Computational Geometry

Winter term 2014/15

Lecture: Alexander Wolff (E29)
Tutorial: Philipp Kindermann (E12)

Convex Hull
or
Mixing Things

Lecture #1
Alexander Wolff

Email:
vorname.nachname @uni-wuerzburg.de

Office hours:
Wed, 13–14h

Office:
E29 @ Former Math Build.

Research areas:
- Graph Drawing
- Comp. Geometry
- Alg. for GIS
- Alg. Graph Theory
Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...
Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...

– decide which algorithms help to solve a number of fundamental geometric problems,
Computational Geometry

Learning goals: At the end of this lecture you will be able to...

– decide which algorithms help to solve a number of fundamental geometric problems,

– analyze new problems and find – with the concepts of the lecture – efficient solutions.
Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...

- decide which algorithms help to solve a number of fundamental *geometric* problems,

- analyze new problems and find – with the concepts of the lecture – *efficient* solutions.
Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...

– decide which algorithms help to solve a number of fundamental geometric problems,

– analyze new problems and find – with the concepts of the lecture – efficient solutions.

**Requirements:**
Computational Geometry

Learning goals: At the end of this lecture you will be able to...

– decide which algorithms help to solve a number of fundamental geometric problems,

– analyze new problems and find – with the concepts of the lecture – efficient solutions.

Requirements: – Algorithms & Data Structures
– Algorithmic Graph Theory
Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...

- decide which algorithms help to solve a number of fundamental geometric problems,
- analyze new problems and find – with the concepts of the lecture – efficient solutions.

**Requirements:**
- Algorithms & Data Structures
- Algorithmic Graph Theory
- 50% of the exercise points
- oral exam (15–20’)

Computational Geometry

**Learning goals:** At the end of this lecture you will be able to...

- decide which algorithms help to solve a number of fundamental *geometric* problems,
- analyze new problems and find – with the concepts of the lecture – *efficient* solutions.

**Requirements:**
- Algorithms & Data Structures
- Algorithmic Graph Theory
- 50% of the exercise points
- oral exam (15–20’)

**My vision:**
Computational Geometry

Learning goals: At the end of this lecture you will be able to...

– decide which algorithms help to solve a number of fundamental geometric problems,

– analyze new problems and find – with the concepts of the lecture – efficient solutions.

Requirements:
– Algorithms & Data Structures
– Algorithmic Graph Theory
– 50% of the exercise points
– oral exam (15–20’)

My vision:
• “hands-on”
• interactive
Literature


Achtung: Werbung!

Andere Master-Veranstaltungen des LS I in diesem Semester:

- Vorlesung: Approximationsalgorithmen
  Joachim Spoerhase, VL: Mi 8:30–10:00, ÜR I

- Seminar: Visualisierung von geografischen Netzwerken
  AW & Thomas van Dijk: Di 14–16, SE 36
Chapter 1

Mixing Things

or:

Convex Hull
Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>frac. $A$</th>
<th>frac. $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. A</th>
<th>fract. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

can we mix

| $q_1$  | 25 %    | 28 %    |
| $q_2$  | 15 %    | 18 %    |

using $s_1, s_2, s_3$?
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. A</th>
<th>fract. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

can we mix

<table>
<thead>
<tr>
<th></th>
<th>fract. A</th>
<th>fract. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>25 %</td>
<td>28 %</td>
</tr>
<tr>
<td>$q_2$</td>
<td>15 %</td>
<td>18 %</td>
</tr>
</tbody>
</table>

using $s_1$, $s_2$, $s_3$ ?
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. $A$</th>
<th>fract. $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10%</td>
<td>35%</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Can we mix $q_1$ using $s_1$, $s_2$, $s_3$?
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. $A$</th>
<th>fract. $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

can we mix $q_1$ and $q_2$ using $s_1$, $s_2$, $s_3$?
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. A</th>
<th>fract. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>s₂</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>s₃</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

can we mix

| q₁     | 25 %     | 28 %     |
| q₂     | 15 %     | 18 %     |

using s₁, s₂, s₃ ?

Observe: Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in $S$ $\iff$
Given... 

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. $A$</th>
<th>fract. $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10%</td>
<td>35%</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Can we mix $q_1$, $q_2$ using $s_1$, $s_2$, $s_3$?

**Observe:** Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in $S$ $\iff$ $q \in \text{CH}(S)$. 
Mixing Things

Given...

<table>
<thead>
<tr>
<th>subst.</th>
<th>fract. A</th>
<th>fract. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10 %</td>
<td>35 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>20 %</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>40 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>

Can we mix using $s_1$, $s_2$, $s_3$?

Observe: Given a set $S \subset \mathbb{R}^d$ of substances, we can mix a substance $q \in \mathbb{R}^d$ using the substances in $S$ $\iff q \in \text{CH}(S)$. 
Formally...
Given $S \subseteq \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:
Formally...

