Convex hull
Convex hull
Convex hull
Convex hull

- Definition, extremal point
- Jarvis algorithm
- Orientation predicate
- Buggy degenerate example
- Real RAM model and general position hypothesis
- Graham algorithm
- Lower bound
- Other results
- Higher dimensions
- Another lower bound
- A simple randomized algorithm for linear programming
Convex hull

Definition, extremal point
Convex hull
Set of points
Convex hull

Set of points

Smallest enclosing convex set
Convex hull

Set of points

Smallest enclosing convex set

Extremal point

Supporting line ("tangent" line)
Convex hull

lowest point is extremal

Jarvis algorithm
Convex hull

Jarvis algorithm

rotate
Convex hull

rotate

Jarvis algorithm
Convex hull

rotate

Jarvis algorithm
Convex hull
rotate

Jarvis algorithm
Convex hull

next vertex found
Convex hull

next vertex found and next one

Jarvis algorithm
Convex hull

next vertex found and next one

until back to starting point
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
  if $\text{angle}(ux, uw) < min$
    then $min = \text{angle}(ux, uw)$; $v = w$;
$u.next = v$;
Do
  $S = S \setminus \{v\}$
  $min = \infty$
For each $w \in S$
  if $\text{angle}(v.prev v, vw) < min$
    then $min = \text{angle}(v.prev v, vw)$; $v.next = w$;
  $v = v.next$;
While $v \neq u$

Jarvis algorithm
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
  if $\text{angle}(ux, uw) < min$
    then $min = \text{angle}(ux, uw); v = w;$
$u.\text{next} = v;$
Do
    $S = S \setminus \{v\}$
    $min = \infty$
  For each $w \in S$
    if $\text{angle}(v.\text{prev} v, vw) < min$
      then $min = \text{angle}(v.\text{prev} v, vw); v.\text{next} = w;$
    $v = v.\text{next};$
    While $v \neq u$

4 - 2
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$

For each $w \in S \setminus \{u\}$
if $\angle(ux, uw) < min$
then $min = \angle(ux, uw); v = w;$

$u.next = v;$

Do

$S = S \setminus \{v\}$
$min = \infty$

For each $w \in S$
if $\angle(v.prev v, vw) < min$
then $min = \angle(v.prev v, vw); v.next = w;$

$v = v.next;$

While $v \neq u$

Complexity? $O(n)$

Jarvis algorithm
Convex hull

Input: point set $S$

$u =$ lowest point in $S$; $\min = \infty$

For each $w \in S \setminus \{u\}$

\[
\text{if } \text{angle}(ux, uw) < \min \\
\text{then } \min = \text{angle}(ux, uw); \ v = w;
\]

$u.next = v;$

Do

\[
S = S \setminus \{v\} \\
\min = \infty
\]

For each $w \in S$

\[
\text{if } \text{angle}(v.prev v, vw) < \min \\
\text{then } \min = \text{angle}(v.prev v, vw); \ v.next = w;
\]

$v = v.next;$

While $v \neq u$

$O(n)$
Convex hull

Input : point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
  if $\text{angle}(ux, uw) < min$
    then $min = \text{angle}(ux, uw)$; $v = w$;

$u\.next = v$;

Do

$S = S \setminus \{v\}$
$min = \infty$
For each $w \in S$
  if $\text{angle}(v\.prev v, vw) < min$
    then $min = \text{angle}(v\.prev v, vw)$; $v\.next = w$;

$v = v\.next$;

While $v \neq u$

Jarvis algorithm

Complexity ?
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
    if $\text{angle}(ux, uw) < min$
        then $min = \text{angle}(ux, uw); v = w;$

Do

$S = S \setminus \{v\}$
$min = \infty$
For each $w \in S$
    if $\text{angle}(v.\text{prev v}, vw) < min$
        then $min = \text{angle}(v.\text{prev v}, vw); v.\text{next} = w;$
$v = v.\text{next};$

While $v \neq u$

$O(n)$
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
    if $\text{angle}(ux, uw) < min$
        then $min = \text{angle}(ux, uw)$; $v = w$
$u.next = v$
Do
    $S = S \setminus \{v\}$
    $min = \infty$
For each $w \in S$
    if $\text{angle}(v\text{.prev} v, vw) < min$
        then $min = \text{angle}(v\text{.prev} v, vw)$; $v.next = w$
$v = v.next$
While $v \neq u$

$O(n^2)$
Convex hull

Input : point set $S$
$u =$ lowest point in $S$; $\min = \infty$
For each $w \in S \setminus \{u\}$
    if $\text{angle}(ux, uw) < \min$
        then $\min = \text{angle}(ux, uw)$; $v = w$
$u.$next $= v$

