Computational Geometry

Winter term 2016

Convex Hull
or
Mixing Things

Lecture #1

Prof. Dr. Alexander Wolff
Computational Geometry

Winter term 2016

Convex Hull
or
Mixing Things

Lecture #1

Prof. Dr. Alexander Wolff

Chair for Informatics I
Chair I

a) Efficient Algorithms

b) Knowledge-Based Systems
   Prof. Dietmar Seipel

c) Theoretical Comp. Science
   Prof. Christian Glaßer

d) Alg. for Large Networks
   Prof. Sabine Storandt
Chair I

a) Efficient Algorithms

Alexander Wolff  
Professor

Joachim Spoerhase  
PostDoc

Fabian Lipp

Benedikt Budig

Steven Chaplik  
PostDoc

Thomas van Dijk  
PostDoc

Dongliang Peng

Krzysztof Fleszar

b) Knowledge-Based Systems  
Prof. Dietmar Seipel

c) Theoretical Comp. Science  
Prof. Christian Glaßer

d) Alg. for Large Networks  
Prof. Sabine Storandt
Constellations

Joachim Spoerhase

Email: firstname.lastname@uni-wuerzburg.de
Constellations

Joachim Spoerhase
Email:  firstname.lastname
@uni-wuerzburg.de
Office hours: Wed, 14–15ʰ
Constellations

Joachim Spoerhase

Email: firstname.lastname@uni-wuerzburg.de

Office hours: Wed, 14–15h

Office: E33 @ Former Math Build. (M8)
Constellations

Joachim Spoerhase
Email: firstname.lastname@uni-wuerzburg.de
Office hours: Wed, 14–15h
Office: E33 @ Former Math Build. (M8)
Constellations

Joachim Spoerhase
Email: firstname.lastname@uni-wuerzburg.de
Office hours: Wed, 14–15h
Office: E33 @ Former Math Build. (M8)

Research areas:
- Approximation Alg.
- Graph Algorithms
- Location Problems
- Comp. Geometry
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.

**Requirements:**
- Big-Oh notation (Landau); e.g., \( O(n \log n) \)
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: – Big-Oh notation (Landau); e.g., $O(n \log n)$

– Some basic Algorithms & Data Structures

(Balanced) binary search tree, priority queue
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:  

– Big-Oh notation (Landau); e.g., $O(n \log n)$

– Some basic *Algorithms & Data Structures*
  
  (Balanced) binary search tree, priority queue

– Some basic *Algorithmic Graph Theory*

  Breadth-first search, Dijkstra’s algorithm
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:**

- Big-Oh notation (Landau); e.g., $O(n \log n)$
- Some basic *Algorithms & Data Structures* (Balanced) binary search tree, priority queue
- Some basic *Algorithmic Graph Theory* Breadth-first search, Dijkstra’s algorithm
- Oral exam at the end of the semester
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: • Big-Oh notation (Landau); e.g., \( O(n \log n) \)

– Some basic Algorithms & Data Structures
  (Balanced) binary search tree, priority queue

– Some basic Algorithmic Graph Theory
  Breadth-first search, Dijkstra’s algorithm

– Oral exam at the end of the semester

My vision: • “hands-on”

• interactive
Content

1. Convex Hull in 2D
2. Map Overlay
3. Polygon Triangulation
4. Linear Programming
5. Orthogonal Range Queries
6. Point Location
7. Voronoi Diagram
8. Delaunay Triangulation
9. Convex Hull in 3D
10. Motion Planning
11. Simplex Range Searching
12. Arrangements
13. Binary Space Partition
14. Randomized Triangulation
1. Convex Hull in 2D
2. Map Overlay
3. Polygon Triangulation
4. Linear Programming
5. Orthogonal Range Queries
6. Point Location
7. Voronoi Diagram
8. Delaunay Triangulation
9. Convex Hull in 3D
10. Motion Planning
11. Simplex Range Searching
12. Arrangements
13. Binary Space Partition
14. Randomized Triangulation
Literature

Literature


Rolf Klein:
Literature


Chapter 1

Convex Hull

or

Mixing Things
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>
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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**

| $q_1$  | 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 $q_1$ 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>(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 $q_1$, $q_2$ 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 $S$ ⇔
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>

Given a set $S \subset \mathbb{R}^2$ of substances, we can mix a substance $q \in \mathbb{R}^2$ using the substances in $S$ if and only if $q \in \text{CH}(S)$.