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach: – take (large enough) elastic rope
Formally...

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:
- take (large enough) elastic rope
- stretch and let go
Formally...

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:
- take (large enough) elastic rope
- stretch and let go
Formally...  

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?  

Physics approach:  
- take (large enough) elastic rope  
- stretch and let go  
- take area inside (and on) the rope
Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:  
- take (large enough) elastic rope
- stretch and let go
- take area inside (and on) the rope
Formally...

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:  
- take (large enough) elastic **rope**
- stretch and let go
- take area inside (and on) the rope

Math approach:
Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:  
– take (large enough) elastic rope  
– stretch and let go  
– take area inside (and on) the rope

Math approach:  
– define convex
Formally...

Given $S \subset \mathbb{R}^2$, how do we define the convex hull $\text{CH}(S)$?

Physics approach:  
- take (large enough) elastic rope
- stretch and let go
- take area inside (and on) the rope

Math approach:  
- define convex
- define $\text{CH}(S) = \bigcap_{C \supseteq S : C \text{ convex}} C$
Towards Computation

\[ CH(S) = \text{def } \bigcap_{C \supseteq S: C \text{ convex}} C \]

Problem with math approach:
Towards Computation

\[ \text{CH}(S) =_{\text{def}} \bigcap C \supseteq S : C \text{ convex} \]

Problem with math approach: \textit{This set is HUGE!}
Towards Computation

\[ \text{CH}(S) =_{\text{def}} \bigcap C \supseteq S: C \text{ convex} \]

Problem with math approach: \textit{This set is huge!}

Maybe we can do with a little less?
Towards Computation

$$\text{CH}(S) \overset{\text{def}}{=} \bigcup C$$

This set is HUGE!

Problem with math approach:

Maybe we can do with a little less?

Claim: $$\text{CH}(S) =$$
Towards Computation

$$\text{CH}(S) = \text{def } \bigcap C$$

$C \supseteq S$: $C$ convex

Problem with math approach: *This set is HUGE!*

Maybe we can do with a little less?

**Claim:**

$$\text{CH}(S) = \bigcap H$$

$H \supseteq S$: $H$ closed halfplane
Towards Computation

$$\text{CH}(S) = \text{def} \bigcap C$$

**Problem with math approach:** This set is HUGE!

Maybe we can do with a little less?

**Claim:**

$$\text{CH}(S) = \bigcap \bigcup H = \bigcup_{H \supseteq S} H$$

$H$ closed halfplane
Towards Computation

Problem with math approach: This set is HUGE!

Maybe we can do with a little less?

Claim: \( \text{CH}(S) = \bigcap H \) where:
- \( H \supseteq S \) and
  - \( H \supseteq S \): \( H \) closed halfplane
  - \( |\partial H \cap S| \geq 2 \)
- \( H \supseteq S \): \( H \) cl. halfplane
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$
Input: set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$

**Output:** list of vertices of $\text{CH}(S)$ in clockwise order
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$

**Output:** list of vertices of $\text{CH}(S)$ in clockwise order

**Observation.** $(p, q)$ is an edge of $\text{CH}(S)$ \iff \ldots
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subset \mathbb{R}^2$

**Output:** list of vertices of $\text{CH}(S)$ in clockwise order

**Observation.** $(p, q)$ is an edge of $\text{CH}(S)$ $\iff$
- each point in $S$ lies
  - strictly to the right of the directed line $\overrightarrow{pq}$ or
  - on the line segment $\overline{pq}$
Finally, an Algorithm

\begin{algorithm}
\caption{FirstConvexHull\((S)\)}
\begin{algorithmic}
\State \( E \leftarrow \emptyset \)
\ForEach {\((p, q) \in S \times S \text{ with } p \neq q\)}
\State \( \text{valid} \leftarrow \text{true} \)
\ForEach {\(r \in S\)}
\If {not \((r \text{ strictly right of } pq \text{ or } r \in pq)\)}
\State \( \text{valid} \leftarrow \text{false} \)
\EndIf
\EndFor
\If {\text{valid}}
\State \( E \leftarrow E \cup \{(p, q)\} \)
\EndIf
\EndFor
\State \text{from } E \text{ construct sorted list } L \text{ of vertices of } \text{CH}(S)
\Return \(L\)
\end{algorithmic}
\end{algorithm}
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do} \quad \begin{cases} \\
\text{valid} \leftarrow \text{true} \\
\text{foreach } r \in S \text{ do} \\
\quad \text{if not } (r \text{ strictly right of } p\rightarrow q \text{ or } r \in p\overrightarrow{q}) \text{ then} \\
\quad \quad \quad \text{valid} \leftarrow \text{false} \\
\quad \text{if valid then} \\
\quad \quad E \leftarrow E \cup \{(p, q)\} \\
\end{cases} \\
\text{from } E \text{ construct sorted list } L \text{ of vertices of CH}(S) \]