Do

$S = S \setminus \{v\}$
$\min = \infty$
For each $w \in S$
    if $\text{angle}(v.$prev$ v, vw) < \min$
        then $\min = \text{angle}(v.$prev$ v, vw)$; $v.$next $= w$
$v = v.$next

While $v \neq u$

Jarvis algorithm

$O(n^2)$

$O(nh)$
Convex hull

Input: point set $S$
$u =$ lowest point in $S$; $min = \infty$
For each $w \in S \setminus \{u\}$
  if $\text{angle}(ux, uw) < min$
    then $min = \text{angle}(ux, uw); v = w;$
$u.next = v;$
Do
$S = S \setminus \{v\}$
For each $w \in S$
  $min = \infty$
  if $\text{angle}(v.prev v, vw) < min$
    then $min = \text{angle}(v.prev v, vw); v.next = w;$
$v = v.next;$
While $v \neq u$
Convex hull

if \( \text{angle}(pv, vw) < \text{min} \)
Convex hull

if $\text{angle}(pv, vw) < \text{min}$

$$\text{angle}(pv, vw) = \arccos\left(\frac{vw \cdot pv}{\|vw\| \cdot \|pv\|}\right)$$
Convex hull

if \( \text{angle}(pv, vw) < \text{min} \)

\[
\text{angle}(pv, vw) = \arccos \left( \frac{vw \cdot pv}{\|vw\| \cdot \|pv\|} \right)
\]
Convex hull

if $\text{angle}(pv, vw) < \text{min}$

$\text{angle}(pv, vw) = \arccos\left(\frac{\vec{vw} \cdot \vec{pv}}{\|\vec{vw}\| \cdot \|\vec{pv}\|}\right)$

if $vwn$ turn left

Orientation predicate
Convex hull

if $\text{angle}(pv, vw) < \text{min}$

$\text{angle}(pv, vw) = \arccos \left( \frac{vw \cdot pv}{\|vw\| \cdot \|pv\|} \right)$

if $vwn$ turn left

if triangle $vwn$ counterclockwise (ccw)

if triangle $vwn$ positively oriented

Orientation predicate
Convex hull

\[ vwn + ? \]
Convex hull

\[ vwn + ? \]

\[
\begin{vmatrix}
  x_w - x_v & x_n - x_v \\
  y_w - y_v & y_n - y_v \\
\end{vmatrix}
= \begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n \\
\end{vmatrix} > 0
\]
Convex hull

\[
\begin{vmatrix}
  x_w - x_v & x_n - x_v \\
  y_w - y_v & y_n - y_v
\end{vmatrix}
= \begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n
\end{vmatrix}
> 0
\]

\[
\begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n
\end{vmatrix}
< 0
\]
Convex hull

Orientation predicate

\[
\begin{vmatrix}
  x_w - x_v & x_n - x_v \\
  y_w - y_v & y_n - y_v \\
\end{vmatrix} = \begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n \\
\end{vmatrix} > 0
\]

\[
\begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n \\
\end{vmatrix} < 0
\]

\[
\begin{vmatrix}
  1 & 1 & 1 \\
  x_v & x_w & x_n \\
  y_v & y_w & y_n \\
\end{vmatrix} = 0
\]
Convex hull

Orientation predicate

\[ \text{vwn} + ? \]

\[ \begin{vmatrix} x_w - x_v & x_n - x_v \\ y_w - y_v & y_n - y_v \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ x_v & x_w & x_n \\ y_v & y_w & y_n \end{vmatrix} > 0 \]

\[ \text{vwn} - ? \]

\[ \begin{vmatrix} 1 & 1 & 1 \\ x_v & x_w & x_n \\ y_v & y_w & y_n \end{vmatrix} < 0 \]

\[ \text{vwn} 0 ? \]

\[ \begin{vmatrix} 1 & 1 & 1 \\ x_v & x_w & x_n \\ y_v & y_w & y_n \end{vmatrix} = 0 \]

degenerate case
**Convex hull**

\[
\begin{vmatrix}
    x_w - x_v & x_n - x_v \\
    y_w - y_v & y_n - y_v
\end{vmatrix} = \begin{vmatrix}
    1 & 1 & 1 \\
    x_v & x_w & x_n \\
    y_v & y_w & y_n
\end{vmatrix} > 0
\]

\[
\begin{vmatrix}
    1 & 1 & 1 \\
    x_v & x_w & x_n \\
    y_v & y_w & y_n
\end{vmatrix} < 0
\]

\[
\begin{vmatrix}
    1 & 1 & 1 \\
    x_v & x_w & x_n \\
    y_v & y_w & y_n
\end{vmatrix} = 0
\]
Convex hull

