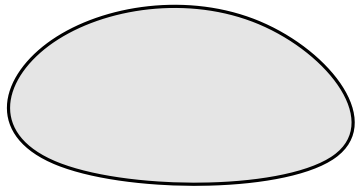
Geometric Algorithms

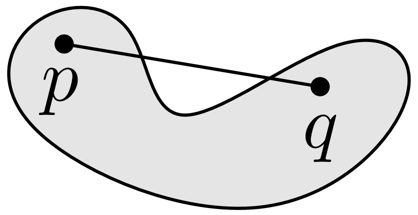
Lecture 3: Convex Hulls

Spring 2025
Computer Science Department
VU Amsterdam, 1081HV Amsterdam

Definition. A set $S \subset \mathbb{R}^2$ is **convex** if for any two points $p, q \in S$ we have $pq \subset S$, where pq is the segment with endpoints p, q.

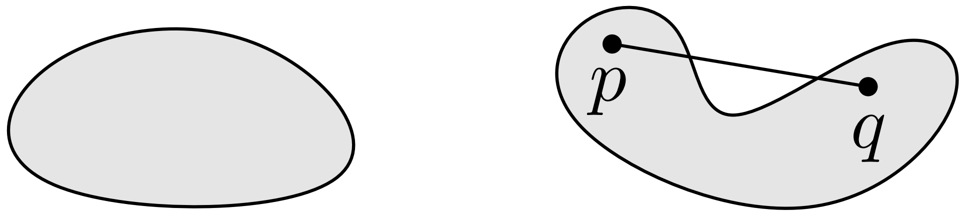


convex



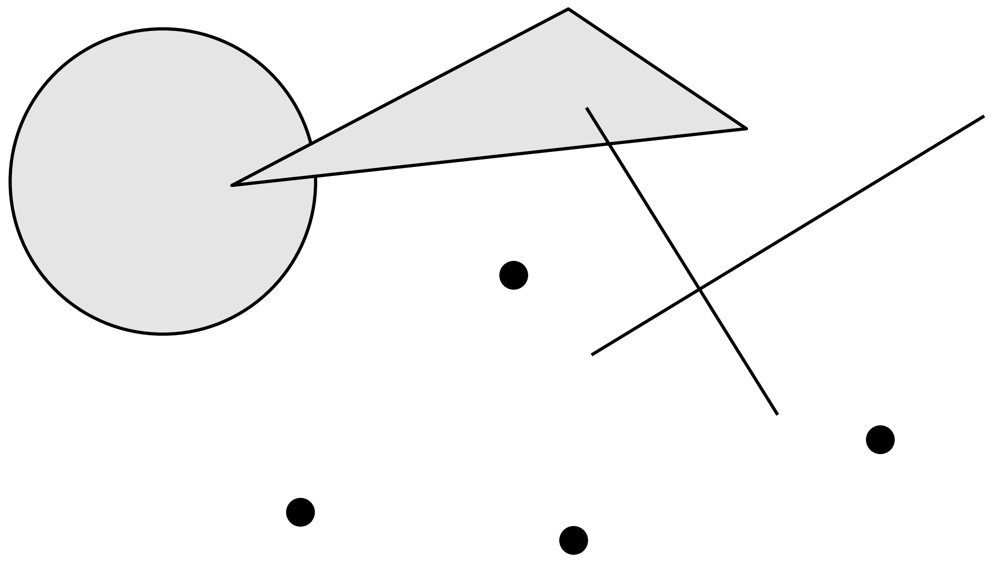
not convex

Definition. A set $S \subset \mathbb{R}^2$ is **convex** if for any two points $p, q \in S$ we have $pq \subset S$, where pq is the segment with endpoints p, q.

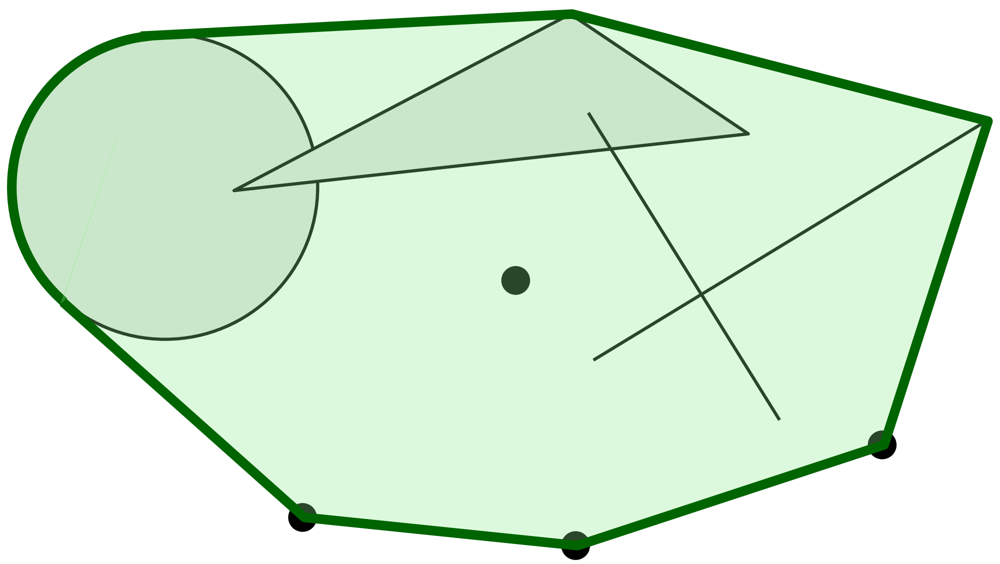


convex not convex

Definition. The **convex hull** of a set S of objects is the "smallest" convex set containing all objects of S.



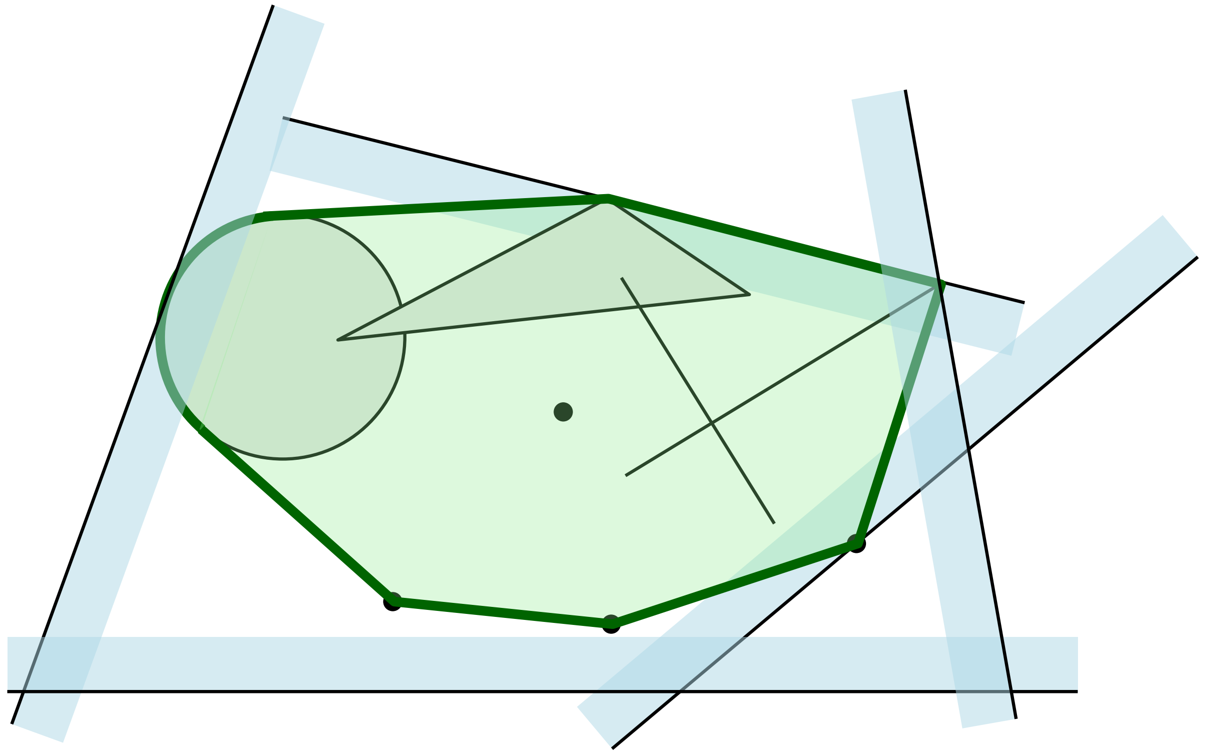
S



$\textsc{ch}(S)$

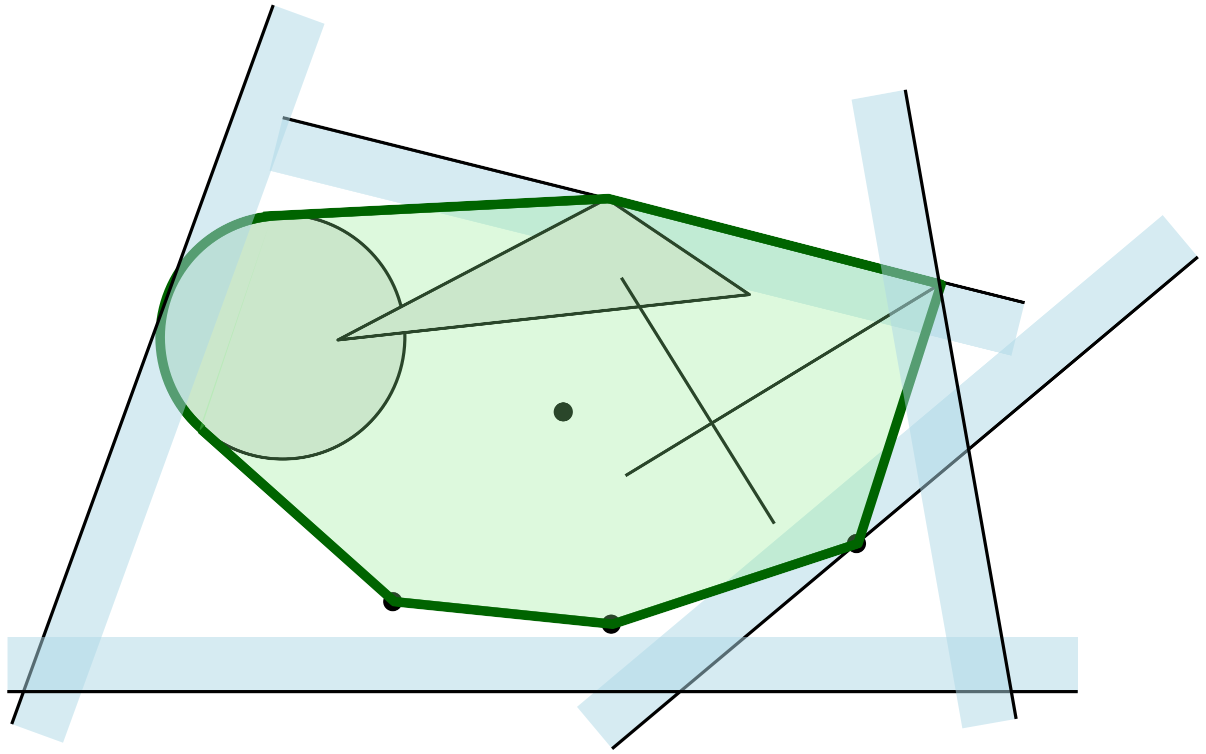
- S = set of objects in the plane.
- $\mathcal{H}(S)$ = (infinite) set of all half-planes h such that, for each $o \in S$, we have $o \in h$ (h covers S.)

Definition. The **convex hull** of a set S of objects in $\mathbb{R}^2$ is defined as $\textsc{ch}(S) := \bigcap_{h \in \mathcal{H}(S)} h$



- S = set of objects in the plane.
- $\mathcal{H}(S)$ = (infinite) set of all half-planes h such that, for each $o \in S$, we have $o \in h$ (h covers S.)

