Convex Hull of Simple Polygon

Ideas?
Remember Graham scan?

1) Sort
2) Locally establish convexity
Step 1 of Graham scan essentially constructs a polygon (what kind?)
Let's run part 2 again.

The local checking can be visualized with 3 coins:

1 2 3
Let's run part 2 again

The local checking can be visualized with 3 cows

Initial placement starting with two extreme points

→ must make a 3 turn
Let's run part 2 again.

The local checking can be visualized with 3 coins.

Advance while 5.
Let's run part 2 again.

The local checking can be visualized with 3 coins.

Advanced until C

pop
Let's run part 2 again

The local checking can be visualized with 3 cows:

1 2 3

popped & backtracked
Let's run part 2 again

The local checking can be visualized with 3 coins
Let’s run part 2 again.

The local checking can be visualized with 3 coins.

1 2 3
Let's run part 2 again.

The local checking can be visualized with 3 cows.

1 2 3
Let's run part 2 again

The local checking can be visualized with 3 coins

1 2 3
Let's run part 2 again

The local checking can be visualized with 3 coins

1 2 3
Let's run part 2 again.

The local checking can be visualized with 3 coins.
Let's run part 2 again

The local checking can be visualized with 3 coins

This is the 3-coins algorithm
a.k.a. Sklansky scan

Remember it's O(n) after polygonizing
Another example

Notice that polygon order ≠ angular order
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example
Another example

etc
Once more
This method can fail
Notice that if we scanned clockwise, we'd have the opposite result.

What is "deep"?
Externally visible

$P_7 \ldots P_{14}$ : POCKET

$P_7 P_{14}$ : LID

$P_1 \ldots P_5$ : POCKET

$P_{17} \ldots P_{20}$ : pocket
Not externally visible, but weakly externally visible (WEV)

Every point in the pocket is visible from:

- Some point outside CH
- Some point on LID
CLAIM: Sklansky scan works on W.E.V.

• No convex hull vertex can be deleted \( \iff \) we only delete C

• Treat each pocket separately
  \( \rightarrow \) can use induction
  \( \rightarrow \) You are given a WEV pocket
  \( \rightarrow \) Assume smaller WEV pockets can be handled
  \( \rightarrow \) Discard a vertex to get a smaller WEV pocket
Claim: a pocket is WEV if its vertices are WEV.

For every edge, endpoints s, t, see s', t' on lid.

2 main cases:

In all cases, every interior point sees lid.
Base case: trivial

When we delete a right turn: we join. 

\( e \) doesn't intersect polygon. This would imply: 
\( \Rightarrow \) one of the endpoints of \( e \) would not be wex

So we get a simple polygon but also every vertex is still wex.

\( \Rightarrow \) no ray to infinity crossed through cut triangle.
MELKMAN'S ALGORITHM

• Note: not the first $O(n)$ C.H. algo, but it is the simplest.
• It is an online algorithm: at any time you have the C.H. of the current input. No pre-scanning.
$P_1 \ldots P_n$: polyline

$P_k$: last vertex
to make it onto hull
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

L: 1st hull vertex CCW from $P_k$

R: 1st hull vertex CW from $P_k$
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

Red region: left of $L_{P_k}$ and left of $R_{P_k}$

$L$: 1st hull vertex CCW from $P_k$

$R$: 1st hull vertex CW from $P_k$
\( P_1 \ldots P_n : \text{polyline} \)

\( P_k : \text{last vertex to make it onto hull} \)

**Red region:** left of \( \overrightarrow{L_{P_k}} \) and left of \( \overrightarrow{R_{P_k}} \)

**Green region:** right of \( \overrightarrow{L_{P_k}} \) and right of \( \overrightarrow{R_{P_k}} \)

\( L : 1\text{st hull vertex CCW from } P_k \)

\( R : 1\text{st hull vertex CW from } P_k \)
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

Red region: left of $\overrightarrow{L_{P_k}}$ and left of $\overrightarrow{R_{P_k}}$

Green region: right of $\overrightarrow{L_{P_k}}$ and right of $\overrightarrow{R_{P_k}}$

Blue region: Red $\cap$ Green

L: 1st hull vertex CCW from $P_k$

R: 1st hull vertex CW from $P_k$
After finding $P_k$, ignore all vertices (if any) that do not fall in RGB regions.
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

L: 1st hull vertex CCW from $P_k$

R: 1st hull vertex CW from $P_k$

---

need to update hull

---

After finding $P_k$
ignore all vertices (if any)
that do not fall in RGB regions.

Eventually, we scan all input or we get a new hull vertex.
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

After finding $P_k$, ignore all vertices (if any) that do not fall in RGB regions.

Eventually, we scan all input or we get a new hull vertex.
After finding $P_k$
ignore all vertices (if any) that do not fall in RGB regions.
Eventually, we scan all input or we get a new hull vertex.
$P_1 \ldots P_n$: polyline

$P_k$: last vertex to make it onto hull

L: 1st hull vertex CCW from $P_k$

R: 1st hull vertex CW from $P_k$

If we get a new hull vertex, $P_{n+1}$
$P_1...P_n$: polyline

$P_k$: last vertex to make it onto hull

If we get a new hull vertex, $P_{n+1}$:
- If in red, then restore convexity by popping at top.

L: 1st hull vertex CCW from $P_k$
R: 1st hull vertex CW from $P_k$
\(P_1 \ldots P_n: \) polyline

\(P_k: \) last vertex to make it onto hull

if we get a new hull vertex, \(P_{n+1}\)

if in green then restore convexity by popping at bottom

\(L: \) 1st hull vertex CCW from \(P_k\)

\(R: \) 1st hull vertex CW from \(P_k\)
$P_1, ..., P_n$: polyline

$P_k$: last vertex to make it onto hull

L: 1st hull vertex CCW from $P_k$

R: 1st hull vertex CW from $P_k$

if we get a new hull vertex, $P_{n+1}$

if in red then restore convexity by popping at top

if in green then restore convexity by popping at bottom

finally push $P_{n+1}$
if we get a new hull vertex, $P_{n+1}$

- if in red then restore convexity by popping at top
- if in green then restore convexity by popping at bottom

finally push $P_{n+1}$
If we get a new hull vertex, $P_{n+1}$

- If in red, then restore convexity by popping at top.
- If in green, then restore convexity by popping at bottom.
- Finally push $P_{n+1}$.
if we get a new hull vertex, \( P_{n+1} \)

- if in red then restore convexity by popping at top
- if in green then restore convexity by popping at bottom

finally push \( P_{n+1} \)
if we get a new hull vertex, P_{n+1}

if in red then restore convexity by popping at top

if blue
  if in green then restore convexity by popping at bottom
finally push P_{n+1}
if we get a new hull vertex, $p_{n+1}$

if in red then restore convexity by popping at top

if blue

if in green then restore convexity by popping at bottom

finally push $p_{n+1}$
If we get a new hull vertex, \( P_{n+1} \):

- If in red, then restore convexity by popping at top.
- If in green, then restore convexity by popping at bottom.

Finally, push \( P_{n+1} \).
MELKMAN'S ALGORITHM

time: $\Theta(n)$: popping can be amortized (charged to deletion)

- Start w/ 3 points
- Store them in a DEQUE:
  - WLOG let 123 be $\rightarrow$
  - then: bottom of deque 3,1,2
  - top of deque 3,2,1

$P_k$