Rounding errors possible

\[ p = \left( \frac{1}{2} + x.u, \frac{1}{2} + y.u \right) \]

\[ 0 \leq x, y \leq 256, \ u = 2^{-53} \]

\[ q = (12, 12) \]

\[ r = (24, 24) \]

\[ \text{orientation}(p, q, r) \]

evaluated with double

Orientation predicate
Convex hull

Rounding errors possible

Orientation predicate

\[ p = \left( \frac{1}{2} + x.u, \frac{1}{2} + y.u \right) \]

\[ 0 \leq x, y \leq 256, \ u = 2^{-53} \]

\[ q = (12, 12) \]

\[ r = (24, 24) \]

\[ \text{orientation}(p, q, r) \]

evaluated with double

\[ \leq 0 \]

\[ 0 \]

\[ \geq 0 \]
Convex hull

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]
Convex hull

Buggy degenerate example
(single precision)

Input: point set \( S \)
\( u = v = \) lowest point in \( S \);

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]
Convex hull

Buggy degenerate example
(single precision)

\( w_1 = (12, 12) \)
\( w_2 = (24, 24) \)
\( w_3 = (30, 30.000001) \)
\( w_4 = (23, 36) \)
\( w_5 = (0.5000029, 0.5000027) \)

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
\[ \text{if } vwn \text{ positive} \]
\[ \text{then } n = w; \]
\[ v.next = n; v = n; \]
\[ S = S \setminus \{v\} \]
While \( v \neq u \)
Convex hull

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\( v.next = n; v = n; \)
\( S = S \setminus \{v\} \)
While \( v \neq u \)
Convex hull

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)

if \( vwn \) positive

then \( n = w; \)

\( v.next = n; \ v = n; \)

\( S = S \setminus \{v\} \)

While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\( v.next = n; v = n; \)
\( S = S \setminus \{v\} \)
While \( v \neq u \)
Convex hull

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)

\[ \text{if } vwn \text{ positive} \]
\[ \text{then } n = w; \]

\[ v.next = n; v = n; \]
\[ S = S \setminus \{v\} \]

While \( v \neq u \)

Teaser robustness lecture
Convex hull

Buggy degenerate example
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\( v.next = n; v = n; \)
\( S = S \setminus \{v\} \)
While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
\[ \text{if } vwn \text{ positive} \]
\[ \text{then } n = w; \]
\[ v.next = n; v = n; \]
\[ S = S \setminus \{v\} \]
While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\(w_1 = (12, 12)\)
\(w_2 = (24, 24)\)
\(w_3 = (30, 30.000001)\)
\(w_4 = (23, 36)\)
\(w_5 = (0.5000029, 0.5000027)\)

Do

\(n = \text{first in } S;\)
For each \(w \in S\)
if \(vwn\) positive
then \(n = w;\)
\(v.next = n;\ v = n;\)
\(S = S \setminus \{v\}\)
While \(v \neq u\)
**Convex hull**

![Diagram of convex hull with points u, v, w, n labeled.]

---

**Buggy degenerate example**

(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

---

**Do**

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\[ v.next = n; v = n; \]
\[ S = S \setminus \{v\} \]
While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)

\[ \text{if } vwn \text{ positive} \]
\[ \text{then } n = w; \]
\[ v.\text{next} = n; v = n; \]
\[ S = S \setminus \{v\} \]

While \( v \neq u \)

Teaser robustness lecture
Convex hull

**Buggy degenerate example**
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\( v.next = n; v = n; \)
\( S = S \setminus \{v\} \)
While \( v \neq u \)
Convex hull

Teaser robustness lecture

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
  \[ \text{if } vwn \text{ positive} \]
  \[ \text{then } n = w; \]
\[ v.n\text{ext} = n; \ v = n; \]
\[ S = S \setminus \{v\} \]
While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Do

\[ n = \text{first in } S; \]
For each \( w \in S \)
if \( vwn \) positive
then \( n = w; \)
\( v.next = n; v = n; \)
\( S = S \setminus \{v\} \)
While \( v \neq u \)
Convex hull

Buggy degenerate example
(single precision)

\[
\begin{align*}
\mathbf{w}_1 &= (12, 12) \\
\mathbf{w}_2 &= (24, 24) \\
\mathbf{w}_3 &= (30, 30.000001) \\
\mathbf{w}_4 &= (23, 36) \\
\mathbf{u} &= \mathbf{w}_5 = (0.5000029, 0.5000027)
\end{align*}
\]