Definition. The **convex hull** of a set S of objects in $\mathbb{R}^2$ is defined as $\textsc{ch}(S) := \cap_{h \in \mathcal{H}(S)} h$

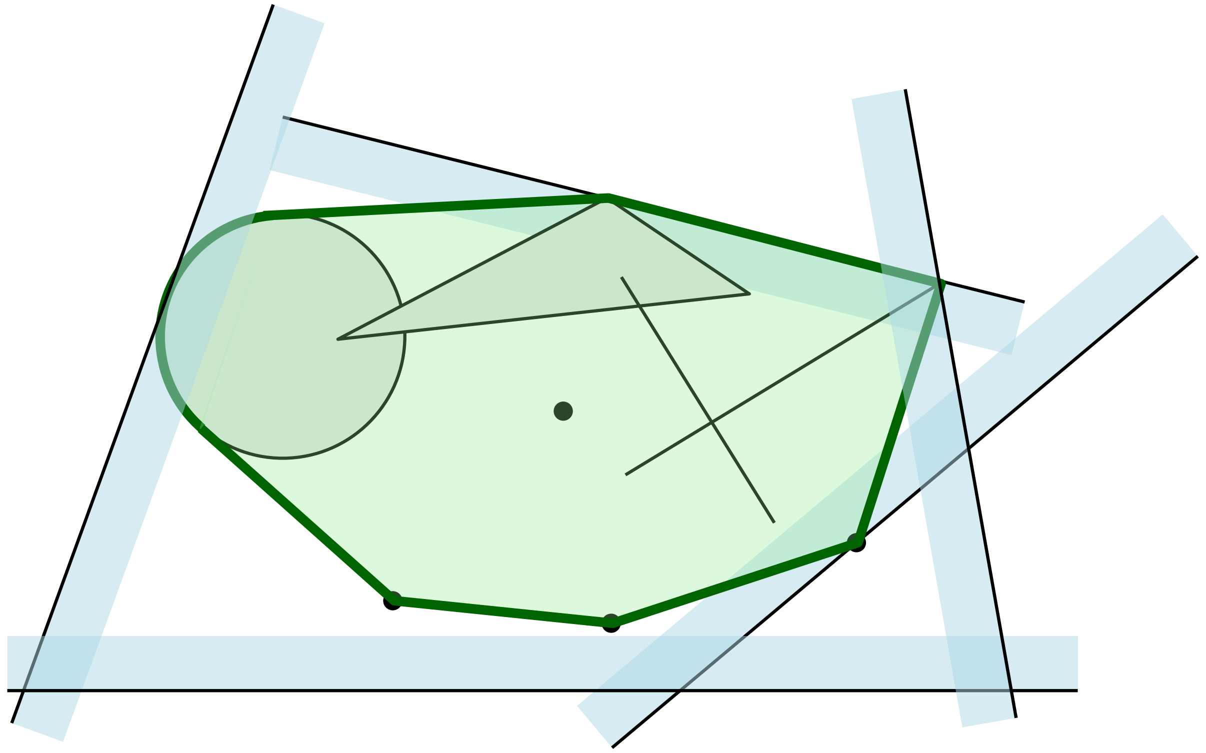


Properties:

- $\textsc{ch}(S)$ is convex.
- $\textsc{ch}(S)$ is the minimum-perimeter set that contains S.
- $\textsc{ch}(S)$ is the minimum-area set that contains S.

- S = set of objects in the plane.
- $\mathcal{H}(S)$ = (infinite) set of all half-planes h such that, for each $o \in S$, we have $o \in h$ (h covers S.)

Definition. The **convex hull** of a set S of objects in $\mathbb{R}^2$ is defined as $\textsc{ch}(S) := \bigcap_{h \in \mathcal{H}(S)} h$

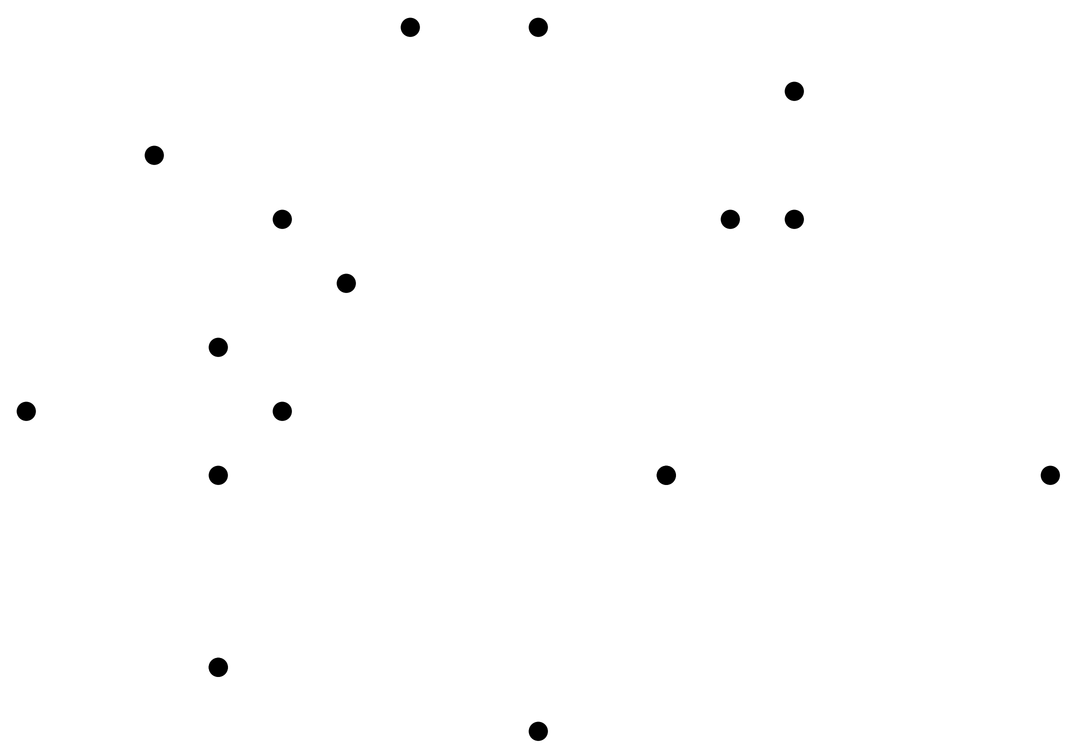


Properties:

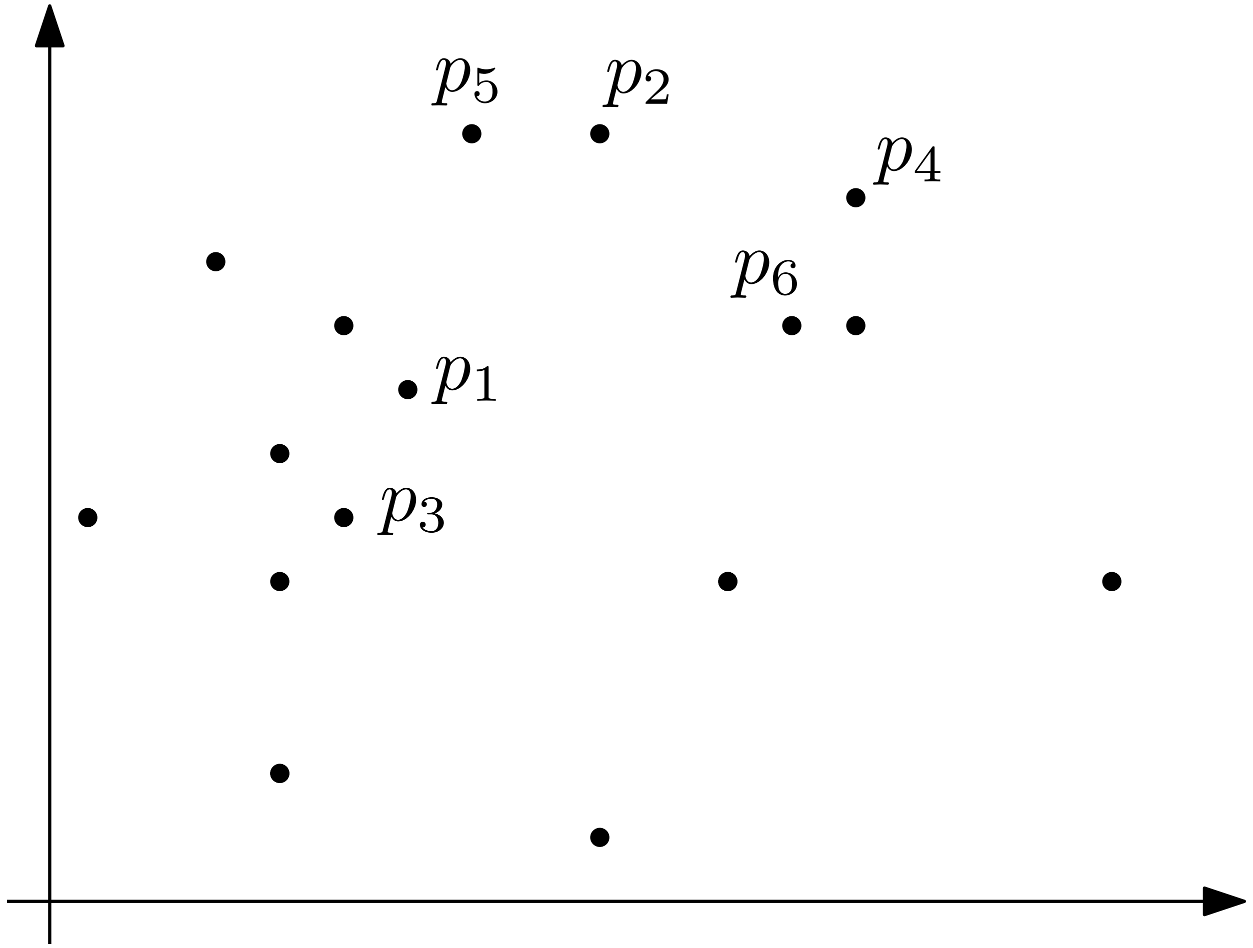
- $\textsc{ch}(S)$ is convex.
- $\textsc{ch}(S)$ is the minimum-perimeter set that contains S.
- $\textsc{ch}(S)$ is the minimum-area set that contains S.

Exercise. Try to prove these properties.

We want to develop an algorithm that computes the convex hull of a set P of n points in the plane.

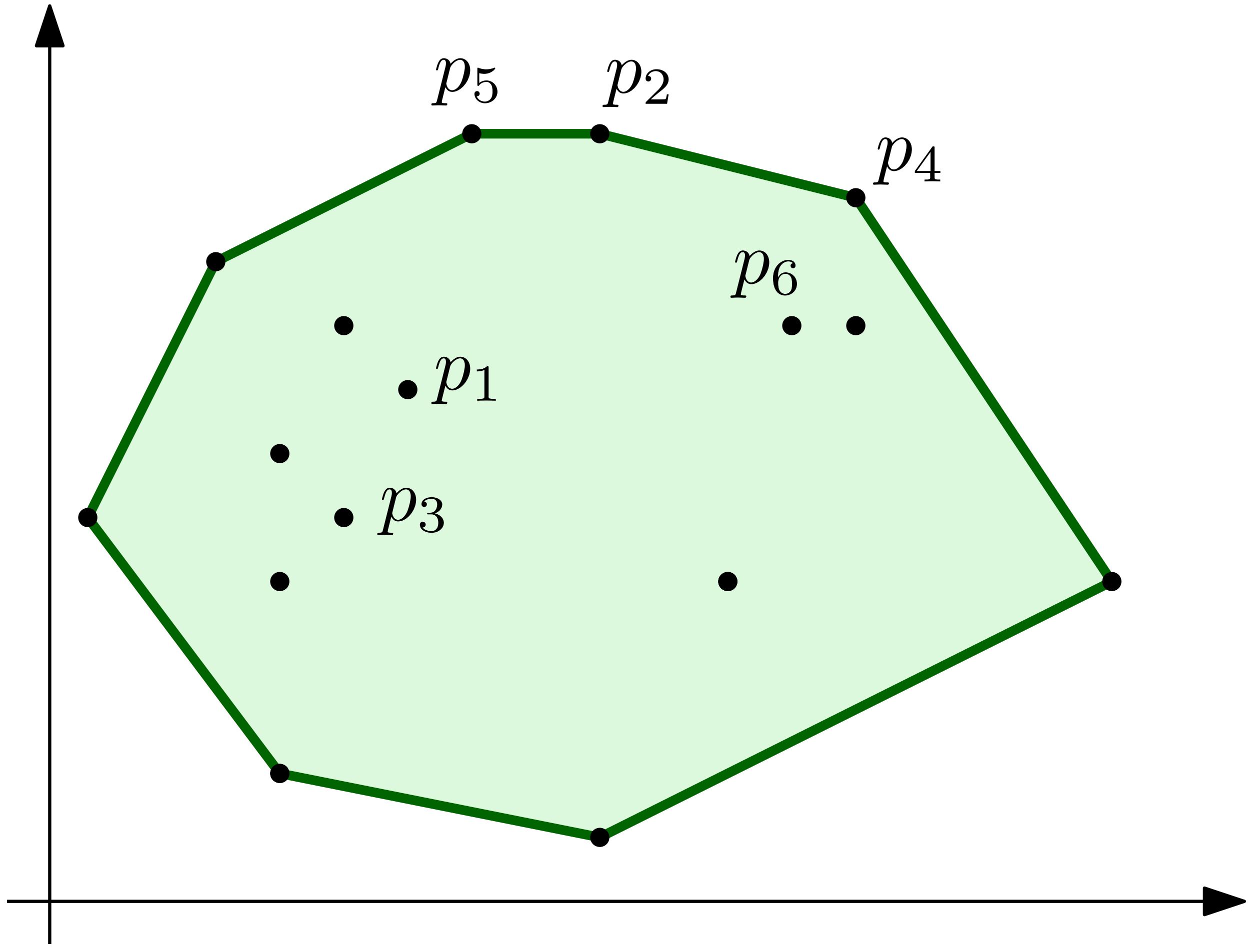


We want to develop an algorithm that computes the convex hull of a set P of n points in the plane.



