Convex Hulls continued

We have seen

Jarvis march $O(n \cdot h)$
Graham scan $O(n \cdot \log n)$
Quick hull $O(n^2)$ ... $O(n \cdot \log n)$ expected
QUICKHULL in a little more detail

- What is the worst case?

[Diagram of a convex hull with points and lines]

Form 0 zero deletions
**QuickHull** in a little more detail

- What is the worst case?

Search extremes
Find only one
Zero deletions
QuickHull in a little more detail

- What is the worst case?

- Search extremes
- Find only one
- Zero deletions
Quickhull in a little more detail

- What is the worst case?

Search extremes
Find only one
Zero deletions
Etc
... $O(n^2)$
QuickHull in a little more detail

Also output-sensitive? $t(n, h)$

Notice that every new search happens when we find a convex hull vertex $O(n \cdot h)$
More convex hull algorithms

Incremental

First test if \( O(\log n) \)

Is yes? Then ignore

Update hull:
- Find 2 tangents
- Delete chain between

\( \text{P}_n+1 \)

\( \text{CH}(n) \)
Binary search to find tangents
Incremental C.H.

Could also sort \( x \) and add points in order

1. Suppose you have C.H. \((p_1, \ldots, p_n)\)
**Incremental C.H.**

Could also sort \( x \) and add points in order.

- Suppose you have \( C.H.(p_1 \ldots p_n) \)
- Start with \( p_n \) \( \rightarrow p_{n+1} \)
Incremental C.H.

Could also sort and add points in order

1) Start with \( p_n \)  \( p_{n+1} \)
2) Pop vertices

\( \Rightarrow \) as w/ Graham scan
**Incremental C.H.**

Could also sort and add points in order:

1) Start with $P_n\rightarrow P_{n+1}$
2) Pop vertices

$\Rightarrow$ as with Graham scan

Suppose you have C.H.($P_1...P_n$)
Incremental C.H.

Could also sort and add points in order

0) Suppose you have C.H.\((P_1...P_n)\)
1) Start with \(P_n\) \(P_{n+1}\)
2) Pop vertices

\(\Rightarrow\) as w/ Graham scan

\(O(n)\) pops per increment

but also in total:

\(\text{TIME} = \text{SORT} + O(n)\)
C.H. by DIVIDE & CONQUER

Goal: $O(n \log n) = T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Heart of problem: how to merge two hulls in $O(n)$ time

Ugly

better?
Find upper tangent/bridge

Only upper hull points are candidates
find point-hull tangent
Alternate sides
4. Find point-hull tangent
Alternate sides
by finding point-hull tangent
In each iteration, how do we find point-hull tangent?

But this could advance **discard** only one point:

\[ T_n = T_{n/2} + n \log n \]

BAD
Alternate sides
L find point-hull tangent

Just walk up.
"Linear" time per alternation

Total \( O(n) \)
"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

Upper hull only
"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

Upper hull only

divide-conquer-merge
divide-merge-conquer
"ULTIMATE PLANAR C.H. ALGORITHM?"

KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-merge-conquer

Upper hull only
Goal: Get $O(n \log h)$

we can't sort anything

how do we split? Median $O(n)$

we can only afford $O(n)$ to merge
i.e. to find a bridge

this ignores $h$
but if $h = n$
we'd be in trouble

$T(n) = 2T\left(\frac{n}{2}\right) + T_{\text{bridge}}(n)$
Finding a Bridge in Linear Time

Let the bridge have slope $k^*$. Suppose we guess slope $k$. Sweep $k$.

Guess $K < k^*$:
- Sweep stops on blue

Guess $K > k^*$:
- Sweep stops on red

Guess $K = k^*$:
- Confirm bridge

$O(n)$ time to guess & verify
- Arbitrarily pair up points
- Find median slope
- Arbitrarily pair up points
- Find median slope
- Guess $K = \text{median}$
Case 1

$k > k^*$

Half of the pairs have slope $k' > k$, so $k' > k^*$
Case 1: $k > k^*$

Half of the pairs have slope $k' > k$, so $k' > k^*$

$k^*$ can't shift below $b$, so $a$ can't be on bridge (it could be on C.H.)
Case 2

$k < k^*$

Half of the pairs have slope $k^* < k$, so $k' < k^*$

$k^*$ can’t shift below $a$, $b$ can’t be on bridge (it could be on C.H.)
THROW AWAY ONE POINT (a or b) FROM HALF THE PAIRS

Case 1

$K > K^*$

Half of the pairs have slope $K' > K$, so $K' > K^*$

$K < K^*$

Case 2

Half of the pairs have slope $K' < K$, so $K' < K^*$

a can't be on bridge (it could be on C.H.)

b can't be on bridge (it could be on C.H.)
If we guess wrong:

- THROW AWAY ONE POINT \((a \text{ or } b)\)
- FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again (with all original pts)

Time: \(c \cdot n\) for first wrong guess

- \(c \cdot \frac{3n}{4}\) for second ""
- \(c \cdot \frac{3}{4} \cdot \frac{3n}{4}\) for third.

\[\text{etc}\]

\[\text{total: } O(n)\]
My client couldn't have killed anyone with this arrow, and I can prove it!

I'd like to examine your proof, Zeno. You may approach the bench.

But never reach it!
"Prune & Search"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

\[ T(n) = F(n) + T\left(\frac{n}{c}\right) \quad [c > 1] \]

Ok I got myself confused in class here.
Here's a way to describe this.

Say you had to process the leaves of a full binary tree of depth \( n \), to find *something*.
If you fail, you somehow get to throw out half the leaves and try again. Thus \( 2^n \) for the first try, \( 2^{n-1} \) for the 2nd, etc.

\[ O(\log n) \quad O(1) \quad : \quad \text{binary search} \quad [c=2] \]

\[ O(n) \quad O(n) \quad : \quad \text{finding a bridge} \quad [c=\frac{4}{3}] \]

\[ O(n^k) \quad O(n^k) \quad \frac{n^k}{2^k} + \frac{n^k}{4^k} + \ldots + \frac{n^k}{2^i} \quad [c=2] \]

\[ O(2^n) \quad O(2^n) \quad 2^n + 2^{n-1} + 2^{n-2} + \ldots + 2 \quad [c=2] \]
We know how to find a bridge in linear time.

Might as well throw out potential non-C.H. pts inside ... it's "free".

Of course we might not throw anything out.
We know how to find a bridge in linear time.

Solve 2 smaller problems with \( \frac{n}{2} \) half points each.

That still only gives us \( O(n \log n) \).

Do we have to find a bridge that "splits" the hull evenly?

If we at least find one new bridge on both sides then we get \( O(\log h) \) depth.

If we don't find a bridge on one side, we must have thrown out \( \frac{n}{2} \) pts.
Cost tree

- First bridge: $c \cdot n$
- 2 more bridges: $c \cdot \frac{n}{2}$
- "Only" 3 bridges

Tree has depth $\min \{O(log n), O(h)\}$ and must have exactly $h$ nodes.

Swap nodes: $A < c \cdot \frac{n}{4}$ (and recursively the same)
Every node ascends only. Weight per level: less than full tree case.
We get a full tree: depth $\log h$.

$O(n \cdot \log h)$

See web notes for analysis.