Do

\[
\begin{align*}
n &= \text{first in } S; \\
\text{For each } w \in S \\
&\quad \text{if } v.w.n \text{ positive} \\
&\quad \quad \text{then } n = w; \\
&\quad v.next = n; \ v = n; \\
&\quad S = S \setminus \{v\}
\end{align*}
\]

While \( v \neq u \)
Convex hull

Buggy degenerate example (single precision)

\[ w_1 = (12, 12) \]
\[ w_2 = (24, 24) \]
\[ w_3 = (30, 30.000001) \]
\[ w_4 = (23, 36) \]
\[ w_5 = (0.5000029, 0.5000027) \]

Result is really wrong
Convex hull

Real Random Access Memory model

Assume exact computation on real numbers

constant time for single operations: $+$, $-$, $\sqrt{\cdot}$, $\sin \ldots$
Convex hull

Real RAM model and general position hypothesis

Real Random Access Memory model

Assume exact computation on real numbers

constant time for single operations: +, −, √, sin . . .

General position hypotheses

Predicate: Input \(\mapsto\) \{-1, 0, 1\}
Convex hull

Real Random Access Memory model

Assume exact computation on real numbers
constant time for single operations: $+$, $-$, $\sqrt{}$, $\sin$ . . .

General position hypotheses

Predicate: Input $\mapsto \{-1, 0, 1\}$

2D convex hull: no three points colinear
Convex hull

Real Random Access Memory model

Assume exact computation on real numbers

constant time for single operations: +, −, √, sin . . .

General position hypotheses

Predicate: Input ↦ {−1, 0, 1}

2D convex hull: no three points colinear
possibly: no 2 points with same x
Convex hull

Graham algorithm
Convex hull

sort around a point (e.g. lowest point)
Convex hull

sort around a point (e.g. lowest point)
Convex hull

Graham algorithm

leftturn

OK

10 - 4
Convex hull

Graham algorithm

rightturn
remove and go back

10 - 5
Convex hull

Graham algorithm

leftturn

OK
Convex hull

Graham algorithm

leftturn

OK
Convex hull

Graham algorithm

right turn

remove and go back
Convex hull

Graham algorithm

rightturn
remove and go back
Convex hull

Graham algorithm

leftturn

OK
Convex hull

Graham algorithm

rightturn

remove and go back
Convex hull

Graham algorithm

10 - 14

leftturn

OK
Convex hull

Graham algorithm

leftturn

OK
Convex hull

Graham algorithm
Convex hull

Graham algorithm

Input: point set $S$
u lowest point of $S$
sort $S$ around $u$ in a circular list including $u$
v = $u$

while $v.next \neq u$
  if $(v, v.next, v.next.next)$ ccw
    $v = v.next$
  else
    $v.next = v.next.next$
    $v.next.previous = v$
if $v \neq u$  $v = v.previous$
Convex hull

Input: point set $S$
$u$ lowest point of $S$;
sort $S$ around $u$ in a circular list including $u$;
$v = u$;

while $v.next \neq u$
    if $(v, v.next, v.next.next)$ ccw
        $v = v.next$;
    else
        $v.next = v.next.next$; $v.next.previous = v$;
if $v \neq u$ $v = v.previous$;

Complexity $O(n)$
Convex hull

Graham algorithm

Input: point set \( S \)

1. \( u \) lowest point of \( S \); 
2. sort \( S \) around \( u \) in a circular list including \( u \); 
3. \( v = u \); 
4. while \( v.next \neq u \) 
   if \((v, v.next, v.next.next)\) ccw 
   \( v = v.next; \) 
   else 
   \( v.next = v.next.next; v.next.previous = v; \) 
   if \( v \neq u \) \( v = v.previous; \)

Complexity: \( O(n \log n) \)
Convex hull

Input: point set $S$
u lowest point of $S$;
sort $S$ around $u$ in a circular list including $u$;
v = $u$;

while $v.next \neq u$
    if $(v, v.next, v.next.next)$ ccw
        $v = v.next$;
    else
        $v.next = v.next.next$; $v.next.previous = v$;
    if $v \neq u$ $v = v.previous$;
**Convex hull**

**Graham algorithm**

Input: point set $S$

- $u$ lowest point of $S$;
- sort $S$ around $u$ in a circular list including $u$;
- $v = u$;

\[
\textbf{while } v.next \neq u \\
\quad \text{if } (v, v.next, v.next.next) \text{ ccw} \\
\quad \quad v = v.next;
\]

else

\[
\text{v.next} = \text{v.next.next}; \quad \text{v.next.previous} = \text{v};
\]

\[
\text{if } v \neq u \quad v = v\.previous;
\]

