"ULTIMATE PLANAR C.H. ALGORITHM?"

KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm.

For comp260, this is just for context.
"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

divide-conquer-merge

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

It's a divide & conquer algorithm

divide-conquer-merge
divide-merge-conquer!

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

It's a divide & conquer algorithm

divide-merge-conquer

Upper hull only
comp260: these points are colored depending on what side of the ray they are on. We just care that there is some ray. In the context of convex hulls, the ray is the median of the point set, in a particular direction that depends on what has happened before in the convex hull algorithm that uses this bridge-finding procedure.
**Finding a Bridge in Linear Time**

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

\[ \{ \text{Guess } k = k^* \} \quad \Rightarrow \quad \text{confirm bridge} \]
**Finding a Bridge in Linear Time**

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

\[ \text{Guess } k < k^* \quad \rightarrow \text{sweep stops on blue} \]

\[ \{ \text{Guess } k = k^* \quad \rightarrow \text{confirm bridge} \} \]
Finding a bridge in linear time

Let the bridge have slope $k^*$.

Suppose we guess slope $k$.

Sweep $k$.

Guess $k < k^*$

\rightarrow sweep stops on blue.

\{ Guess $k > k^*$

\rightarrow sweep stops on red.

\{ Guess $k = k^*$

\rightarrow confirm bridge.
Finding a Bridge in Linear Time

Let the bridge have slope \( K^* \)

Suppose we guess slope \( K \).

Sweep \( K \)

\[
\begin{align*}
\text{Guess } K < K^* \\
\rightarrow & \text{ sweep stops on blue}
\end{align*}
\]

\[
\begin{align*}
\text{Guess } K > K^* \\
\rightarrow & \text{ sweep stops on red}
\end{align*}
\]

\[\{ \text{Guess } K = K^* \]
\[\rightarrow \text{ confirm bridge} \]

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

Half of the pairs have slope \( K' > K \), so \( K' > K^* \).
Case 1

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

$k^*$ can't sweep below $b$, so $@$ 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 sweep below $a$

$K^*$ can't be on bridge (it could be on C.H.)

$a_x < b_x$
Example of linear-time bridge finding
Randomly pair points
Find median slope
Test slope:
- too steep
- only left side is extremal
Because slope is too steep:
Discard left endpoints of steeper pairs
Subset: \( \leq \frac{3}{4} \) original
Random pairs
Median slope
Discard left endpoints of steeper pairs.
New subset
$\leq \frac{3}{4} \cdot \frac{3}{4} \cdot \text{original}$
New random pairs and median slope
Too steep yet again
Discard...
Fourth attempt...
on $\leq \frac{3}{4} \cdot \frac{3}{4} \cdot \frac{3}{4}$ original
This time: Too shallow
Discard right endpoints of shallower pairs.
Find median slope.
Extreme finds one point on each side.
DONE
Case 1

\[ K > K^* \]

Half of the pairs have slope \( K' > K \), so
\[ K' > K^* \]

@ can’t be on bridge (it could be on C.H.)

Case 2

\[ K < K^* \]

Half of the pairs have slope \( K' < K \), so
\[ K' < K^* \]

6 can’t be on bridge (it could be on C.H.)
Case 1: \( K > K^* \)

- Half of the pairs have slope \( K' > K \), so \( K' > K^* \)

- Can't be on bridge (it could be on C.H.)

Case 2: \( K < K^* \)

- Half of the pairs have slope \( K' < K \), so \( K' < K^* \)

- Can't be on bridge (it could be on C.H.)

Throw away one point (a or b) from half the pairs
If we guess wrong:

THROW AWAY
ONE POINT (a or b)
FROM HALF THE PAIRS
If we guess wrong:

THROW AWAY
ONE POINT (a or b)
FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again.
If we guess wrong:

THROW AWAY ONE POINT \((a \lor b)\)
FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

Time: \(c \cdot n\) for first wrong guess
\[
\frac{3n}{4}
\]
for second ""
\[
\frac{3}{4} \cdot \frac{3n}{4}
\]
for third.
If we guess wrong:

THROW AWAY
ONE POINT \((a \lor b)\)
FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

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.

\(\text{etc.}\)

\(\text{total: } O(n)\)
"Prune & Search"

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

\[ T(n) = F(n) + T\left(\frac{n}{c}\right) \quad [c > 1] \]
“Prune & Search”

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

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

\[ \Downarrow \]

\[ O(1) \quad [c=2] \]
“Prune & Search”

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

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

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

\[ O(1) \quad O\left(\frac{4}{3}\right) \quad [c=\frac{4}{3}] \]
"Prune & Search"

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

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

\[\downarrow\]

$O(\log n)$ $O(1)$ : Binary search $[c=2]$

$O(n)$ $O(n)$ : Finding a bridge $[c=\frac{4}{3}]$
“Prune & Search”

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

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

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

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

\[ O(n^k) \quad O(n^k) \quad n^k + \frac{n^k}{2^k} + \frac{n^k}{4^k} + \ldots + \frac{n^k}{2^{ik}} \quad [c=2] \]
"Prune & Search"

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

\[ T(n) = F(n) + T(n/c) \quad [c > 1] \]

- \( O(\log n) \) \( O(1) \): Binary search \([c=2]\)
- \( O(n) \) \( O(n) \): Finding a bridge \([c=\frac{4}{3}]\)
- \( O(n^k) \) \( O(n^k) \): \( n + \frac{n^k}{2^k} + \frac{n^k}{4^k} + \cdots + \frac{n^k}{2^{ik}} \) \([c=2]\)
- \( O(2^n) \) \( O(2^n) \): \( 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2 \) \([c=2]\)

Search leaves, if "fail" then search parents etc.
We know how to find a bridge in linear time.

Take comp163!
We know how to find a bridge in linear time.

Might as well throw out potential non-C.H. pts inside... it's "free"
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 \( n \) half points each.

That still only gives us \( O(n \log n) \).
Do we have to find a bridge that “splits” the hull evenly?
We know how to find a bridge in linear time.

Solve 2 smaller problems with \( n \) 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.
We know how to find a bridge in linear time.

Solve 2 smaller problems with \( n/2 \) 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 \( n/2 \) pts.