\text{return } L
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\textbf{foreach} \((p, q) \in S \times S \text{ with } p \neq q \text{ do} \]  // test: \((p, q)\) edge of CH(S)?

valid \leftarrow \text{true}

\textbf{foreach}\( r \in S \text{ do} \]

\textbf{if} not \((r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq})\) then

valid \leftarrow \text{false}

\textbf{if} valid \text{ then}

\[ E \leftarrow E \cup \{(p, q)\} \]

from \(E\) construct sorted list \(L\) of vertices of CH(S)

return \(L\)
Finally, an Algorithm

FirstConvexHull($S$)

$E \leftarrow \emptyset$

foreach $(p, q) \in S \times S$ with $p \neq q$ do

valid $\leftarrow$ true

foreach $r \in S$ do

if not ($r$ strictly right of $\vec{pq}$ or $r \in \overline{pq}$) then

valid $\leftarrow$ false

if valid then

$E \leftarrow E \cup \{(p, q)\}$

from $E$ construct sorted list $L$ of vertices of CH($S$)

return $L$

$r$ strictly right of $\vec{pq}$

$\begin{array}{ccc}
    x_r & y_r & 1 \\
    x_p & y_p & 1 \\
    x_q & y_q & 1 \\
\end{array} < 0$
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do}
\]

\[
\text{valid } \leftarrow \text{true}
\]

\[
\text{foreach } r \in S \text{ do}
\]

\[
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then}
\]

\[
\text{valid } \leftarrow \text{false}
\]

\[
\text{if valid then}
\]

\[
E \leftarrow E \cup \{(p, q)\}
\]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[ \text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do} \]

\[ \text{valid } \leftarrow \text{true} \]

\[ \text{foreach } r \in S \text{ do} \]

\[ \text{if not (} r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq} \text{) then} \]

\[ \text{valid } \leftarrow \text{false} \]

\[ \text{if valid then} \]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do}
\]

\[ \text{valid } \leftarrow \text{true} \]

\[
\text{foreach } r \in S \text{ do}
\]

\[ \text{if not (r strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then} \]

\[ \text{valid } \leftarrow \text{false} \]

\[
\text{if valid then}
\]

\[ E \leftarrow E \cup \{(p, q)\} \]

\[
\text{from } E \text{ construct sorted list } L \text{ of vertices of } \text{CH}(S)
\]

return \( L \)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach} \ (p, q) \in S \times S \text{ with } p \neq q \text{ do}
\]

\[ \text{valid} \leftarrow \text{true} \]

\[
\text{foreach} \ r \in S \text{ do}
\]

\[ \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then} \]

\[ \text{valid} \leftarrow \text{false} \]

\[ \text{if valid then} \]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do }
\]

\[ \text{valid } \leftarrow \text{true} \]

\[
\text{foreach } r \in S \text{ do }
\]

\[ \text{if not (} r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq} \text{) then} \]

\[ \text{valid } \leftarrow \text{false} \]

\[
\text{if } \text{valid} \text{ then }
\]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

forall \((p, q) \in S \times S \) with \( p \neq q \) do

\[ valid \leftarrow true \]

forall \( r \in S \) do

\[ \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then} \]

\[ valid \leftarrow false \]

\[ \text{if } valid \text{ then} \]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)

---

```
r strictly right of \( \overrightarrow{pq} \)
\[ \begin{array}{ccc}
  x_r & y_r & 1 \\
  x_p & y_p & 1 \\
  x_q & y_q & 1 \\
\end{array} \]
< 0
```

Important:
Test takes \( O(1) \) time!
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[ \text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do} \]

\[ \text{valid } \leftarrow \text{true} \]

\[ \text{foreach } r \in S \text{ do} \]

\[ \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \text{ then} \]

\[ \text{valid } \leftarrow \text{false} \]

\[ \text{if valid then} \]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\textbf{foreach} \ (p, q) \in S \times S \text{ with } p \neq q \text{ do}

\quad \text{valid} \leftarrow \text{true}

\textbf{foreach} \ r \in S \text{ do}

\quad \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \text{ then}

\quad \quad \text{valid} \leftarrow \text{false}

\textbf{if valid then}

\quad E \leftarrow E \cup \{(p, q)\}

\text{from } E \text{ construct sorted list } L \text{ of vertices of } \text{CH}(S)

\text{return } L
Running Time Analysis

FirstConvexHull(S)

\[
E \leftarrow \emptyset
\]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do}
\]

\[
\text{valid } \leftarrow \text{true}
\]

\[
\text{foreach } r \in S \text{ do}
\]

\[
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then}
\]

\[
\text{valid } \leftarrow \text{false}
\]

\[
\text{if valid then}
\]

\[
E \leftarrow E \cup \{(p, q)\}
\]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\textbf{foreach} \((p, q) \in S \times S\) with \(p \neq q\) \textbf{do}