**Complexity**

- at most $n$ times
- delete one point

\[
11 - 5
\]
Convex hull

Input: point set $S$
u lowest point of $S$;
sort $S$ around $u$ in a circular list including $u$;
v = $u$;

while $v_{\text{next}} \neq u$
  if $(v, v_{\text{next}}, v_{\text{next.next}})$ ccw
    $v = v_{\text{next}}$;
  else
    $v_{\text{next}} = v_{\text{next.next}}$; $v_{\text{next}.previous} = v$;
  if $v \neq u$ $v = v_{\text{previous}}$;

Complexity
at most $n$ times

delete one point
delete one point

Graham algorithm
distance to $u$ decreases
distance to $u$ decreases

at most $n$ times

at most $n$ times
Convex hull

Input: point set $S$
$u$ lowest point of $S$;
sort $S$ around $u$ in a circular list including $u$;
$v = u$;

while $v.next \neq u$
if $(v, v.next, v.next.next)$ ccw
$v = v.next;$
else
$v.next = v.next.next; v.next.previous = v;$
if $v \neq u$ $v = v.previous;$

Complexity
$O(n)$

Graham algorithm

delete one point
at most $n$ times

distance to $u$ decreases
at most $n$ times
Convex hull

Input: point set $S$
u lowest point of $S$;sort $S$ around $u$ in a circular list including $u$;$v = u$;

while $v.next \neq u$
  if $(v, v.next, v.next.next) \text{ ccw}$
    $v = v.next$;
  else
    $v.next = v.next.next$; $v.next.previous = v$;
  if $v \neq u$  $v = v.previous$;

Complexity

$O(n \log n)$
Convex hull

Problem lower bound is $\Omega(f(n))$

Iff there is NO algorithm

solving all size $n$ problems

using less than $Cf(n)$ operations

\[ \forall n \]

$C$ constant independent of $n$
Input: \( n \) real (positive) numbers
Input: \( n \) real (positive) numbers

Output: sorting permutation
Input: $n$ real (positive) numbers

Output: sorting permutation

Monitoring execution
Sorting

Input: $n$ real (positive) numbers

Output: sorting permutation

Monitoring execution

Yes  No
Input: $n$ real (positive) numbers

Output: sorting permutation

Monitoring execution
Sorting

Input: $n$ real (positive) numbers

Output: sorting permutation

Monitoring execution

Lower bound
Input: \( n \) real (positive) numbers

Output: sorting permutation

Monitoring execution

\[ \# \text{ leaves} \geq \# \text{ permutations} \]
Sorting

Input: \( n \) real (positive) numbers

Output: sorting permutation

Monitoring execution

\( \# \text{ leaves} \geq \# \text{ permutations} \)

There are \( n! \) permutations
Sorting

Input: \( n \) real (positive) numbers

Output: sorting permutation

Monitoring execution

\( \# \) leaves \( \geq \) \( \# \) permutations

There are \( n! \) permutations

Tree height is at least \( \log_2 \# \) leaves
Input: \( n \) real (positive) numbers

Output: sorting permutation

Monitoring execution

\# leaves \( \geq \) \# permutations

There are \( n! \) permutations

Tree height is at least \( \log_2 \# \) leaves

\# comparisons \( \leq \log_2 n! \approx n \log_2 n \)
Convex hull

Input: \( n \) 2D points (real coordinates)

Output: list of points along the convex hull
Convex hull

A stupid algorithm for sorting numbers
Convex hull

Lower bound

project on parabola
Convex hull

Lower bound

project on parabola
compute convex hull
Convex hull

Lower bound

project on parabola
compute convex hull
find lowest point
Convex hull

Lower bound

project on parabola
compute convex hull
find lowest point
enumerate $x$ coordinates in ccw CH order
Convex hull

- Project on parabola
- Compute convex hull
- Find lowest point
- Enumerate $x$ coordinates in ccw CH order

Lower bound on sorting:

$$f(n) + O(n) \geq \Omega(n \log n)$$
Convex hull

- \(O(n)\) project on parabola
- \(f(n)\) compute convex hull
- \(O(n)\) find lowest point
- \(O(n)\) enumerate \(x\) coordinates in ccw CH order

Lower bound on sorting:

\[ f(n) + O(n) \geq \Omega(n \log n) \]
Convex hull

Input: $n$ points in $\mathbb{R}^2$ (real coordinates)