Input: list of points, for each point its coordinates.

We want to develop an algorithm that computes the convex hull of a set P of n points in the plane.



Input: list of points, for each point its coordinates.

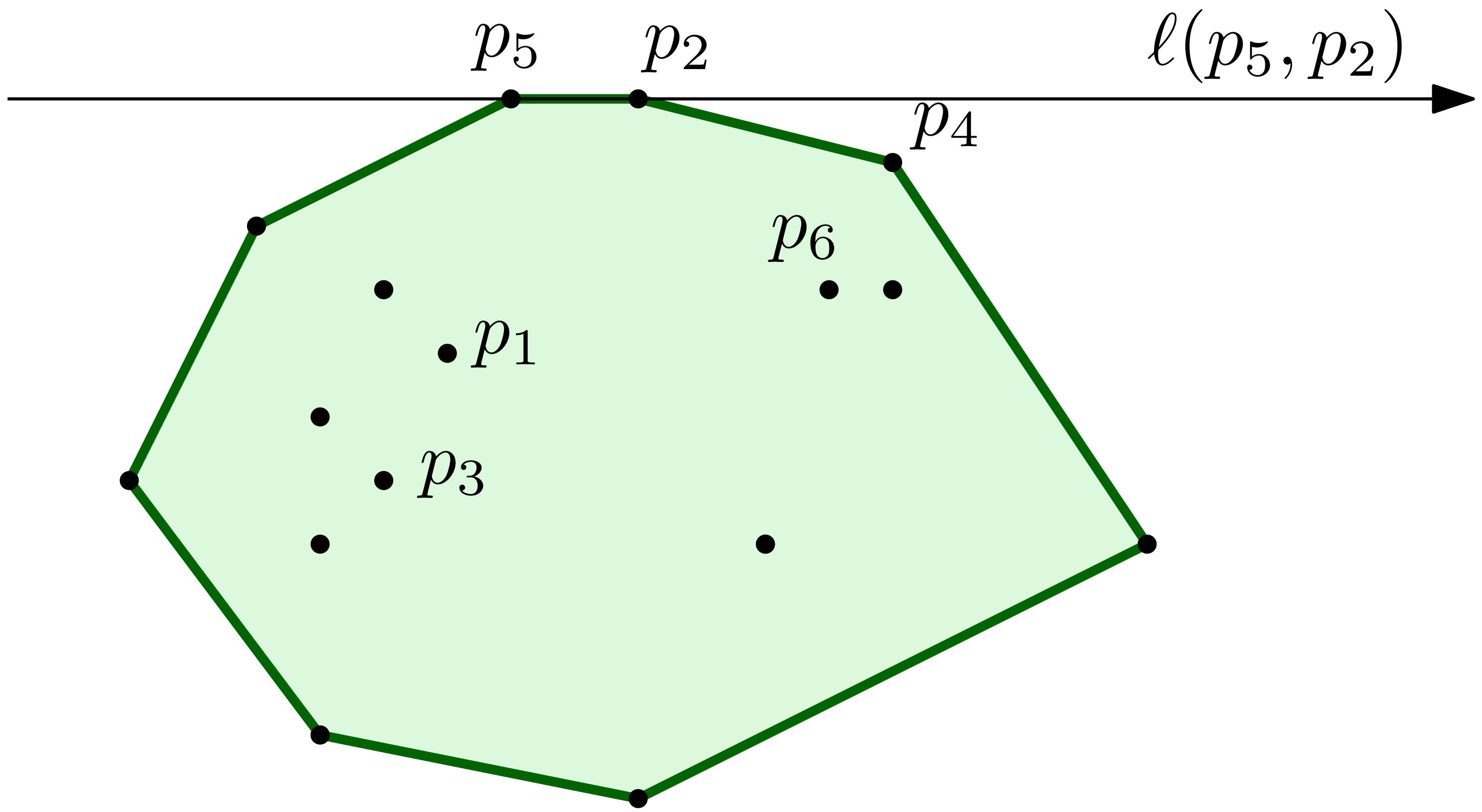
Output: list of vertices of $\text{CH}(P)$ in cyclic order: $p_5, p_2, p_4, \dots$

Lemma. Let $P = \{p_1, \ldots, p_n\}$ be a set of $n \geq 3$ points in the plane that are not all collinear.

(i) $\textsc{ch}(P)$ is a convex polygon whose vertices are points in P.

(ii) Two points p_i, p_j form a clockwise edge of $\textsc{ch}(P)$ if and only if the following holds:

- for all $k \neq i, j$: the point p_k lies to the right of the directed line $\ell(p_i, p_j)$ or p_k lies on $\overline{p_i p_j}$.



A First Convex-Hull Algorithm

SLOW-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

$E \leftarrow \emptyset$ $\quad \triangleright$ E will contain the edges of $\text{CH}(P)$.
for all ordered pairs p_i, p_j with $i \neq j$ **do**
  valid $\leftarrow$ TRUE
  for all points p_k with $k \neq i, j$ **do**
    if $[(p_k$ lies to the right of $\ell(p_i, p_j))$ or $p_k \in \overline{p_i p_j}] =$ FALSE **then**
      valid $\leftarrow$ FALSE
  if *valid* $=$ TRUE **then**
    Add $\overline{p_i p_j}$ to E
Construct the list $\mathcal{L}$ of convex-hull vertices in clockwise order from the set E.
return $\mathcal{L}$

A First Convex-Hull Algorithm

SLOW-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

$E \leftarrow \emptyset$ $\qquad \triangleright$ E will contain the edges of $\textsc{ch}(P)$.
for all ordered pairs p_i, p_j with $i \neq j$ **do**
  valid $\leftarrow$ TRUE
  for all points p_k with $k \neq i, j$ **do**
    if $[(p_k$ lies to the right of $\ell(p_i, p_j))$ or $p_k \in p_i p_j] =$ FALSE **then**
      valid $\leftarrow$ FALSE
  if *valid* $=$ TRUE **then**
    Add $p_i p_j$ to E
Construct the list $\mathcal{L}$ of convex-hull vertices in clockwise order from the set E. **How?**
return $\mathcal{L}$

A First Convex-Hull Algorithm

SLOW-CONVEX-HULL(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

$E \leftarrow \emptyset$ $\quad \triangleright$ E will contain the edges of $\text{CH}(P)$.
for all ordered pairs p_i, p_j with $i \neq j$ **do**
    valid $\leftarrow$ TRUE
    for all points p_k with $k \neq i, j$ **do**
        if $[(p_k$ lies to the right of $\ell(p_i, p_j))$ or $p_k \in \overline{p_i p_j}] =$ FALSE **then**
            valid $\leftarrow$ FALSE
    if *valid* $=$ TRUE **then**
        Add $\overline{p_i p_j}$ to E
Construct the list $\mathcal{L}$ of convex-hull vertices in clockwise order from the set E. **How?**
return $\mathcal{L}$

Running Time?

A First Convex-Hull Algorithm

$\textsc{Slow-Convex-Hull}(P)$
Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

$E \leftarrow \emptyset$ $\quad\quad \triangleright$ E will contain the edges of $\textsc{ch}(P)$.
for all ordered pairs p_i, p_j with $i \neq j$ **do**
    valid $\leftarrow$ TRUE
    for all points p_k with $k \neq i, j$ **do**
        if $[(p_k$ lies to the right of $\ell(p_i, p_j))$ or $p_k \in \overline{p_i p_j}] =$ FALSE **then**
            valid $\leftarrow$ FALSE
    if *valid* $=$ TRUE **then**
        Add $\overline{p_i p_j}$ to E
Construct the list $\mathcal{L}$ of convex-hull vertices in clockwise order from the set E. **How?**
return $\mathcal{L}$

Running Time? $O(n^3)$

We now have an $O(n^3)$-time algorithm to compute the convex hull of n points in the plane.

Can we do better?

- $O(n^2)$?
- $O(n \log n)$?
- $O(n)$?

We now have an $O(n^3)$-time algorithm to compute the convex hull of n points in the plane.

Can we do better?

- $O(n^2)$?
- $O(n \log n)$?
- $O(n)$?

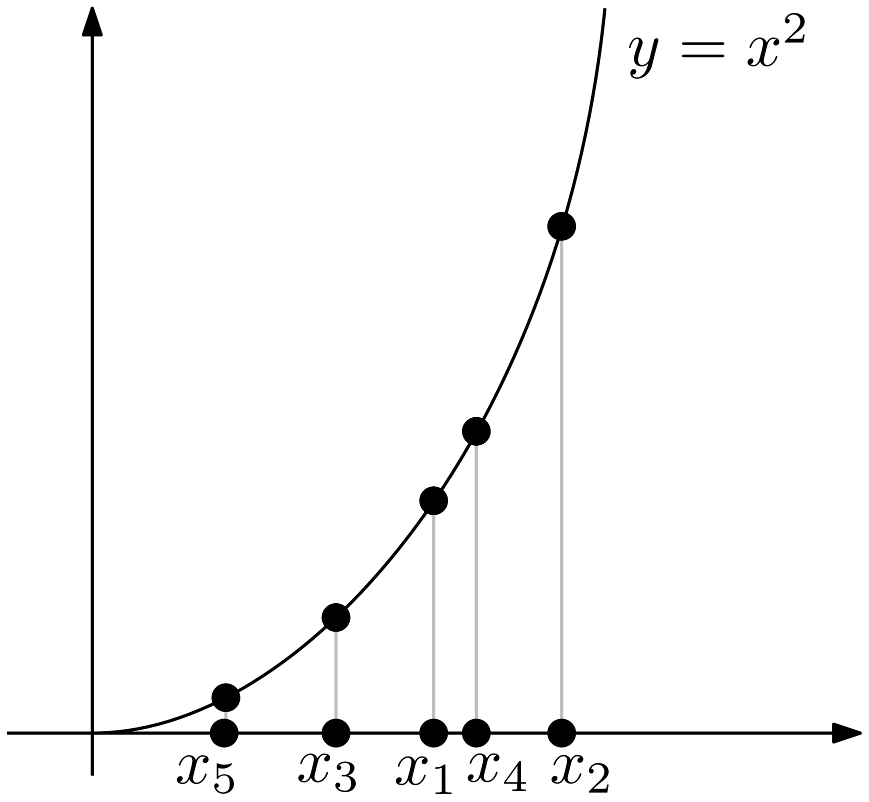
Theorem. Any algorithm for computing the convex hull of n points in the plane needs $\Omega(n \log n)$ time in the worst case.

We now have an $O(n^3)$-time algorithm to compute the convex hull of n points in the plane.

Can we do better?

- $O(n^2)$?
- $O(n \log n)$?
- $O(n)$?

Theorem. Any algorithm for computing the convex hull of n points in the plane needs $\Omega(n \log n)$ time in the worst case.



Proof. By a reduction from the $\Omega(n \log n)$-time lower bound for sorting.