\quad valid \leftarrow \text{true}

\quad \textbf{foreach} \(r \in S\) \textbf{do}

\quad \quad \textbf{if not} \((r\) strictly right of \(\overrightarrow{pq}\) or \(r \in \overline{pq}\)) \textbf{then}

\quad \quad \quad valid \leftarrow \text{false}

\quad \textbf{if} valid \textbf{then}

\quad \quad \quad E \leftarrow E \cup \{(p, q)\}

from \(E\) construct sorted list \(L\) of vertices of \(\text{CH}(S)\)

\textbf{return} \(L\)
Running Time Analysis

FirstConvexHull(S)

\[
E \leftarrow \emptyset
\]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do}
\]

\[
\text{valid } \leftarrow \text{true}
\]

\[
\text{foreach } r \in S \text{ do}
\]

\[
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then}
\]

\[
\text{valid } \leftarrow \text{false}
\]

\[
\text{if valid then}
\]

\[
E \leftarrow E \cup \{(p, q)\}
\]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\textbf{foreach} \((p, q) \in S \times S\) with \(p \neq q\) \textbf{do}

\[ valid \leftarrow \text{true} \]

\textbf{foreach} \(r \in S\) \textbf{do}

\begin{align*}
&\text{if not (} r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq} \text{) then} \\
&\hspace{1em} valid \leftarrow \text{false}
\end{align*}

\textbf{if} \ valid \ then

\[ E \leftarrow E \cup \{(p, q)\} \]

from \(E\) construct sorted list \(L\) of vertices of \(\text{CH}(S)\)

return \(L\)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\textbf{foreach} \ (p, q) \in S \times S \text{ with } p \neq q \ 	extbf{do}
\]

\[ \text{valid} \leftarrow \text{true} \]

\[
\textbf{foreach} \ r \in S \ 	extbf{do}
\]

\[
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \ 	ext{then}
\]

\[ \text{valid} \leftarrow \text{false} \]

\[
\text{if valid then}
\]

\[ E \leftarrow E \cup \{(p, q)\} \]

from \( E \) construct sorted list \( L \) of vertices of \( \text{CH}(S) \)

return \( L \)

\[ \Theta(1) \]

\[ \Theta(n) \]

\[ \Theta(n^3) \]

\[ O(n^2) \]
Running Time Analysis

FirstConvexHull($S$)

\[ E \leftarrow \emptyset \]

\[ \text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do} \]

\[ \text{valid } \leftarrow \text{true} \]

\[ \text{foreach } r \in S \text{ do} \]

\[ \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \text{ then} \]

\[ \text{valid } \leftarrow \text{false} \]

\[ \text{if valid then} \]

\[ E \leftarrow E \cup \{(p, q)\} \]

from $E$ construct sorted list $L$ of vertices of $\text{CH}(S)$

return $L$

\[ \Theta(1) \Theta(n) \Theta(n^3) \]

\[ O(n^2 - n) \cdot \Theta(n^3) \]

Lemma. We can compute the convex hull of $n$ pts in the plane in $\Theta(n^3)$ time.
if not $(r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq})$ then

valid ← false
if not (r strictly right of \vec{pq} or r \in \overline{pq}) then
valid \leftarrow false
if not (\( r \) strictly right of \( \overrightarrow{pq} \) or \( r \in \overrightarrow{pq} \)) then

\[ \text{valid} \leftarrow \text{false} \]
Discussion

if not \((r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq})\) then

\[ \text{valid} \leftarrow \text{false} \]

Test may return wrong answer (floating pt arithmetic!):
\(r \text{ right of } \overrightarrow{pq} :-(\)
if not \((r\ \text{strictly right of } \vec{pq}\ \text{or } r \in \overline{pq})\) then 
\[
\text{valid } \leftarrow \text{false}
\]

Test may return wrong answer (floating pt arithmetic!):
\[
r \ \text{right of } \vec{pq} \ :-(
\]
Discussion

if not (r strictly right of \( \overrightarrow{pq} \) or \( r \in \overrightarrow{pq} \)) then

\[
\text{valid } \leftarrow \text{false}
\]
Discussion

\[
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then}
\]
\[
valid \leftarrow false
\]

Test may return wrong answer (floating pt arithmetic!):
\[
r \text{ right of } \overrightarrow{pq} :-(
\]

\[
p \text{ not right of } \overrightarrow{r q}
\]

\[
q \text{ not right of } \overrightarrow{pr}
\]
if not (r strictly right of \( \overrightarrow{pq} \) or \( r \in \overrightarrow{pq} \)) then
valid ← false

Test may return wrong answer
(floating pt arithmetic!):
\( r \) right of \( \overrightarrow{pq} \) :-(

\( p \) not right of \( \overrightarrow{rq} \)

\( q \) not right of \( \overrightarrow{pr} \)
Discussion

\[
\text{if not } \left( r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq} \right) \text{ then valid } \leftarrow \text{false}
\]

