Chan '96: Convex hull in $O(n \log h)$.

Compare to Graham $O(n \log n)$, or Jarvis $O(n h)$.

$h$: # vertices on boundary of convex hull

1) $h^*$: a guess for value of $h$

1) Break up input set $P$ into $k = \lceil n/h^* \rceil$ of site at most $h^*$.

Run Graham scan on each subset to make $k$ mini-hulls.
Take $O(k(h^* \log h^*)) = O(n \log h^*)$ time.

Observations:

(a) Every vertex of $\text{conv}(P)$ is a vertex of some mini-hull.

(b) Suppose $p$ is on $\text{conv}(P)$. The next vertex $q$ is on a mini-hull $H$, $pq$ is a support line for $H$.

(c) Suppose $H$ has $m$ vertices. Can find both support lines.
in $O(\log m)$ time.

So to "merge" the mini-hulls, run Jarvis march for $p$. For each $v_i$ on $\text{conv}(p)$, consider the $\leq 2k$ vertices that make support lines between $v_i$ and its mini-hull.
Takes $O(k \log h^*) + O(k)$ time per vertex of conv($P$).

So merging takes

$$O(h(k \log h^*))$$

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

time.
$h^*$ is a guess for $h$
if $h^* > h$, both phases take $O(n \log h^*)$,
could be large, but if $h < h^* \leq h^2$, then
$O(n \log h^*) = O(n \log h)$.

if $h^* \leq h$, mini-hulls take $O(n \log h^*) = O(n \log h)$,
but march could take very long

... but you can start if you add $2h^*$ vertices to $\text{conv}(P)$
Chan’s Algorithm for the Conditional Hull Problem

**ConditionalHull**(\(P, h^*\)):

1. Let \(k \leftarrow \lceil n/h^* \rceil\). Partition \(P\) (arbitrarily) into disjoint subsets \(P_1, \ldots, P_k\), each of size at most \(h^*\).
2. For \(j \leftarrow 1\) to \(k\), compute \(H_j = \text{conv}(P_j)\) using Graham’s scan, storing each in an ordered array.
3. Let \(v_0 \leftarrow (-\infty, 0)\), and let \(v_1\) be the bottommost point of \(P\).
4. For \(i \leftarrow 1, 2, \ldots, h^*\):
   a. For \(j \leftarrow 1\) to \(k\), using the utility lemma, compute the tangents points \(q_j^-\) and \(q_j^+\) for \(H_j\) with respect to \(v_{i-1}\).
   b. Set \(v_i\) to be the tangent point that minimizes the turning angle with respect to \(v_{i-2}\) and \(v_{i-1}\).
   c. If \(v_i = v_1\) then return the pair \((\text{success}, V = (v_1, \ldots, v_{i-1}))\).
5. If we get here, we know that \(h^* < h\), and we return \((\text{failure}, \emptyset)\).

\[\text{Takes } O(n \log h^*) \text{ time.}\]

Start with \(h^* = 4\) and increase the square \(h^*\) for each new guess.

---

**Hull**(\(P\)):

1. \(h^* \leftarrow 2\); status \(\leftarrow\) failure
2. while (status \(\neq\) failure):
   a. Let \(h^* \leftarrow \min((h^*)^2, n)\)
   b. (status, \(V\)) \(\leftarrow\) ConditionalHull\((P, h^*)\)
3. return \(V\)
Will succeed first time if \( h_0 \geq h \). At that time \( h_0 = \frac{h}{2} \).

Analysis

Let \( h_0 = 2^u \).

Conditional Hall run \( w \) takes

\[
O(n \log h_0) = O(n \log 2^u) = O(n 2^u)
\]

\[ \log i = \log_2 \]

Total time is...

\[
O(n), \quad \sum_{i=1}^{\log h} 2^u = O(n) \cdot O(2^{\log h}) = O(n \log h)
\]

↑ geometric (prop. to largest term)
Cannot do better!
Line Segment Intersection

Given a set $S$ of $n$ line segments in $\mathbb{R}^2$.

Want to report all intersections.

Could check all $\binom{n}{2} = \Theta(n^2)$ pairs for intersection.
Can we do better if \( \# \text{intersections} \) is small? Yes.

**Assumptions:**

General-position assumptions
- no duplicate \( x \)-coordinates
- no vertical segments
- no 3-way intersections
- no endpoint lies on another segment
Plane Sweep Algorithm

\( n := |S| \quad m := \# \text{ intersections} \)

Bentley & Ottmann [‘79]:

\[ O((n+m) \log n) \]

A vertical line \( l \) moves from \(-\infty \) to \( \infty \), left to right...

We'll note when something changes
about intersections between $l$ and segments.

At certain moments (x-coordinates) something happens. Call these moments event points.

Specifically track...

1) the partial solution of segment intersections left of $l$

2) the sweep-line status into on objects intersecting $l$

(Here, segments intersecting $l$ order top to bottom by intersection)

3) a subset of future events
Including next event (here, upcoming endpoints & some segment intersections)

- discovered intersection point
- future endpoint event
- future intersection point event (only some will be stored)
Detecting Events

Status changes at event points

1) left endpoints ← sort these
2) right endpoints ← left-to-right
3) segment intersections ↑

Can't easily sort.
We still need to find them!

Only need to know of next event!
Lemma: Suppose segments $s_i$ and $s_j$ intersect at point $p$.

Proof: By assumption, no three segments share a common point.

So $s_i$ and $s_j$ are adjacent on sweep line immediately after event before $p$. 

$q$: event point just before $p$. 

$s_i$ and $s_j$ on sweep line status immediately after event before $p$. 

$s_i$ and $s_j$ are adjacent on sweep line immediately lost of $p$. 

$q$: event point just before $p$. 

Lemma: Suppose segments $s_i$ and $s_j$ intersect at point $p$. 

Proof: By assumption, no three segments share a common point.

So $s_i$ and $s_j$ are adjacent on sweep line immediately after event before $p$. 

$q$: event point just before $p$. 

Lemma: Suppose segments $s_i$ and $s_j$ intersect at point $p$.

Proof: By assumption, no three segments share a common point.

So $s_i$ and $s_j$ are adjacent on sweep line immediately after event before $p$. 

$q$: event point just before $p$. 

Lemma: Suppose segments $s_i$ and $s_j$ intersect at point $p$. 

Proof: By assumption, no three segments share a common point.

So $s_i$ and $s_j$ are adjacent on sweep line immediately after event before $p$. 

$q$: event point just before $p$.
No segments start, stop, or cross between q and p, so $s_i + s_j$ are adjacent just after q.

So if a next event is an intersection, it's between two adjacent segments on sweep line.

So keep an event queue of future events: contains all left or right segment endpoints, contains future intersections between adjacent segments on sweep line.
The number of additions and deletions from the queue is proportional to the number of events, i.e., \( \# \text{events} = O(n+m) \).