A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

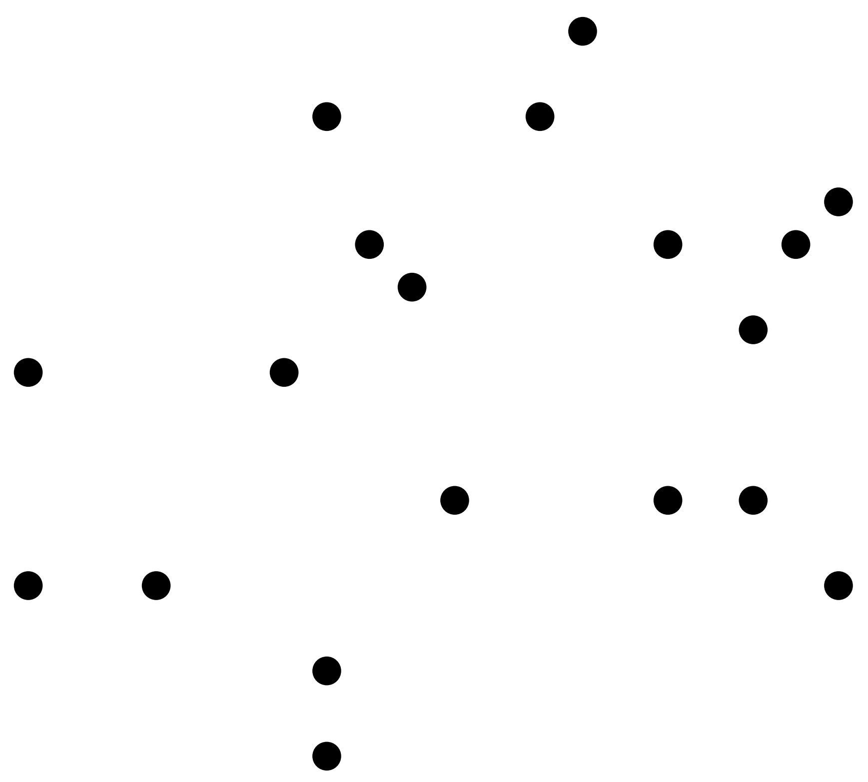
$\textsc{Divide-and-Conquer-Convex-Hull}(P)$
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

```
if n ⩽ 3 then
    Compute CH(P) in a brute-force manner.
else
    Split P into a subset P1 = {p1, ..., p⌈n/2⌉}
    and a subset P2 = {p⌈n/2⌉+1, ..., pn}.
    Compute CH(P1) and CH(P2) recursively.
    Compute CH(P) from CH(P1) and CH(P2).
```

Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into a subset $P_1 = \{p_1, \ldots, p_{\lceil n/2 \rceil}\}$
    and a subset $P_2 = \{p_{\lceil n/2 \rceil + 1}, \ldots, p_n\}$.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.

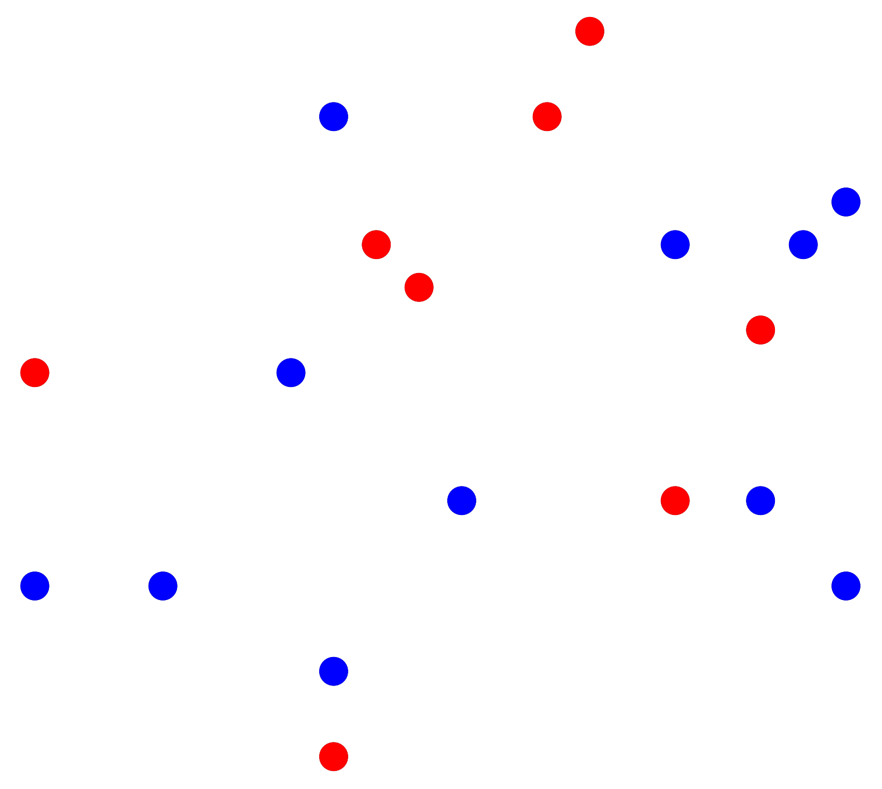


A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

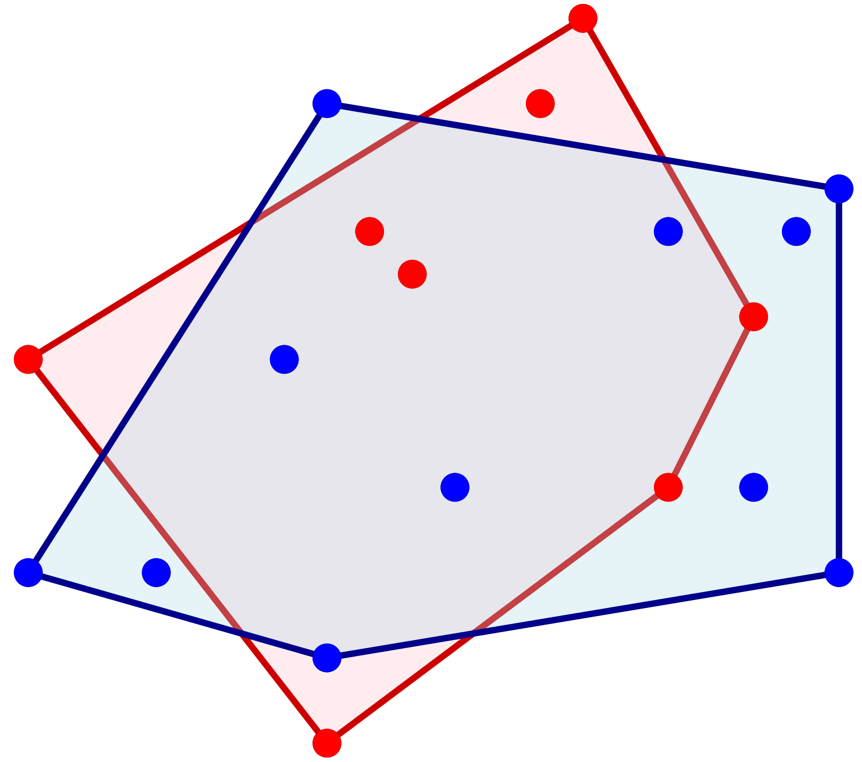
```
if n ⩽ 3 then
    Compute CH(P) in a brute-force manner.
else
    Split P into a subset P_1 = {p_1, ..., p_⌈n/2⌉}
    and a subset P_2 = {p_⌈n/2⌉+1, ..., p_n}.
    Compute CH(P_1) and CH(P_2) recursively.
    Compute CH(P) from CH(P_1) and CH(P_2).
```



Let's try a standard algorithmic design technique: **divide-and-conquer.**

Divide-and-Conquer-Convex-Hull(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: **if** $n \leqslant 3$ **then**
2: Compute $\text{CH}(P)$ in a brute-force manner.
3: **else**
4: Split P into a subset $P_1 = \{p_1, \ldots, p_{\lceil n/2 \rceil}\}$ and a subset $P_2 = \{p_{\lceil n/2 \rceil + 1}, \ldots, p_n\}$.
5: Compute $\text{CH}(P_1)$ and $\text{CH}(P_2)$ recursively.
6: Compute $\text{CH}(P)$ from $\text{CH}(P_1)$ and $\text{CH}(P_2)$.

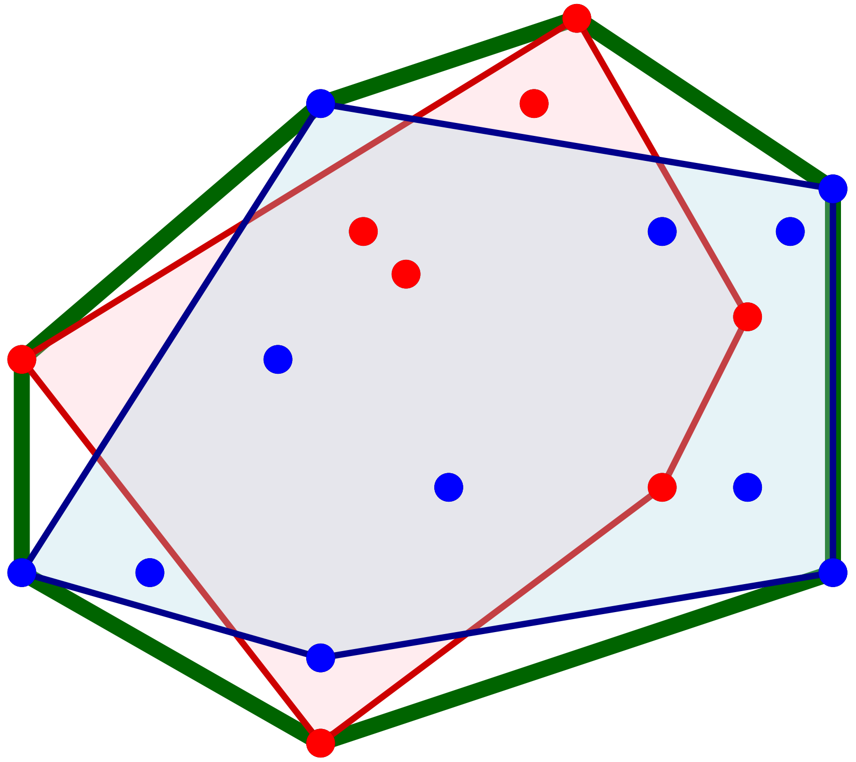


A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into a subset $P_1 = \{p_1, \ldots, p_{\lceil n/2 \rceil}\}$
    and a subset $P_2 = \{p_{\lceil n/2 \rceil + 1}, \ldots, p_n\}$.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.

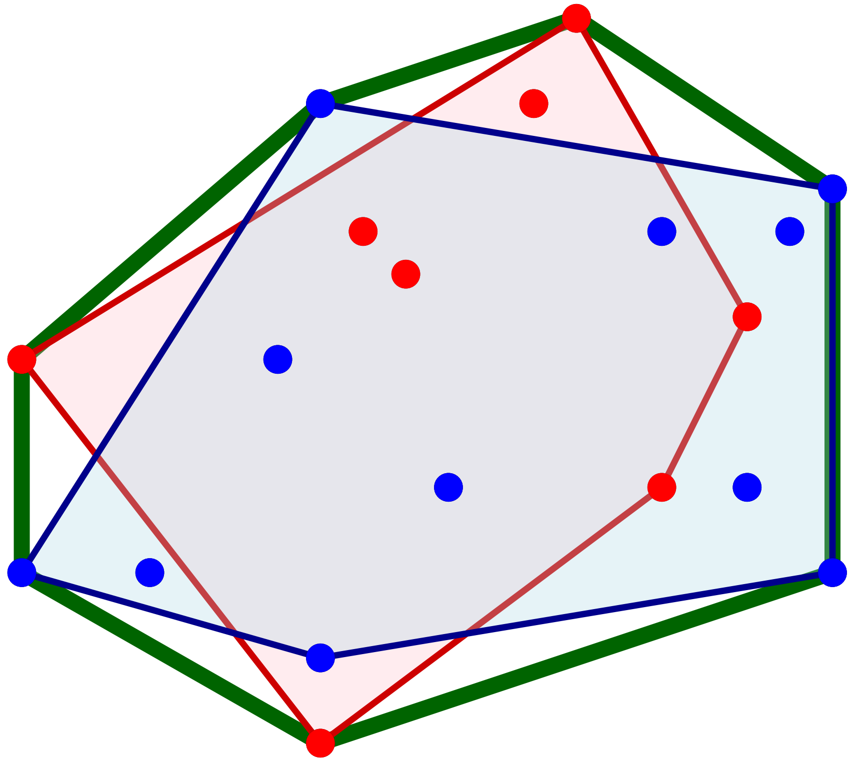


A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

$\textsc{Divide-and-Conquer-Convex-Hull}(P)$
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into a subset $P_1 = \{p_1, \ldots, p_{\lceil n/2 \rceil}\}$
    and a subset $P_2 = \{p_{\lceil n/2 \rceil + 1}, \ldots, p_n\}$.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.



For geometric divide-and-conquer algorithms it is often better to do the divide step in a geometric way, since this will simplify the merge step.