Test may return wrong answer (floating pt arithmetic!):
\[
r \text{ right of } \overrightarrow{pq} \quad :-(
\]

Observation. Algorithm FirstConvexHull is not robust.
New Ideas
New Ideas

- split computation in two
New Ideas

- split computation in two

upper convex hull
New Ideas

- split computation in two

upper convex hull

lower convex hull
New Ideas

- split computation in two
- bring pts in lexicographic order
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull($S$: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort $S$ lexicographically

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ to $n$ do

$L$.append($p_i$)

return $L$
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

\[
\text{UpperConvexHull}(S: \text{set of pts in the plane}) \\
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \\
L \leftarrow \langle p_1, p_2 \rangle \\
\text{for } i \leftarrow 3 \text{ to } n \text{ do} \\
\quad L.\text{append}(p_i) \\
\text{return } L
\]
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull(S: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically}
\]

\[
L \leftarrow \langle p_1, p_2 \rangle
\]

for \( i \leftarrow 3 \) to \( n \) do

\[
L.\text{append}(p_i)
\]

return \( L \)
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

**UpperConvexHull**(*S*: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically}
\]

\[
L \leftarrow \langle p_1, p_2 \rangle
\]

// compute upper convex hull of \{p_1, p_2, \ldots, p_i\}

\[
\text{for } i \leftarrow 3 \text{ to } n \text{ do}
\]

\[
L.\text{append}(p_i)
\]

\[
\text{return } L
\]
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically}
\]

\[
L \leftarrow \langle p_1, p_2 \rangle
\]

for $i \leftarrow 3 \text{ to } n$ do // compute upper convex hull of \{\(p_1, p_2, \ldots, p_i\)\}

\[
L.\text{append}(p_i)
\]

return $L$
New Ideas

- split computation in two
- bring pts in lexicographic order
- proceed incrementally

UpperConvexHull($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \\
L \leftarrow \langle p_1, p_2 \rangle \\
\text{for } i \leftarrow 3 \text{ to } n \text{ do} \quad \text{// compute upper convex hull of } \{p_1, p_2, \ldots, p_i\} \\
\quad L.\text{append}(p_i) \\
\quad \text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn} \text{ do} \\
\quad \quad \text{remove second last pt from } L \\
\text{return } L
\]
Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort $S$ lexicographically

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ to $n$ do

$L$.append($p_i$)

while $|L| > 2$ and last 3 pts in $L$ do not make right turn do

remove second last pt from $L$

return $L$
UpperConvexHull($S$: set of pts in the plane)\
\[\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically}\]
\[L \leftarrow \langle p_1, p_2 \rangle\]
\[\text{for } i \leftarrow 3 \text{ to } n \text{ do}\]
\[\quad L.\text{append}(p_i)\]
\[\quad \text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn do}\]
\[\quad \quad \text{remove second last pt from } L\]
\[\text{return } L\]
## Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

\[\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \]  
\[L \leftarrow \langle p_1, p_2 \rangle \]

for $i \leftarrow 3$ to $n$ do

\[L.\text{append}(p_i)\]

while $|L| > 2$ and last 3 pts in $L$ do not make right turn do

remove second last pt from $L$

return $L$

\[O(n \log n)\]

\[(n - 2) \cdot \]
Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \quad O(n \log n)
\]

\[
L \leftarrow \langle p_1, p_2 \rangle
\]

\[
\text{for } i \leftarrow 3 \text{ to } n \text{ do} \quad (n - 2) \cdot \quad \text{O} (n - 2)
\]

\[
L.\text{append}(p_i)
\]

\[
\text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn do}
\]

\[
\text{remove second last pt from } L
\]

return $L$
Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort $S$ lexicographically $\quad O(n \log n)$

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ to $n$ do $\quad (n - 2)\cdot$

$L$.append($p_i$)

while $|L| > 2$ and last 3 pts in $L$ do not make right turn do

remove second last pt from $L$

return $L$

Amortized analysis:
Running Time Analysis

**UpperConvexHull**($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \\
L \leftarrow \langle p_1, p_2 \rangle \\
\text{for } i \leftarrow 3 \text{ to } n \text{ do} \\
\quad L.\text{append}(p_i) \\
\quad \text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn do} \\
\quad \quad \text{remove second last pt from } L
\]

return $L$

**O**($n \log n$)

\((n - 2)\cdot ?\)

**Amortized analysis:**

– each pt $p_2, \ldots, p_{n-1}$ pays 1 $ for its potential removal later on
Running Time Analysis

**UpperConvexHull** ($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \\
L \leftarrow \langle p_1, p_2 \rangle \\
\text{for } i \leftarrow 3 \text{ to } n \text{ do} \\
\quad L.\text{append}(p_i) \\
\quad \text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn do} \\
\quad \quad \text{remove second last pt from } L \\
\text{return } L
\]

\[O(n \log n)\]

\[O((n - 2) \cdot \text{?)}\]