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 $S$ if and only if $q \in \text{CH}(S)$.
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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 $s_1$, $s_2$, $s_3$ ?

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

Given $S \subset \mathbb{R}^2$, 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 $CH(S)$?

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
Formally...

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

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

Given $S \subseteq \mathbb{R}^2$, 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

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

Problem with math approach:
Towards Computation

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

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

\[ \text{CH}(S) = \text{def} \bigcup C \]

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

Maybe we can do with a little less?
Towards Computation

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

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

Maybe we can do with a little less?

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

\[ \text{CH}(S) = \bigcap C \subseteq S: \text{C convex} \]

Problem with math approach: \( \text{This set is HUGE!} \)

Maybe we can do with a little less?

**Claim:** \( \text{CH}(S) = \bigcap H \subseteq S: \text{H closed halfplane} \)
Towards Computation

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

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

Maybe we can do with a little less?

\textbf{Claim:} \[ \text{CH}(S) = \bigcap_{H \supseteq S} H = H \subseteq S: \text{H closed halfplane} \]
Towards Computation

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

Problem with math approach: \text{This set is \textbf{HUGE}!}

Maybe we can do with a little less?

\textbf{Claim:} \quad \text{CH}(S) = \bigcap \left( \bigcup H \bigg| H \supseteq S: \begin{array}{c} H \text{ closed halfplane} \\ |\partial H \cap S| \geq 2 \end{array} \right) = \bigcap H \supseteq S: H \text{ cl. halfplane,}
Computer Science Approach

**Input:** set $S$ of $n$ points in the plane, that is, $S \subseteq \mathbb{R}^2$
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, $S \subset \mathbb{R}^2$
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 $\text{CH}(S)$ $\iff$
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 $\text{CH}(S)$ if and only if 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

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($S$)

$$E \leftarrow \emptyset$$

foreach $(p, q) \in S \times S$ with $p \neq q$ do  // test: $(p, q)$ edge of CH($S$)?

valid $\leftarrow$ true

foreach $r \in S$ do

if not $(r$ strictly right of $\overrightarrow{pq}$ 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 $L$
Finally, an Algorithm

FirstConvexHull\(S\)

\[ E \leftarrow \emptyset \]

\begin{align*}
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do} & \quad \text{// test: } (p, q) \text{ edge of } \text{CH}(S)\
\hspace{1cm} \text{valid } \leftarrow \text{true} \\
\hspace{1cm} \text{foreach } r \in S \text{ do} \\
\hspace{2cm} \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \text{ then} \\
\hspace{3cm} \text{valid } \leftarrow \text{false} \\
\hspace{1cm} \text{if } \text{valid} \text{ then} \\
\hspace{2cm} E \leftarrow E \cup \{(p, q)\}
\end{align*}

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

return \(L\)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

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

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

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

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

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

\textbf{if valid} \text{ then}

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

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

\text{return } L

\begin{align*}
\begin{array}{c|c|c}
\text{r strictly right of } \vec{pq} & x_r & y_r \\
\text{} & \geq & 1 \\
\text{r } \in \overrightarrow{pq} & x_p & y_p \\
\text{or } r \in \overrightarrow{pq} & x_q & y_q \\
\end{array} < 0
\end{align*}
Finally, an Algorithm

FirstConvexHull($S$)

$E \leftarrow \emptyset$

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

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

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

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

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

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

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

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

return $L$

\[\begin{array}{ccc}
  x_r & y_r & 1 \\
  x_p & y_p & 1 \\
  x_q & y_q & 1 \\
\end{array}\]

Important:
Test takes $O(1)$ time!
Finally, an Algorithm

FirstConvexHull($S$)

$$E \leftarrow \emptyset$$

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

$$\text{valid} \leftarrow \text{true}$$

foreach $r \in S$ do

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

$$\text{valid} \leftarrow \text{false}$$

if $\text{valid}$ then

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

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

return $L$

$$\begin{array}{c|c|c}
  x_r & y_r & 1 \\
  x_p & y_p & 1 \\
  x_q & y_q & 1 \\
\end{array} < 0$$

Important:
Test takes $O(1)$ time!
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

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

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

\textbf{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)

\[ E \leftarrow \emptyset \]

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

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

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

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

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

\textbf{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)

\[ E \leftarrow \emptyset \]