A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

$\textsc{Divide-and-Conquer-Convex-Hull}(P)$
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

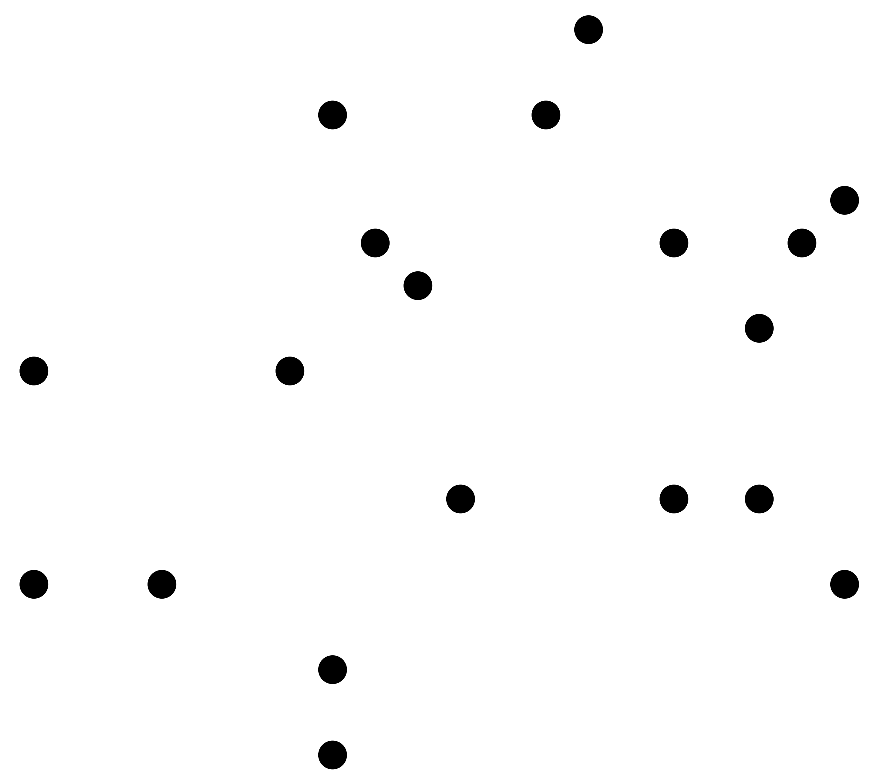
if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into subsets P_1 and P_2 with a vertical line.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.

A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

$\textsc{Divide-and-Conquer-Convex-Hull}(P)$
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into subsets P_1 and P_2 with a vertical line.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.

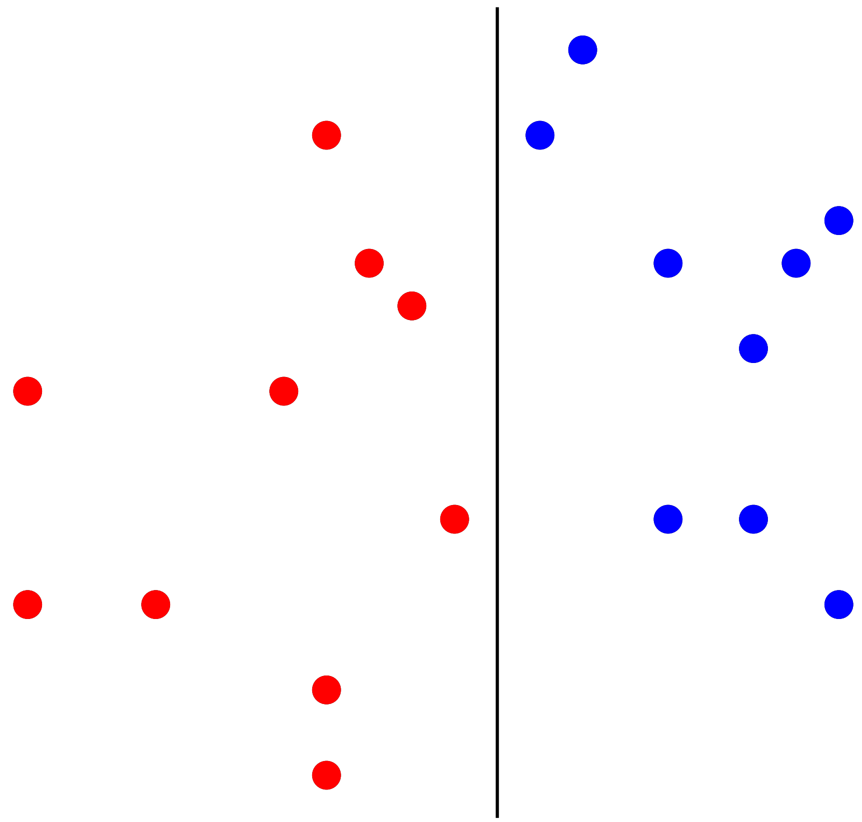


A Worst-Case Optimal Convex-Hull Algorithm

Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

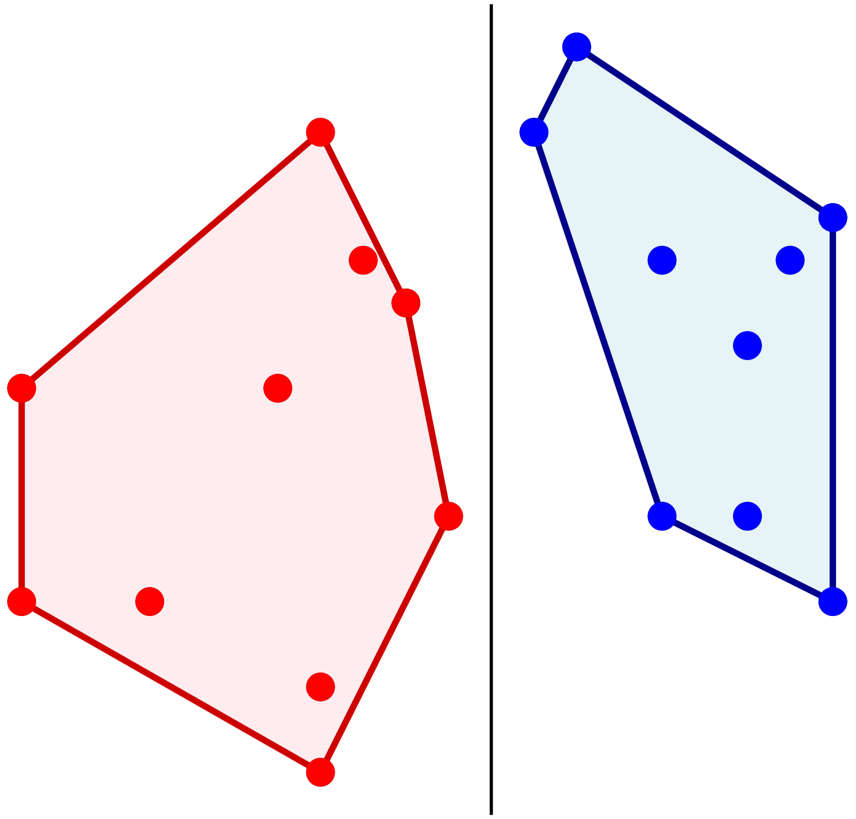
if $n \leqslant 3$ **then**
    Compute $\textsc{ch}(P)$ in a brute-force manner.
else
    Split P into subsets P_1 and P_2 with a vertical line.
    Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
    Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.



Let's try a standard algorithmic design technique: **divide-and-conquer.**

Divide-and-Conquer-Convex-Hull(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

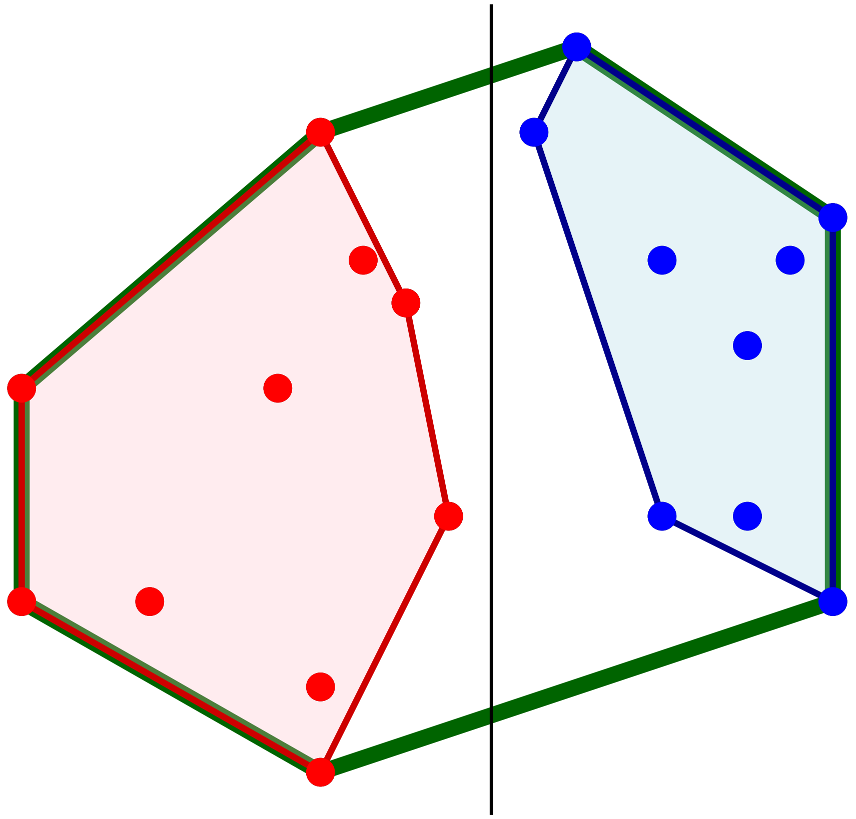
```
if n ⩽ 3 then
    Compute CH(P) in a brute-force manner.
else
    Split P into subsets P1 and P2 with a vertical line.
    Compute CH(P1) and CH(P2) recursively.
    Compute CH(P) from CH(P1) and CH(P2).
```



Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

1: **if** $n \leqslant 3$ **then**
2: Compute $\textsc{ch}(P)$ in a brute-force manner.
3: **else**
4: Split P into subsets P_1 and P_2 with a vertical line.
5: Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
6: Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.

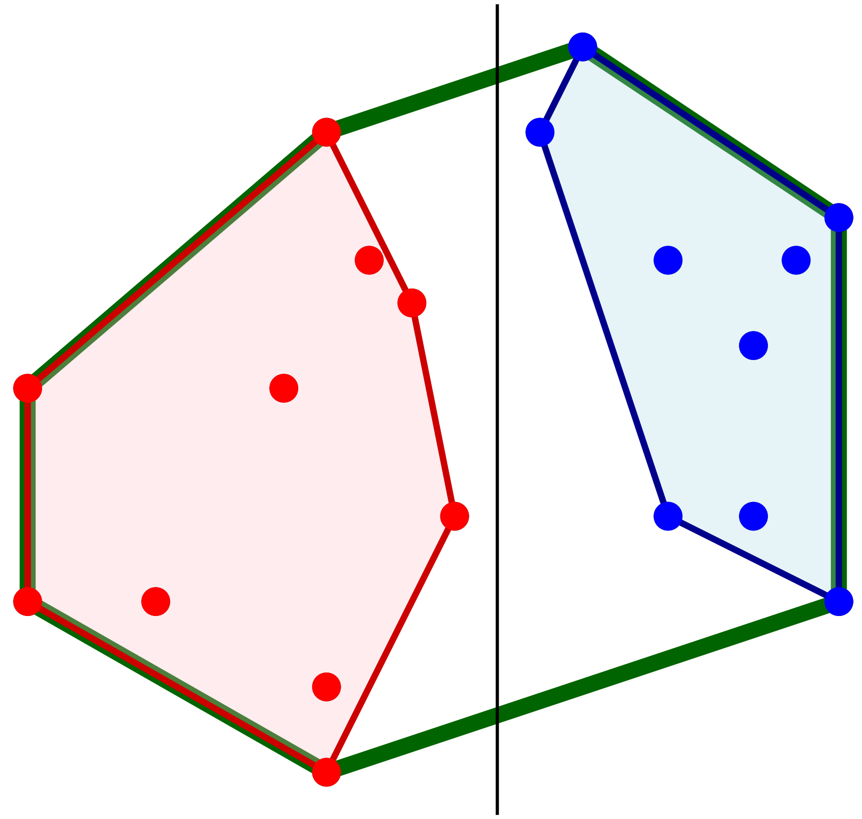


only two new edges

Let's try a standard algorithmic design technique: **divide-and-conquer.**

DIVIDE-AND-CONQUER-CONVEX-HULL(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

1: **if** $n \leqslant 3$ **then**
2: Compute $\textsc{ch}(P)$ in a brute-force manner.
3: **else**
4: Split P into subsets P_1 and P_2 with a vertical line.
5: Compute $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$ recursively.
6: Compute $\textsc{ch}(P)$ from $\textsc{ch}(P_1)$ and $\textsc{ch}(P_2)$.



only two new edges

The merge step is not so easy but can be done in $O(n)$ time $\Longrightarrow$ the algorithm runs in

$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

time.