**Amortized analysis:**

- each pt $p_2, \ldots, p_{n-1}$ pays 1 $ for its potential removal later on
- this pays for the total effort of all executions of the while loop
Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

$\langle p_1, p_2, \ldots, p_n \rangle \leftarrow$ sort $S$ lexicographically

$L \leftarrow \langle p_1, p_2 \rangle$

for $i \leftarrow 3$ to $n$ do

$L$.append($p_i$)

while $|L| > 2$ and last 3 pts in $L$ do not make right turn do

remove second last pt from $L$

return $L$

$O(n \log n)$

$(n-2) \cdot ?$

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 $ for its potential removal later on

– this pays for the total effort of all executions of the while loop
Amortized analysis:

- each pt $p_2, \ldots, p_{n-1}$ pays 1 $ for its potential removal later on
- this pays for the total effort of all executions of the while loop
Running Time Analysis

UpperConvexHull($S$: set of pts in the plane)

\[
\langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically}
\]

\[
L \leftarrow \langle p_1, p_2 \rangle
\]

\[
\text{for } i \leftarrow 3 \text{ to } n \text{ do}
\]

\[
L.\text{append}(p_i)
\]

\[
\text{while } |L| > 2 \text{ and last 3 pts in } L \text{ do not make right turn } \text{ do}
\]

\[
\text{remove second last pt from } L
\]

return $L$

\[O(n)\]

\[O(n \log n)\]

\[(n - 2)\cdot ?\]

Amortized analysis:

– each pt $p_2, \ldots, p_{n-1}$ pays 1 $ for its potential removal later on

– this pays for the total effort of all executions of the while loop

Theorem. We can compute the convex hull of $n$ pts in the plane in $O(n \log n)$ time – in a robust way.
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

Jarvis’ gift-wrapping algorithm
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm

Runtime?
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm

Runtime? \( O(n \cdot h) \)
Jarvis’ gift-wrapping algorithm

Runtime? \( O(n \cdot h) \)

where \( h = |CH(S)| = \) size of the output
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm
  - Runtime? $O(n \cdot h)$

- Chan’s exponential search
  - $O(n \log h)$

\[ h = |\text{CH}(S)| = \text{size of the output} \]
Chan’s Algorithm
Chan’s Algorithm

**Algorithm** Hull2D($P$, $m$, $H$), where $P \subset \mathbb{R}^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition $P$ into subsets $P_1, \ldots, P_{[n/m]}$ each of size at most $m$
2. for $i = 1, \ldots, [n/m]$ do
3. compute $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of $P$
Chan’s Algorithm

**Algorithm** Hull2D\((P, m, H)\), where \(P \subset E^2\), \(3 \leq m \leq n\), and \(H \geq 1\)

1. partition \(P\) into subsets \(P_1, \ldots, P_{[n/m]}\) each of size at most \(m\)
2. for \(i = 1, \ldots, [n/m]\) do
3. compute conv\((P_i)\) by Graham’s scan and store its vertices in an array in ccw order
4. \(p_0 \leftarrow (0, -\infty)\)
5. \(p_1 \leftarrow\) the rightmost point of \(P\)
6. for \(k = 1, \ldots, H\) do
7. \(\text{for } i = 1, \ldots, [n/m] \text{ do}\)
8. \(\text{compute the point } q_i \in P_i \text{ that maximizes } \angle p_{k-1} p_k q_i \text{ (} q_i \neq p_k \text{) by performing a binary search on the vertices of conv}(P_i)\)
9. \(p_{k+1} \leftarrow\) the point \(q\) from \(\{q_1, \ldots, q_{[n/m]}\}\) that maximizes \(\angle p_{k-1} p_k q\)
10. if \(p_{k+1} = p_1\) then return the list \(\langle p_1, \ldots, p_k \rangle\)
11. return incomplete
Chan’s Algorithm

Algorithm Hull2D($P$, $m$, $H$), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition $P$ into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most $m$
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. \hspace{1em} compute conv($P_i$) by Graham’s scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of $P$
6. for $k = 1, \ldots, H$ do
7. \hspace{1em} for $i = 1, \ldots, \lceil n/m \rceil$ do
8. \hspace{2em} compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ ($q_i \neq p_k$) by performing a binary search on the vertices of conv($P_i$)
9. $p_{k+1} \leftarrow$ the point $q$ from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return incomplete
Chan’s Algorithm

**Algorithm** Hull2D($P$), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. \hspace{1em} $L \leftarrow$ Hull2D($P$, $m$, $H$), where $m = H = \min\{2^t, n\}$
3. \hspace{1em} if $L \neq$ incomplete then return $L$