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

\[
\begin{align*}
\text{valid} & \leftarrow \text{true} \\
\textbf{foreach} \ r \in S & \text{ do} \\
\text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) & \text{ then} \\
\quad \text{valid} & \leftarrow \text{false} \\
\text{if } \text{valid} & \text{ then} \\
\quad \text{E} & \leftarrow \text{E} \cup \{(p, q)\}
\end{align*}
\]

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

return \(L\)
Finally, an Algorithm

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

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

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

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

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

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

\textbf{if} \ valid \textbf{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)

\[ E \leftarrow \emptyset \]
\[ \text{foreach} \ (p, q) \in S \times S \text{ with } p \neq q \ \text{do} \]
\[ \quad \text{valid} \leftarrow \text{true} \]
\[ \quad \text{foreach} \ r \in S \ \text{do} \]
\[ \quad \quad \text{if not} \ (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \ \text{then} \]
\[ \quad \quad \quad \text{valid} \leftarrow \text{false} \]
\[ \text{if valid then} \]
\[ \quad \quad E \leftarrow E \cup \{(p, q)\} \]

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}

\[
\text{valid} \leftarrow \text{true} \\
\textbf{foreach} r \in S \textbf{do} \\
\quad \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overline{pq}) \text{ then} \\
\quad\quad \text{valid} \leftarrow \text{false}
\]

\textbf{if} \ valid \textbf{then} \\
\quad \quad E \leftarrow E \cup \{(p, q)\}

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

\text{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} \]

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

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

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

\textbf{if} \ valid \text{ 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\((S)\)

\[
E \leftarrow \emptyset
\]

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

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

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

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

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

\[
\text{if valid then}
\]

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

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

return \(L\)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do } \\
\quad \text{valid } \leftarrow \text{true} \\
\quad \text{foreach } r \in S \text{ do } \\
\quad \\
\quad \quad \text{if not } (r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq}) \text{ then } \\
\quad \\
\quad \quad \quad \text{valid } \leftarrow \text{false} \\
\quad \]

\[
\text{if valid then } \\
\quad E \leftarrow E \cup \{(p, q)\}
\]

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

return \( L \)

\( \Theta(1) \)

\( \Theta(n^2 - n) \cdot \Theta(n) \)
Running Time Analysis

FirstConvexHull(S)

\[ E \leftarrow \emptyset \]

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

\hspace{1cm} \text{valid} \leftarrow \text{true}

\hspace{1cm} \textbf{foreach} \ r \in S \textbf{ do}

\hspace{2cm} \textbf{if not} (r \text{ strictly right of } pq \text{ or } r \in pq) \textbf{ then}

\hspace{3cm} \text{valid} \leftarrow \text{false}

\hspace{1cm} \textbf{if valid then}

\hspace{2cm} 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}

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

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

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

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

\textbf{if} \ \text{valid} \ \textbf{then}

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

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

\textbf{return} \(L\)

\[ \Theta(1) \]

\[ \Theta(n) \]

\[ \Theta(n^3) \]

\[ O(n^2) \]

\[ (n^2 - n). \]
Running Time Analysis

FirstConvexHull($S$)

$$E \leftarrow \emptyset$$

\[
\text{foreach } (p, q) \in S \times S \text{ with } p \neq q \text{ do } \\
\quad \text{valid } \leftarrow \text{true} \\
\quad \text{foreach } r \in S \text{ do } \\
\quad \quad \text{if not (} r \text{ strictly right of } \overrightarrow{pq} \text{ or } r \in \overrightarrow{pq} \text{) then } \\
\quad \quad \quad \text{valid } \leftarrow \text{false} \\
\quad \text{if valid then } \\
\quad \quad E \leftarrow E \cup \{(p, q)\} \\
\]

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

return $L$

\[\Theta(1) \quad \Theta(n) \quad \Theta(n^3) \quad O(n^2) \quad O(n^2 - n).\]

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

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

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

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

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

\[
\begin{align*}
\text{valid} & \leftarrow \text{false}
\end{align*}
\]
Discussion

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 $\overrightarrow{r\dot q}$
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

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

valid $\leftarrow$ false

Test may return wrong answer (floating pt arithmetic!):

$r$ right of $\overrightarrow{pq}$ :-(
Discussion

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 \( \overrightarrow{pq} \) :-(

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

- split computation in two
New Ideas

- split computation in two
New Ideas

- split computation in two
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

```python
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.append(p_i)

\text{return } L
```
New Ideas

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

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

return \( L \)
New Ideas

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

\[
\text{UpperConvexHull}(S: \text{set of pts in the plane})
\]

