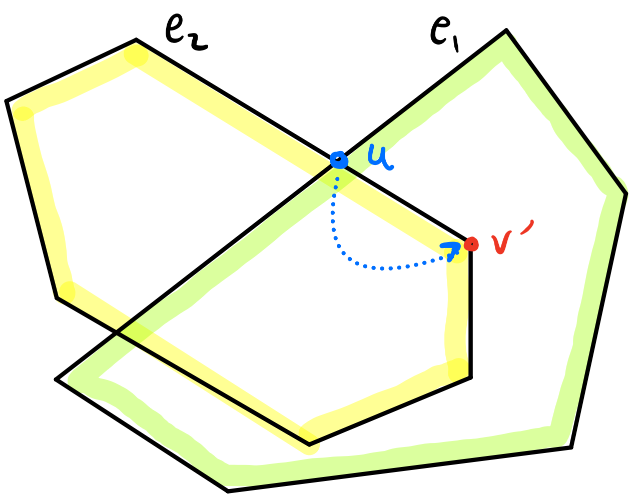
- Vertices of original polygons – n
- Vertices caused by intersections – ??

- We'll "charge" intersection vertices to vertices hidden in the interior
- Suppose edges e_1 + e_2 intersect at u



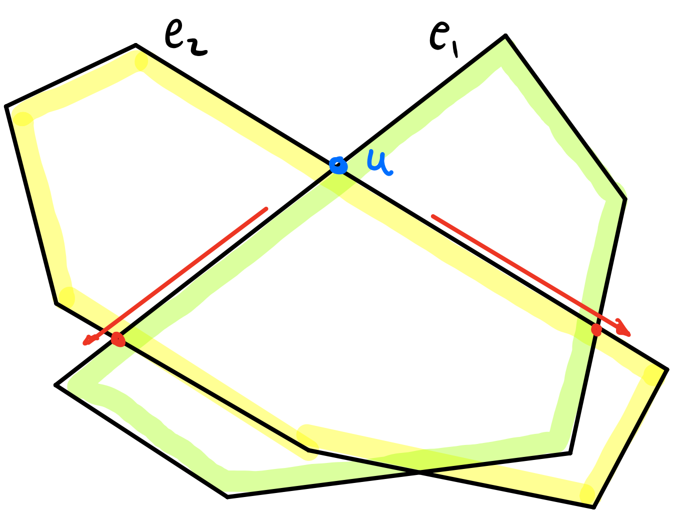
- if e_1 leads to internal vertex v, charge u to v
- v gets ≤ 2 charges

- Otherwise, if e_1 cuts through, but e_2 leads to internal vertex v', charge u to v'



(Again v' can be charged at most twice)

- Otherwise both e_1 + e_2 cut through the other polygon



But this cannot happen since these are pseudodisks!

Since every vertex is charged at most twice union has at most $2n$ vertices. $\square$

Theorem: Given a convex m-sided robot and and a collection of n disjoint obstacles, each with $O(1)$ sides, the total boundary complexity of the union of C-obstacles is $O(m \cdot n)$

Proof: We have a collection of n pseudodisks each with $O(1) + m = O(m)$ sides.

$\Rightarrow$ Total vertices is $O(m \cdot n)$

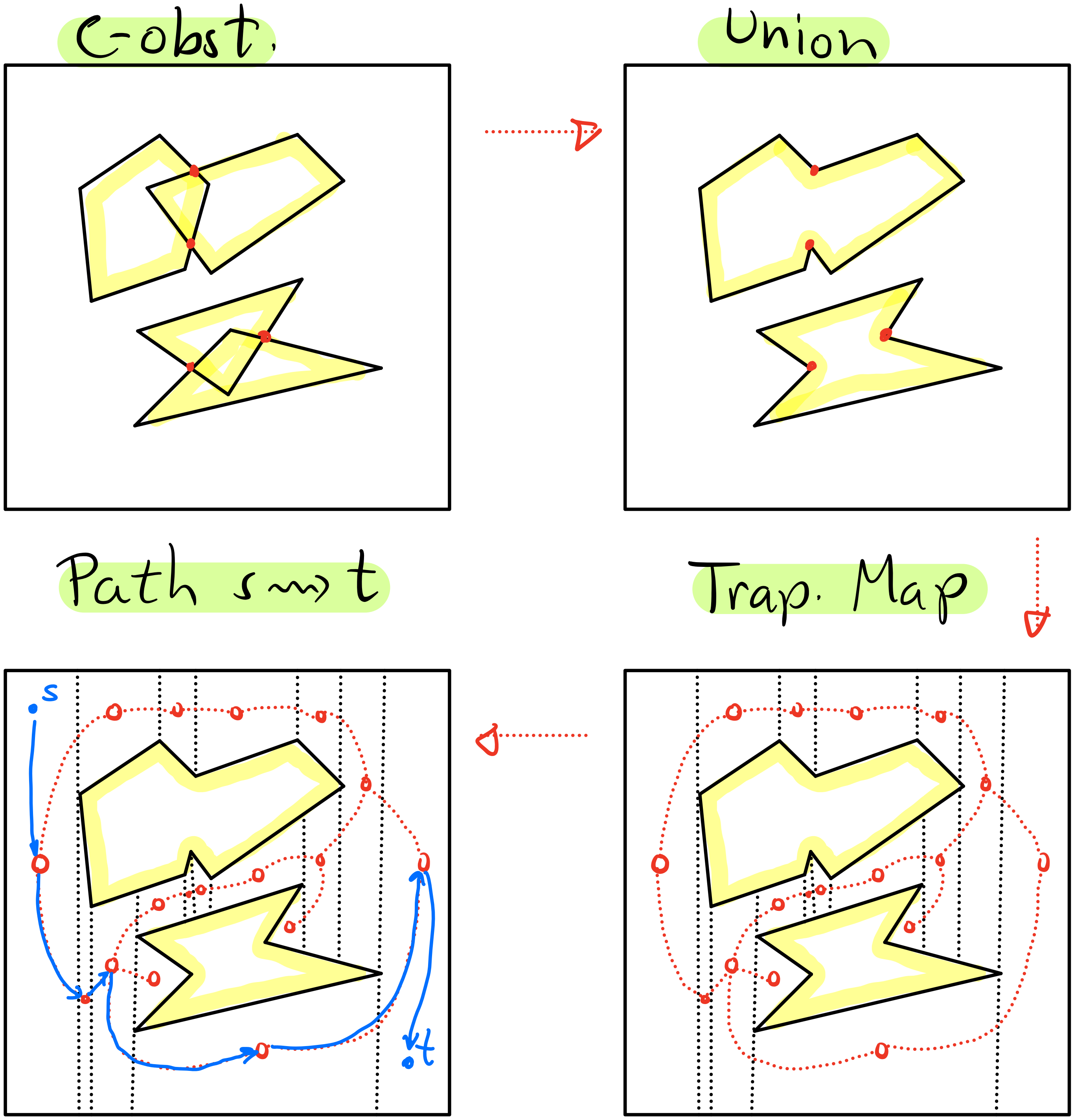
$\Rightarrow$ Union complexity is $O(m \cdot n)$.

Path Planning in Config Space:

Once we have computed the C-obstacles, how to find a path between start + target?

- Compute union of C-obstacles
- Compute a decomposition of the compliment space (outside the C-obstacles)

  Eg. Triangulate or trapezoid map
- Compute dual graph, joining pairs that can reach each other



Finally: Given start s + target t,
- find trapezoids containing them
- if reachable in dual graph
  - create path joining them
- else – output "unreachable"

Note: Not the shortest path