Output: list of points along the convex hull
Convex hull

Input: \( n \) points in \( \mathbb{R}^2 \) (real coordinates)

Output: list of extreme points (not ordered)
Convex hull

Input: $n$ points in $\mathbb{R}^2$ (real coordinates)

Output: list of points along the convex hull

Output: list of extreme points (not ordered)

Weaker output: are all points extreme (strictly)
Convex hull

Input: \( n \) points in \( \mathbb{R}^2 \) (real coordinates)

Output: list of extreme points (not ordered)

Weaker output: are all points extreme (strictly)

i.e., split \( \mathbb{R}^{2n} \) in two parts

- “all points are extreme” part \( = S \)
- complementary part
Theorem [Ben-Or]: Any decision tree algorithm that solve the membership in a set $S$ problem has lower bound $\log_2 \#(S)$ where $\#(S)$ is the number of connected component of $S$. 
Convex hull

Another lower bound

Just prove that \( S' \) has enough connected components

i.e., split \( \mathbb{R}^{2n} \) in two parts

- “all points are extreme” part \( = S \)
- complementary part
Convex hull

Just prove that $S'$ has enough connected components

i.e., split $IR^{2n}$ in two parts

- “all points are extreme” part $= S$
- complementary part

$\begin{align*}
\text{p}_3 & \quad \text{p}_2 & \quad \text{p}_1 \\
\text{p}_3 & \quad \text{p}_2 & \quad \text{p}_1
\end{align*}$
Convex hull

Another lower bound

Just prove that $S'$ has enough connected components

i.e., split $IR^{2n}$ in two parts

- “all points are extreme” part $= S$
- complementary part
Convex hull

Another lower bound

Just prove that $S$ has enough connected components

i.e., split $\mathbb{R}^{2n}$ in two parts

- "all points are extreme" part $= S$
- complementary part

must cross $\partial S$
Convex hull

Another lower bound

Just prove that $S$ has enough connected components

i.e., split $\mathbb{IR}^{2n}$ in two parts

"all points are extreme" part $= S$

complementary part
Just prove that $S$ has enough connected components

One connected component per (non circular) permutation
Convex hull

Another lower bound

Just prove that $S$ has enough connected components

One connected component per (non circular) permutation

$$\Omega(\log(n - 1)!) = \Omega(n \log n)$$
Convex hull

Expected size of the convex hull
Convex hull

Expected size of the convex hull

$n$ random points in a square

$\Theta(\log n)$

Other results

expected
Convex hull

Expected size of the convex hull

\( n \) random points in a square

\( n \) random points in a disk

\[ \Theta(\log n) \]

\[ \Theta(n^{\frac{1}{3}}) \]

Other results

Expected size of the convex hull

\( n \) random points in a square

\( n \) random points in a disk

\[ \Theta(\log n) \]

\[ \Theta(n^{\frac{1}{3}}) \]
Convex hull

Expected size of the convex hull

- $n$ random points in a square
- $n$ random points in a disk
- $n$ random points, Gaussian distribution

Other results

- Expected size:
  - $\Theta(\log n)$
  - $\Theta(n^{\frac{1}{3}})$
  - $\Theta(\sqrt{\log n})$
Maximal points

In a set of points
Maximal points

In a set of points

\( p \) is NE maximal if empty
Maximal points

In a set of points

\( p \) is NE maximal if empty
Maximal points

In a set of points

\( p \) is NE maximal if empty
Maximal points

In a set of points

$p$ is NE maximal if empty
Maximal points contains extreme points

An extreme point is NE, NW, SW, or SE maximal
Maximal points contains extreme points

An extreme point is NE, NW, SW, or SE maximal

extreme point
Maximal points contains extreme points

An extreme point is NE, NW, SW, or SE maximal

NW maximal point

extreme point
Maximal points

$\ell$ random points in a square

NW maximal?
Maximal points

$n$ random points in a square

NW maximal?

Number by increasing $x$

$n$ random points in a square
Maximal points

Expected number?

$n$ random points in a square

NW maximal?

Number by increasing $x$

$p_i$ maximal

if $y_{p_i} > y_{p_j}$ for $j < i$
Maximal points

$n$ random points in a square

Number by increasing $x$

$\pi_i$ maximal

if $y_{\pi_i} > y_{\pi_j}$ for $j < i$

it happens with proba $\frac{1}{j}$

NW maximal?

expected number?
Maximal points

$n$ random points in a square

Number by increasing $x$

$p_i$ maximal

if $y_{p_i} > y_{p_j}$ for $j < i$

it happens with proba $\frac{1}{j}$

$\mathbb{E}(\#(NW\text{maxima})) = \sum \frac{1}{j} \simeq \log n$
Maximal points

Expected number?