Let's try another standard (geometric) algorithmic design technique: **incremental construction.**

$\textsc{Incremental-Convex-Hull}(P)$
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

▷ Define $P_i := \{p_1, \ldots, p_i\}$.
1: Compute $\textsc{ch}(P_3)$ brute-force.
2: **for** $i \leftarrow 4$ to n **do**
3: Compute $\textsc{ch}(P_i)$ from $\textsc{ch}(P_{i-1})$ and the point p_i.
4: **return** $\textsc{ch}(P_n)$

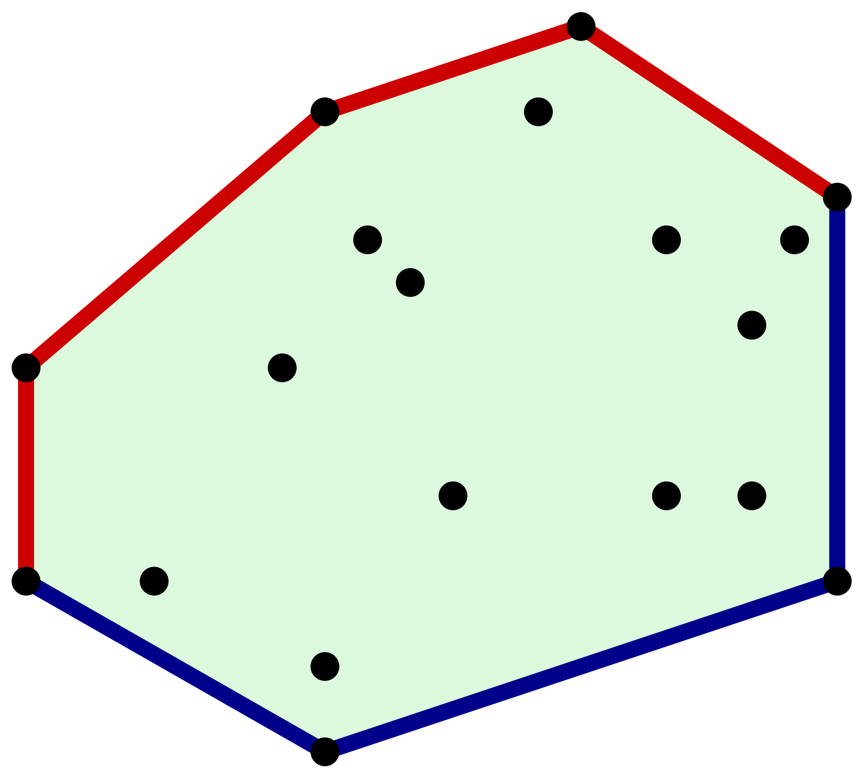
Let's try another standard (geometric) algorithmic design technique: **incremental construction.**

INCREMENTAL-CONVEX-HULL(P)
Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

▷ Define $P_i := \{p_1, \dots, p_i\}$.
1: Compute $\text{CH}(P_3)$ brute-force.
2: **for** $i \leftarrow 4$ to n **do**
3: Compute $\text{CH}(P_i)$ from $\text{CH}(P_{i-1})$ and the point p_i.
4: **return** $\text{CH}(P_n)$

For incremental geometric algorithms it is often better treat the objects in an order determined by the geometry, for example from left to right.

To simplify the algorithm let's compute the **upper hull** and the **lower hull** separately.



GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\textsc{ch}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\textsc{ch}(P)$

Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.

$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\quad\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.

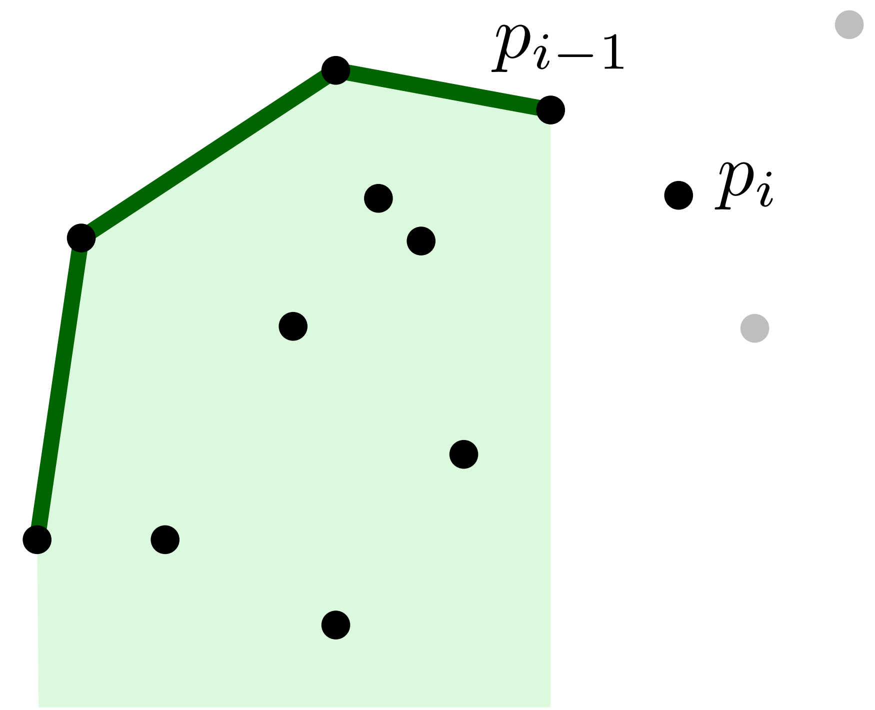
for $i \leftarrow 3$ to n **do**

Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.

Compute lower hull $\mathcal{L}$ in a similar way, from left to right.

$\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)

return $\text{CH}(P)$

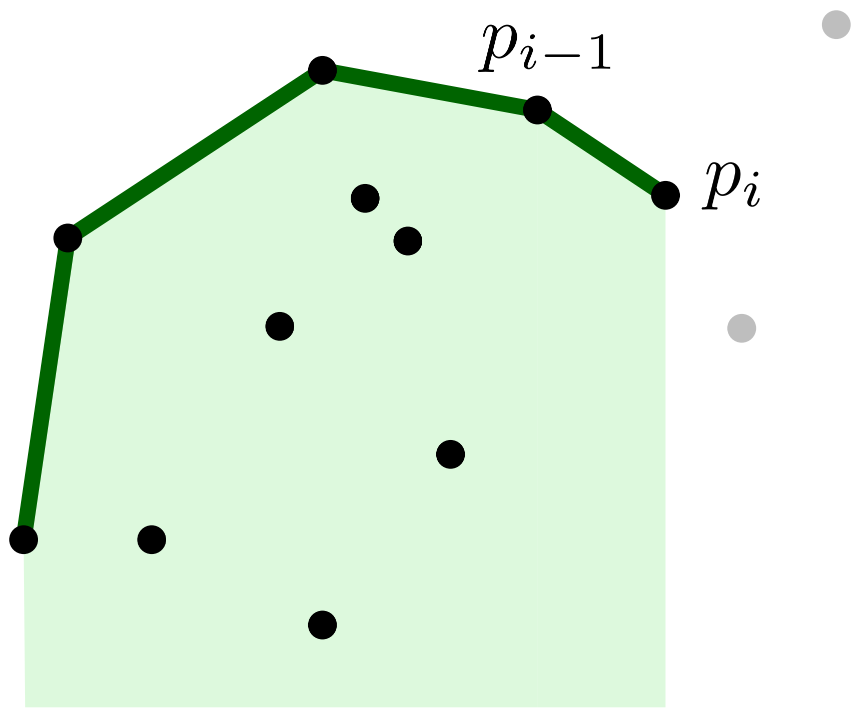


Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

Graham-Scan(P)
Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \dots, p_n$ be the sorted list.
$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
for $i \leftarrow 3$ to n **do**
  Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.
Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
$\textsc{ch}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
return $\textsc{ch}(P)$



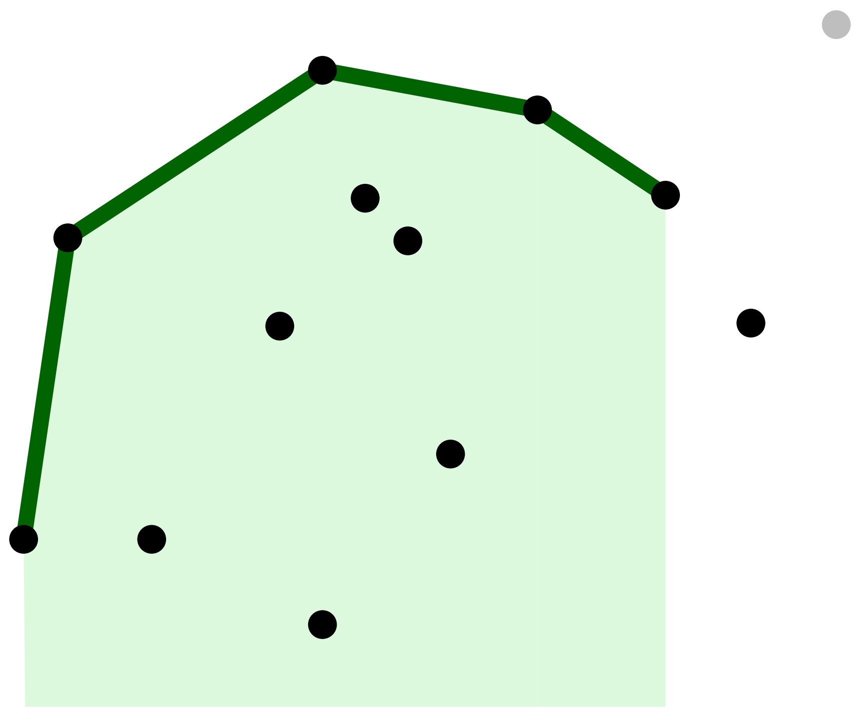
Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \dots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

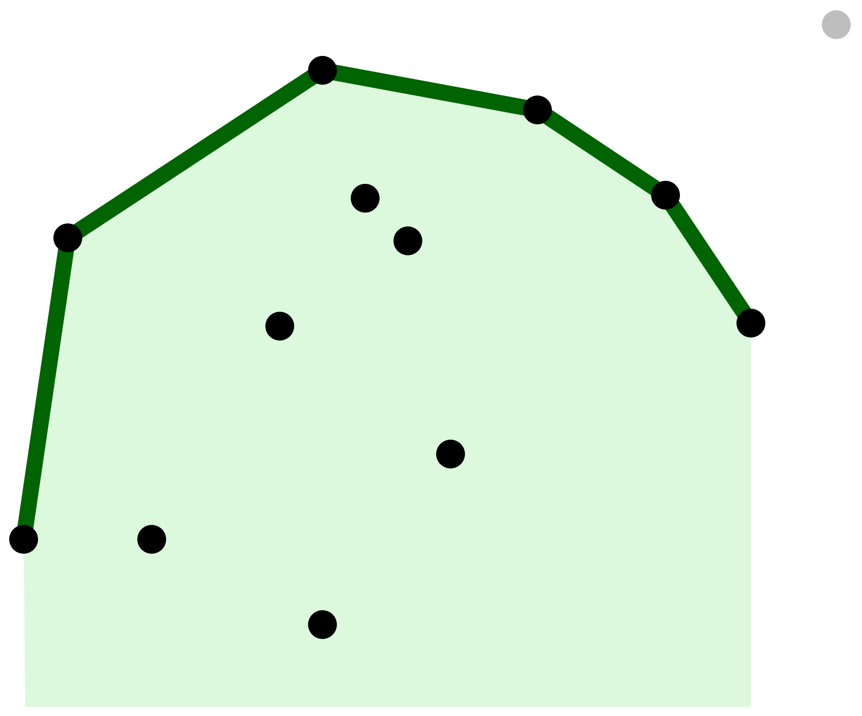
1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\textsc{ch}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\textsc{ch}(P)$