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

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

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

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

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

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

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

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

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

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

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

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

$L$.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 $L$
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 $L$
Running Time Analysis

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

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

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

\textbf{for} \( i \leftarrow 3 \) \textbf{to} \( n \) \textbf{do}

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

\textbf{while} \( |L| > 2 \) \textbf{and} last 3 pts in \( L \) do not make right turn \textbf{do}

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

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

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

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

$O(n \log n)$
Running Time Analysis

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

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

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

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

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

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

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

return $L$
Running Time Analysis

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

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

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

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

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

return $L$

Amortized analysis:
Running Time Analysis

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

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

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

\begin{verbatim}
for i ← 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 $L$
\end{verbatim}

return $L$

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)

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

$O(n \log n)$

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

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
Running Time Analysis

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

\[ \langle p_1, p_2, \ldots, p_n \rangle \leftarrow \text{sort } S \text{ lexicographically} \] \hspace{1cm} O(n \log n)

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

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

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

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

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

return $L$

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

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
Running Time Analysis

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

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

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

$O(n \log n)$

$(n - 2)$
Running Time Analysis

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

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

\[O(n \log n)\]

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
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
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm

Runtime?
Output-Sensitive Algorithms

- Jarvis' gift-wrapping algorithm

Runtime? $O(n \cdot h)$
Output-Sensitive Algorithms

- Jarvis’ gift-wrapping algorithm

Runtime? $O(n \cdot h)$

... where $h = |\text{CH}(S)| = \text{size of the output}$
Output-Sensitive Algorithms

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

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

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

Algorithm Hull2D($P$, $m$, $H$), where $P \subset \mathbb{E}^2$, $3 \leq m \leq n$, 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. compute $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
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 $P_1, \ldots, P_{\lfloor n/m \rfloor}$ each of size at most $m$
2. for $i = 1, \ldots, \lfloor n/m \rfloor$ do
3. compute $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
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 $P_1, \ldots, P_{\lfloor n/m \rfloor}$ each of size at most $m$
2. for $i = 1, \ldots, \lfloor n/m \rfloor$ do
3. compute $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array in ccw order [in $O(m \log m)$ time]
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 $\text{conv}(P_i)$ by Graham’s scan and store its vertices in an array
   in ccw order [in $O(m \log m)$ time]
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of $P$
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, [n/m]$ do
8. 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. $p_{k+1} \leftarrow$ 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 \(i = 1, \ldots, \lfloor n/m \rfloor\) do
   3. compute \(\text{conv}(P_i)\) by Graham’s scan and store its vertices in an array in ccw order \([\text{in } O(m \log m) \text{ time}]\)
4. \(p_0 \leftarrow (0, -\infty)\)
5. \(p_1 \leftarrow \text{the rightmost point of } P\)
6. for \(k = 1, \ldots, H\) do
7.    for \(i = 1, \ldots, \lfloor n/m \rfloor\) do
8.       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.    \(p_{k+1} \leftarrow \text{the point } q \text{ from } \{q_1, \ldots, q_{\lfloor n/m \rfloor}\} \text{ that maximizes } \angle p_{k-1} p_k q\)
10. if \(p_{k+1} = p_1\) then return the list \([p_1, \ldots, p_k]\)
11. return incomplete
Chan’s Algorithm

**Algorithm** Hull2D($P$), where $P \subseteq \mathbb{E}^2$

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

**Algorithm** Hull2D($P, m, H$), where $P \subseteq \mathbb{E}^2$, $3 \leq m \leq n$, 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. \hspace{1em} compute conv($P_i$) by Graham’s scan and store its vertices in an array in ccw order \hspace{1em} [in $O(m \log m)$ time]
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 $i = 1, \ldots, [n/m]$ do
8. \hspace{1em} \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 conv($P_i$)
9. \hspace{1em} $p_{k+1} \leftarrow$ the point $q$ from $\{q_1, \ldots, q_{[n/m]}\}$ that maximizes $\angle p_{k-1} p_k q$
10. \hspace{1em} 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 (for example, Graham scan). Note that, as there are $O(n/m)$ subsets of $O(m)$ points each, this phase takes $O(n/m) \cdot O(m \log m) = O(n \log m)$ time.

The second phase consists of executing Jarvis's march algorithm 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 therefore $O(n \log h)$ time if $m = h$.

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. Similar to strategy used by Chazelle and Matoušek's[4] algorithm, 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

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

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 $O(n \log h)$. 