$n$ random points in a square

Convex hull

$IE(\#(CH)) \leq IE(\#(maximas)) \simeq 4 \log n$

$IE(\#(NW\maxima)) = \sum \frac{1}{j} \simeq \log n$

$NW$ maximal?

Number by increasing $x$

$p_i$ maximal

if $y_{p_i} > y_{p_j}$ for $j < i$

it happens with proba $\frac{1}{j}$
Convex hull

Simple polygon

Other results
Convex hull

Simple polygon

Other results

$O(n)$
Convex hull

Divide and conquer
Convex hull

Divide and conquer
Convex hull

Other results

Divide and conquer
Convex hull

Divide and conquer

Other results
Convex hull

Divide and conquer

$O(n \log n)$
Convex hull

Mariage before conquest

Other results
Convex hull

Mariage before conquest

Other results
Convex hull

Mariage before conquest
Convex hull

Mariage before conquest
Convex hull

Mariage before conquest

$O(n \log h)$

22 - 5
Convex hull

Quickhull

Other results
Convex hull

Quickhull

leftmost rightmost
Convex hull

Quickhull

leftmost rightmost farthest points

Other results

23 - 3
Convex hull

Quickhull

leftmost rightmost farthest points
Convex hull

Quickhull

Other results

leftmost rightmost farthest points
Convex hull

Quickhull

leftmost rightmost farthest points

$O(n \log n)$ expected

$O(n^2)$ worst-case

Other results
Convex hull

Dynamic maintenance

Other results
Convex hull

Dynamic maintenance

Other results
Convex hull

Dynamic maintenance

Other results
Convex hull

Dynamic maintenance

Other results
Convex hull

Dynamic maintenance

Other results
Convex hull

Dynamic maintenance

$O(\log n)$ per insertion

Other results
Convex hull

Euler relation

Polytope boundary

Three dimensions

Vertices

Edges

Faces
Convex hull

Euler relation

Polytope boundary

Three dimensions

Vertices | Edges | Faces
---|---|---
8 | 12 | 6

= 2

25 - 2
Convex hull

Euler relation

Polytope boundary

Three dimensions

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8$</td>
<td>$12$</td>
<td>$6$</td>
</tr>
<tr>
<td>$4$</td>
<td>$6$</td>
<td>$4$</td>
</tr>
</tbody>
</table>

$8 - 12 + 6 = 2$

$4 - 6 + 4 = 2$
Convex hull

Euler relation

Polytope boundary

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

= 2

= 2

= +0
Convex hull

Euler relation

Polytope boundary

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
<td>+0</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

8 − 12 + 6 = 2
4 − 6 + 4 = 2
+1− +1 + 0 = +0
0 − +1 + +1 = +0
Convex hull

Euler relation

Polytope boundary

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>+ 6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>+ 4</td>
</tr>
<tr>
<td>+1</td>
<td>+1</td>
<td>+ 0</td>
</tr>
<tr>
<td>0</td>
<td>+1</td>
<td>+ 1</td>
</tr>
</tbody>
</table>

= 2

= 2

= +0

= +0

25 - 6
Convex hull

Euler relation

Polytope boundary

Three dimensions

Vertices $n$ - Edges $e$ + Faces $f$ = 2
Convex hull

Euler relation

Polytope boundary

triangular faces

Three dimensions

Vertices \ Eges \ Faces

\[ n - e + f = 2 \]

\[ 3f = 2e \]
Convex hull

Euler relation

Polytope boundary

Vertices $n$, Edges $e$, Faces $f$

$\begin{align*}
\text{Faces} & : f = 2n - 4 \\
\text{Edges} & : e = 3n - 6 \\
\text{Vertices} & : n - e + f = 2
\end{align*}$

triangular faces

Three dimensions
Convex hull

Three dimensions

Linear size

$O(n \log n)$ divide and conquer algorithm

$O(nh)$ gift wrapping algorithm
Convex hull

Dehn Sommerville relations

\[ f_i = \#(\text{faces of dim } i) \]

Euler:

\[ f_0 - f_1 + f_2 - \ldots f_{d-1} = (-1)^{d-1} + 1 \]
Convex hull

Dehn Sommerville relations \( f_i = \#(\text{faces of dim } i) \)