**Algorithm** Hull2D($P$, $m$, $H$), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition $P$ into subsets $P_1, \ldots, P_{\lfloor n/m \rfloor}$ each of size at most $m$
2. for $i = 1, \ldots, \lfloor n/m \rfloor$ do
3. \hspace{1em} compute $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array
4. \hspace{1em} in ccw order
4. \hspace{1em} $p_0 \leftarrow (0, -\infty)$
5. \hspace{1em} $p_1 \leftarrow$ the rightmost point of $P$
6. for $k = 1, \ldots, H$ do
7. \hspace{1em} for $i = 1, \ldots, \lfloor n/m \rfloor$ do
8. \hspace{1em} compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ ($q_i \neq p_k$) by performing a binary search on the vertices of $\text{conv}(P_i)$
9. \hspace{1em} $p_{k+1} \leftarrow$ the point $q$ from $\{q_1, \ldots, q_{\lfloor n/m \rfloor}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return incomplete
Algorithm  [edit]

Initially, we assume that the value of \( h \) is known and make a parameter \( m = h \). This assumption is not realistic, but we remove it later. The algorithm starts by arbitrarily partitioning \( P \) into at most \( 1 + n/m \) subsets \( Q \) with at most \( m \) points each. Then, it computes the convex hull of each subset \( Q \) using an \( O(n \log n) \) algorithm. Note that, as there are \( O(n/m) \) subsets of \( O(m) \) points each, this phase takes \( O(n/m) O(m \log m) = O(n \log m) \) time.

The second phase consists of executing **Jarvis's march** and using the precomputed convex hulls to speed up the execution. At each step in Jarvis's march, we have a point \( p_i \) in the convex hull, and need to find a point \( p_{i+1} = f(p_i, P) \) such that all other points of \( P \) are to the right of the line \( p_i p_{i+1} \). If we know the convex hull of a set \( Q \) of \( m \) points, then we can compute \( f(p_i, Q) \) in \( O(\log m) \) time, by using binary search. We can compute \( f(p_i, Q) \) for all the \( O(n/m) \) subsets \( Q \) in \( O(n/m \log m) \) time. Then, we can determine \( f(p_i, P) \) using the same technique as normally used in Jarvis's march, but only considering the points that are \( f(p_i, Q) \) for some subset \( Q \). As Jarvis's march repeats this process \( O(h) \) times, the second phase also takes \( O(n \log m) \) time, and if \( m = h \), \( O(n \log h) \) time.

By running the two phases described above, we can compute the convex hull of \( n \) points in \( O(n \log h) \) time, assuming that we know the value of \( h \). If we make \( m < h \), we can abort the execution after \( m + 1 \) steps, therefore spending only \( O(n \log m) \) time (but not computing the convex hull). We can initially set \( m \) as a small constant (we use 2 for our analysis, but in practice numbers around 5 may work better), and increase the value of \( m \) until \( m > h \), in which case we obtain the convex hull as a result.

If we increase the value of \( m \) too slowly, we may need to repeat the steps mentioned before too many times, and the execution time will be large. On the other hand, if we increase the value of \( m \) too quickly, we risk making \( m \) much larger than \( h \), also increasing the execution time. Chan's algorithm squares the value of \( m \) at each iteration, and makes sure that \( m \) is never larger than \( n \). In other words, at iteration \( t \) (starting at 0), we have \( m = \min(n, 2^{2^t}) \). The total running time of the algorithm is

\[
\sum_{t=0}^{O(\log \log h)} O\left(n \log(2^{2^t})\right) = O(n) \sum_{t=0}^{O(\log \log h)} O(2^t) = O\left(n \cdot 2^{1+O(\log \log h)}\right) = O(n \log h).
\]

To generalize this construction for the 3-dimensional case, an \( O(n \log n) \) algorithm to compute the 3-dimensional convex hull should be used instead of Graham scan, and a 3-dimensional version of Jarvis's march needs to be used. The time complexity remains \( O(n \log h) \).
Algorithm  [edit]

Initially, we assume that the value of $h$ is known and make a parameter $m=h$. This assumption is not realistic, but we remove it later. The algorithm starts by arbitrarily partitioning $P$ into at most $1 + n/m$ subsets $Q$ with at most $m$ points each. Then, it computes the convex hull of each subset $Q$ using an $O(n \log n)$ algorithm. Note that, as there are $O(n/m)$ subsets of $O(m)$ points each, this phase takes $O(n/m)O(m \log m) = O(n \log m)$ time.

The second phase consists of executing *Jarvis's march* and using the precomputed convex hulls to speed up the execution. At each step in Jarvis's march, we have a point $p_i$ in the convex hull, and need to find a point $p_{i+1} = f(p_i, P)$ such that all other points of $P$ are to the right of the line $p_i p_{i+1}$. If we know the convex hull of a set $Q$ of $m$ points, then we can compute $f(p_i, Q)$ in $O(\log m)$ time, by using binary search. We can compute $f(p_i, Q)$ for all the $O(n/m)$ subsets $Q$ in $O(n/m \log m)$ time. Then, we can determine $f(p_i, P)$ using the same technique as normally used in Jarvis's march, but only considering the points that are $f(p_i, Q)$ for some subset $Q$. As Jarvis's march repeats this process $O(h)$ times, the second phase also takes $O(n \log m)$ time, and if $m=h$, $O(n \log h)$ time.