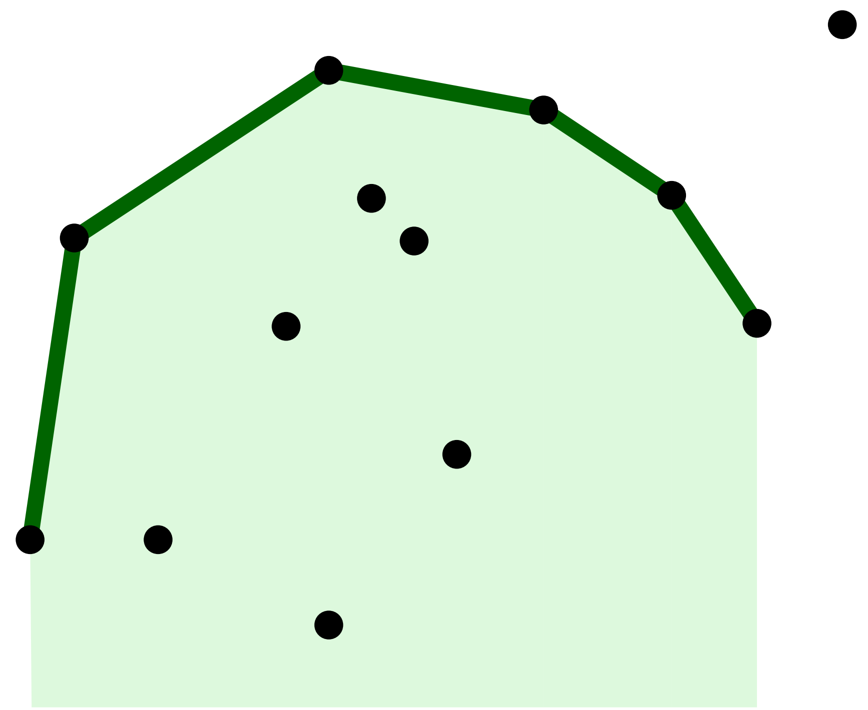
Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Update the upper hull $\mathcal{U}$ by adding p_i and removing other points if necessary.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow (\ \mathcal{U}$ concatenated to $\mathcal{L}\ )$
7: **return** $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

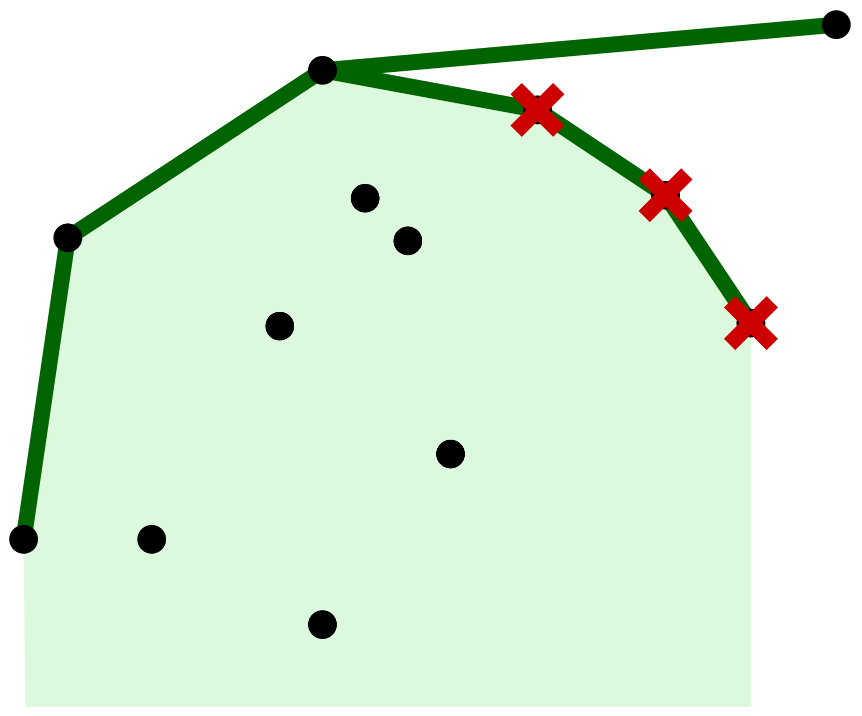
Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

Graham-Scan(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
for $i \leftarrow 3$ to n **do**
    Update the upper hull $\mathcal{U}$ by adding p_i
    and removing other points if necessary.
Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
$\text{CH}(P) \leftarrow (\ \mathcal{U}$ concatenated to $\mathcal{L}\ )$
return $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

$\textsc{Graham-Scan}(P)$

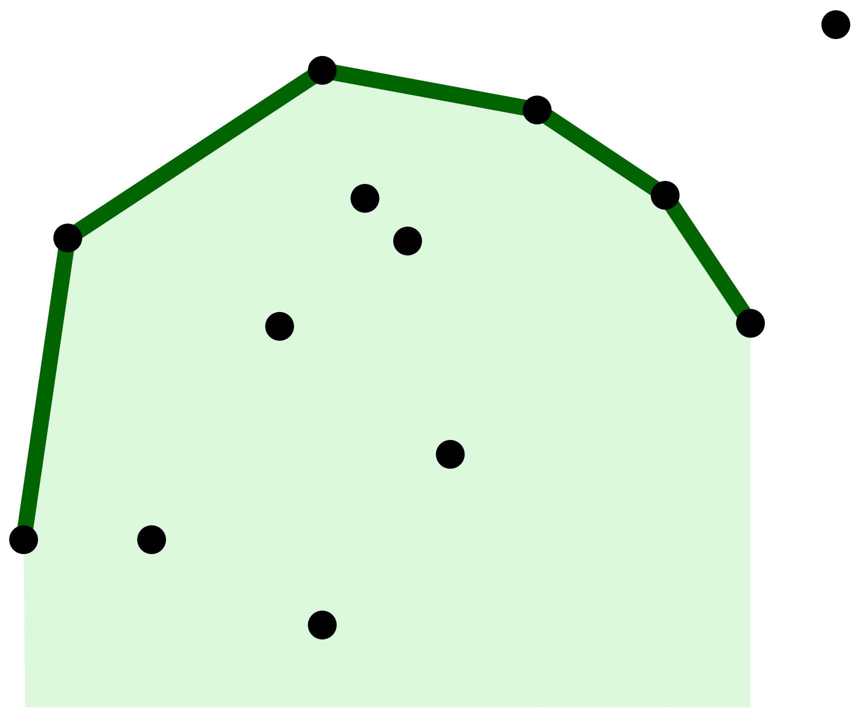
Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \dots, p_n$ be the sorted list.
$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\quad \triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
for $i \leftarrow 3$ to n **do**

> Append p_i to $\mathcal{U}$.
> **while** the last three points in $\mathcal{U}$ do not make a right turn
> **do** remove the middle of the three points.

Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
$\textsc{ch}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
return $\textsc{ch}(P)$



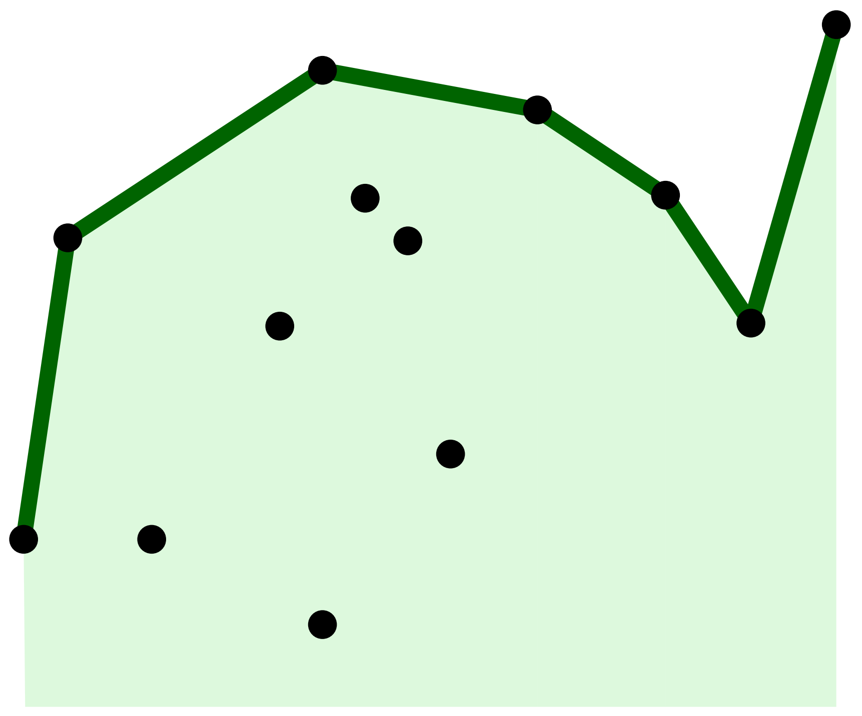
Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \dots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \dots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4:
 Append p_i to $\mathcal{U}$.
 while the last three points in $\mathcal{U}$ do not make a right turn
 do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \dots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.

2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\quad \triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.

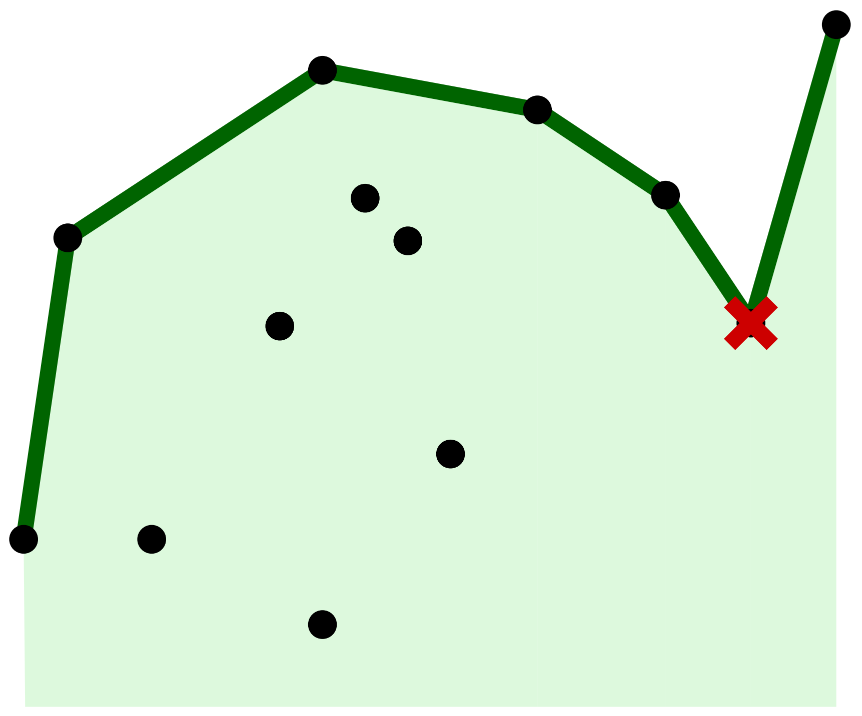
3: **for** $i \leftarrow 3$ to n **do**

4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.

5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.

6: $\text{CH}(P) \leftarrow (\ \mathcal{U}$ concatenated to $\mathcal{L}\ )$

7: **return** $\text{CH}(P)$

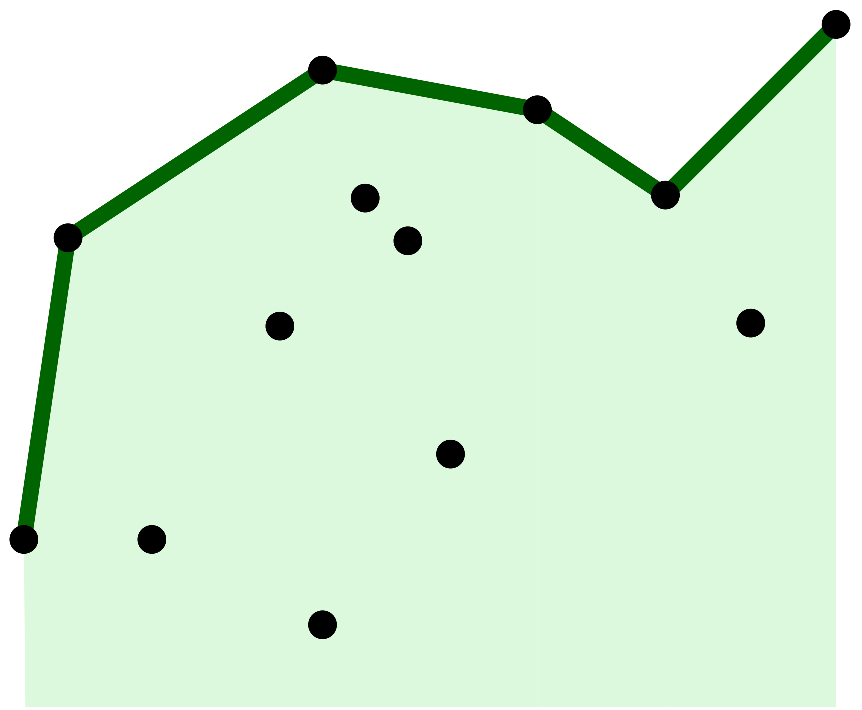


Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4:
Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.