Euler:
\[
f_0 - f_1 + f_2 - \ldots - f_{d-1} = (-1)^{d-1} + 1
\]

\[
\sum_j = k^{d-1} - 1^j \binom{j + 1}{k + 1} f_j = (-1)^{d-1} f_k
\]

\(-1 \leq k \leq d - 2\)

\[
\left\lfloor \frac{d + 1}{2} \right\rfloor \text{ independent equations}
\]

\(f_{-1} = f_d = 1\)
Convex hull

Dehn Sommerville relations

\[ f_i = \#(\text{faces of dim } i) \]

If \( f_0, f_1, \ldots, f_{\left\lfloor \frac{d-1}{2} \right\rfloor} \) are known

\[ f_{\left\lceil \frac{d+1}{2} \right\rceil}, \ldots, f_{d-1} \] can be deduced
Convex hull

Dehn Sommerville relations \( f_i = \#(\text{faces of dim } i) \)

If \( f_0, f_1, \ldots, f_{\left\lfloor \frac{d-1}{2} \right\rfloor} \) are known

\( f_{\left\lfloor \frac{d+1}{2} \right\rfloor}, \ldots, f_{d-1} \) can be deduced

\( f_{\left\lfloor \frac{d-1}{2} \right\rfloor} = O(n^{\left\lfloor \frac{d+1}{2} \right\rfloor}) \)

\( \implies \forall i \ f_i = O(n^{\left\lfloor \frac{d+1}{2} \right\rfloor}) \)
Convex hull

Dehn Sommerville relations

\[ f_i = \#(\text{faces of dim } i) \]

If \( f_0, f_1, \ldots, f_{\left\lfloor \frac{d-1}{2} \right\rfloor} \) are known

\[ f_{\left\lfloor \frac{d+1}{2} \right\rfloor}, \ldots, f_{d-1} \text{ can be deduced} \]

\[ f_{\left\lfloor \frac{d-1}{2} \right\rfloor} = O(n^{\left\lfloor \frac{d+1}{2} \right\rfloor}) \]

\[ \implies \forall i \ f_i = O(n^{\left\lfloor \frac{d+1}{2} \right\rfloor}) \]

\( \exists \) an optimal algorithm
Linear programming

A simple algorithm [Seidel]

\( n \) linear constraints (half-spaces)
Linear programming

A simple algorithm [Seidel]

$n$ linear constraints (half-spaces)
Linear programming

A simple algorithm [Seidel]

$n$ linear constraints (half-spaces)
Linear programming

A simple algorithm [Seidel]

$n$ linear constraints (half-spaces)
Linear programming

$n$ linear constraints (half-spaces)

A criterion to optimize

A simple algorithm [Seidel]
Linear programming

A simple algorithm [Seidel]

$n$ linear constraints (half-spaces)

A criterion to optimize
Linear programming

A simple algorithm [Seidel]

One dimension

Admissible solutions is an interval

Maintain incrementally
Linear programming

A simple algorithm [Seidel]

One dimension

Admissible solutions is an interval

Maintain incrementally

Easy \( O(n) \)
Linear programming

Two dimensions

A simple algorithm [Seidel]
Linear programming

Two dimensions

Incremental

A simple algorithm [Seidel]
Linear programming

A simple algorithm [Seidel]

Two dimensions

Incremental

Check solution with respect to new constraint
Linear programming

A simple algorithm [Seidel]

Two dimensions

Incremental

Check solution with respect to new constraint

OK  →  do nothing (next constraint)
Linear programming

Two dimensions

Incremental

Check solution with respect to new constraint

OK → do nothing (next constraint)

otherwise → solve in 1D on new constraint
Linear programming

A simple algorithm [Seidel]

Two dimensions

Incremental

Check solution with respect to new constraint

OK \rightarrow \text{do nothing (next constraint)}

otherwise \rightarrow \text{solve in 1D on new constraint}
Linear programming

Complexity

Quadratic in worst case

A simple algorithm [Seidel]
Linear programming

A simple algorithm [Seidel]

Complexity

Quadratic in worst case

Random order

\[ f_2(n) = f_2(n - 1) + \frac{n-2}{n} \cdot 1 + \frac{2}{n} f_1(n - 1) = O(n) \]
Linear programming

A simple algorithm [Seidel]

Complexity

Quadratic in worst case

Random order

\[ f_2(n) = f_2(n - 1) + \frac{n-2}{n} \cdot 1 + \frac{2}{n} f_1(n - 1) = O(n) \]

Higher dimension

\[ f_d(n) = f_d(n - 1) + \frac{n-2}{n} \cdot 1 + \frac{2}{n} f_{d-1}(n - 1) = O(n) \]
The end