By running the two phases described above, we can compute the convex hull of $n$ points in $O(n \log h)$ time, assuming that we know the value of $h$. If we make $m<h$, we can abort the execution after $m+1$ steps, therefore spending only $O(n \log m)$ time (but not computing the convex hull). We can initially set $m$ as a small constant (we use 2 for our analysis, but in practice numbers around 5 may work better), and increase the value of $m$ until $m>h$, in which case we obtain the convex hull as a result.

If we increase the value of $m$ too slowly, we may need to repeat the steps mentioned before too many times, and the execution time will be large. On the other hand, if we increase the value of $m$ too quickly, we risk making $m$ much larger than $h$, also increasing the execution time. Chan's algorithm squares the value of $m$ at each iteration, and makes sure that $m$ is never larger than $n$. In other words, at iteration $t$ (starting at 0), we have $m = \min(n, 2^{2^t})$. The total running time of the algorithm is

$$
O(\log \log h) \sum_{t=0}^{O(\log \log h)} O(n \log(2^{2^t})) = O(n) \sum_{t=0}^{O(\log \log h)} O(2^{2^t}) = O(n) \cdot 2^{1+O(\log \log h)} = O(n \log h).
$$

To generalize this construction for the 3-dimensional case, an $O(n \log n)$ algorithm to compute the 3-dimensional convex hull should be used instead of Graham scan, and a 3-dimensional version of Jarvis's march needs to be used. The time complexity remains $O(n \log h)$. 
Algorithm [edit]

Initially, we assume that the value of \( h \) is known and make a parameter \( m = h \). This assumption is not realistic, but we remove it later. The algorithm starts by arbitrarily partitioning \( P \) into at most \( 1 + n/m \) subsets \( Q \) with at most \( m \) points each. Then, it computes the convex hull of each subset \( Q \) using an \( O(n \log n) \) algorithm. Note that, as there are \( O(n/m) \) subsets of \( O(m) \) points each, this phase takes \( O(n/m)O(m \log m) = O(n \log m) \) time.

The second phase consists of executing Jarvis's march and using the precomputed convex hulls to speed up the execution. At each step in Jarvis's march, we have a point \( p_i \) in the convex hull, and need to find a point \( p_{i+1} = f(p_i, P) \) such that all other points of \( P \) are to the right of the line \( p_i p_{i+1} \). If we know the convex hull of a set \( Q \) of \( m \) points, then we can compute \( f(p_i, Q) \) in \( O(\log m) \) time, by using binary search. We can compute \( f(p_i, Q) \) for all the \( O(n/m) \) subsets \( Q \) in \( O(n/m \log m) \) time. Then, we can determine \( f(p_i, P) \) using the same technique as normally used in Jarvis's march, but only considering the points that are \( f(p_i, Q) \) for some subset \( Q \). As Jarvis's march repeats this process \( O(h) \) times, the second phase also takes \( O(n \log m) \) time, and if \( m = h \), \( O(n \log h) \) time.

By running the two phases described above, we can compute the convex hull of \( n \) points in \( O(n \log h) \) time, assuming that we know the value of \( h \). If we make \( m < h \), we can abort the execution after \( m+1 \) steps, therefore spending only \( O(n \log m) \) time (but not computing the convex hull). We can initially set \( m \) as a small constant (we use 2 for our analysis, but in practice numbers around 5 may work better), and increase the value of \( m \) until \( m > h \), in which case we obtain the convex hull as a result.

Use \( \lceil \log \log h \rceil \leq 1 + \log \log h \) instead of \( O(\log \log h) \)!

If we increase the value of \( m \) too slowly, we may need to repeat the steps mentioned before too many times, and the execution time will be large. On the other hand, if we increase the value of \( m \) too quickly, we risk making \( m \) much larger than \( h \), also increasing the execution time. Chan's algorithm squares the value of \( m \) at each iteration, and makes sure that \( m \) is never larger than \( n \). In other words, at iteration \( t \) (starting at 0), we have \( m = \min(n, 2^t) \). The total running time of the algorithm is

\[
O(n) \sum_{t=0}^{O(\log \log h)} O(n \log(2^{2^t})) = \sum_{t=0}^{O(\log \log h)} O(2^t) = O(n \cdot 2^{1 + \log \log m}) = O(n \log m).
\]

To generalize this construction for the 3-dimensional case, an \( O(n \log n) \) algorithm to compute the 3-dimensional convex hull should be used instead of Graham scan, and a 3-dimensional version of Jarvis's march needs to be used. The time complexity remains \( O(n \log h) \).