5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\mathrm{CH}(P)$

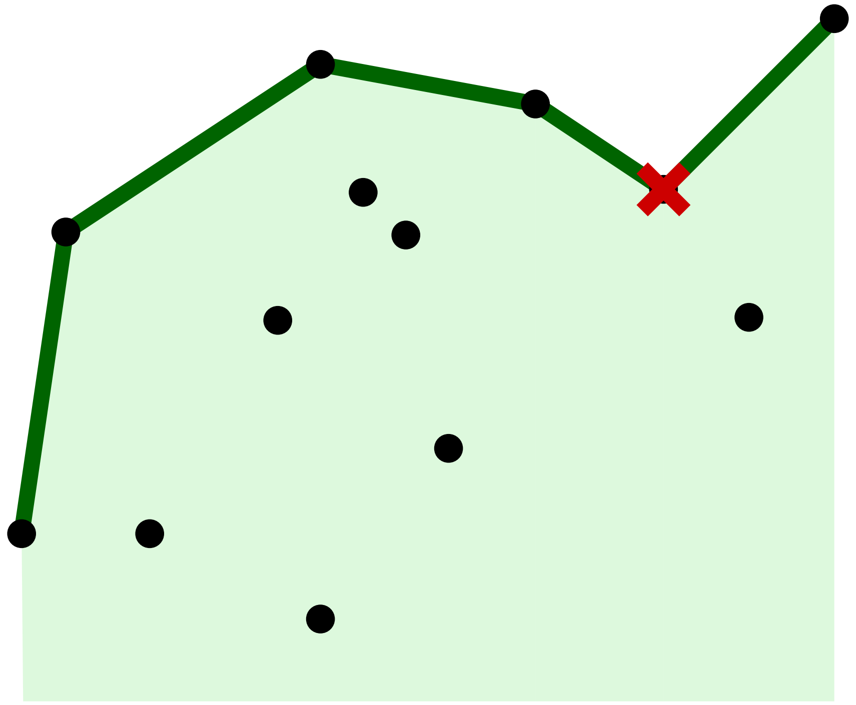


Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

Graham-Scan(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

```
Sort P by x-coordinate. Let p_1, ..., p_n be the sorted list.
U ← ⟨p_1, p_2⟩          ▷ U is a list containing upper-hull vertices.
for i ← 3 to n do
    Append p_i to U.
    while the last three points in U do not make a right turn
        do remove the middle of the three points.
Compute lower hull L in a similar way, from left to right.
CH(P) ← ( U concatenated to L )
return CH(P)
```



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

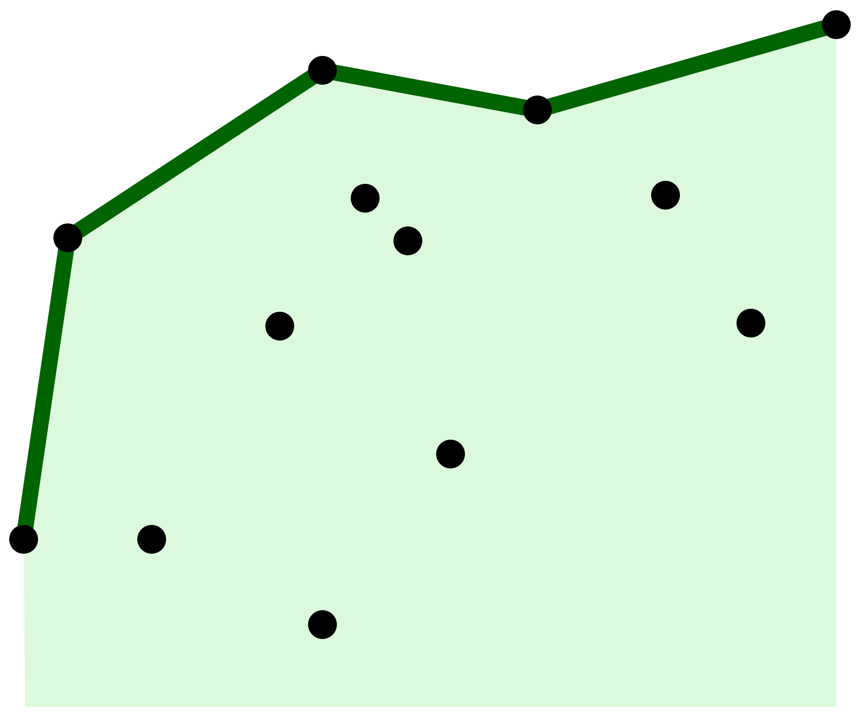
$\textsc{Graham-Scan}(P)$

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\qquad \triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4:
> Append p_i to $\mathcal{U}$.
> **while** the last three points in $\mathcal{U}$ do not make a right turn
> **do** remove the middle of the three points.

5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\textsc{ch}(P) \leftarrow (\ \mathcal{U}$ concatenated to $\mathcal{L}\ )$
7: **return** $\textsc{ch}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.

$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.

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

Append p_i to $\mathcal{U}$.

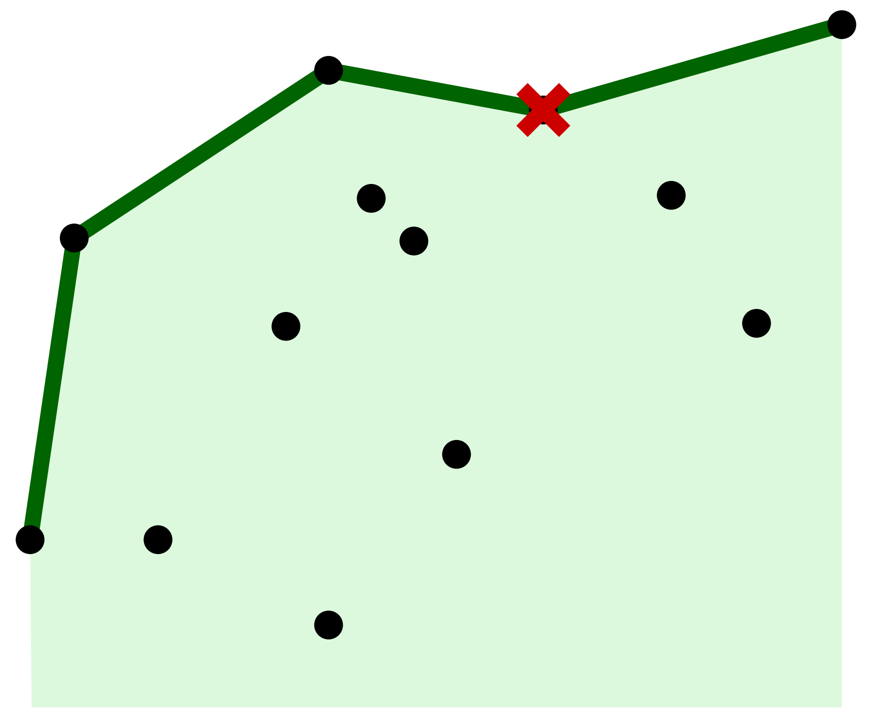
while the last three points in $\mathcal{U}$ do not make a right turn

do remove the middle of the three points.

Compute lower hull $\mathcal{L}$ in a similar way, from left to right.

$\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)

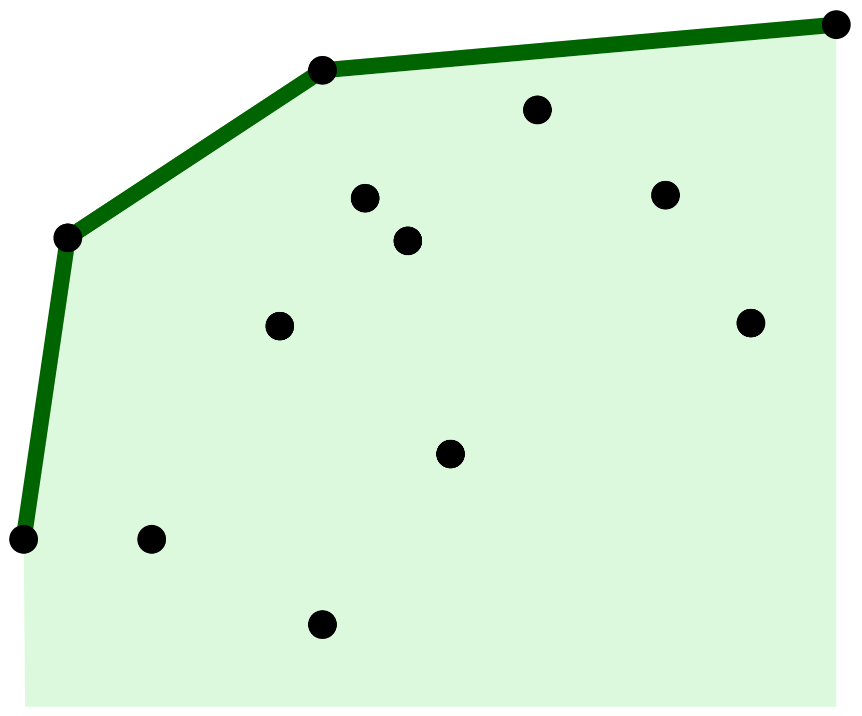
return $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

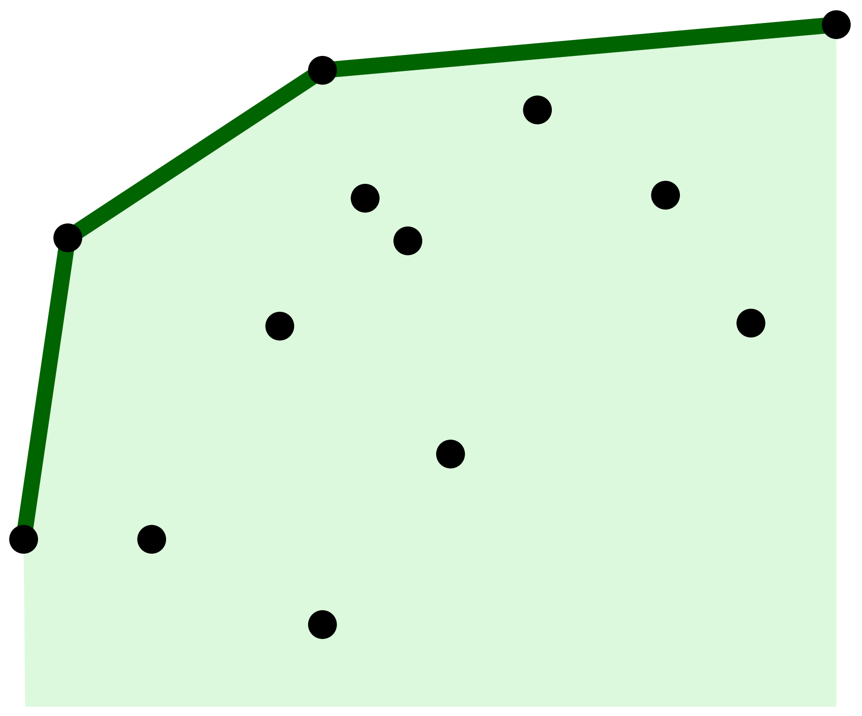
1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\mathrm{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$



Invariant. Just before p_i is added, $\mathcal{U}$ contains the vertices of the upper hull of $\{p_1, \ldots, p_{i-1}\}$ from left to right.

Invariant restored!

Running Time:

Graham-Scan(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\quad \triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\mathrm{CH}(P)$

Graham-Scan(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

Running Time:

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list. $\longrightarrow O(n \log n)$
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices. $\longrightarrow O(1)$
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$ $\longrightarrow O(n)$

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4:
Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$

Running Time:

$O(n \log n)$

$O(1)$

$\sum_{i=3}^{n}$ time to add p_i

$O(n)$

GRAHAM-SCAN(P)

Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.

Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.

$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.

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

Append p_i to $\mathcal{U}$.

while the last three points in $\mathcal{U}$ do not make a right turn

do remove the middle of the three points.

Compute lower hull $\mathcal{L}$ in a similar way, from left to right.

$\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)

return $\mathrm{CH}(P)$

Running Time:

$O(n \log n)$

$O(1)$

$\sum_{i=3}^{n}$ time to add p_i

worst-case $O(n)$

$O(n)$

Running Time:

Graham-Scan(P)
Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

Sort P by x-coordinate. Let $p_1, \dots, p_n$ be the sorted list. → $O(n \log n)$
$\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices. → $O(1)$
for $i \leftarrow 3$ to n **do**
  Append p_i to $\mathcal{U}$.
  while the last three points in $\mathcal{U}$ do not make a right turn
    do remove the middle of the three points.

→ $\sum_{i=3}^{n}$ time to add p_i

worst-case $O(n)$

Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
$\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
return $\mathrm{CH}(P)$
→ $O(n)$

+

$O(n^2)$

Graham's Scan: A Worst-Case Optimal Incremental-Based Convex-Hull Algorithm

Graham-Scan(P)

Input: set $P = \{p_1, \dots, p_n\}$ of n points in the plane..

Output: A list of vertices of $\textsc{ch}(P)$ in clockwise order.

```
Sort P by x-coordinate. Let p_1, ..., p_n be the sorted list.
𝒰 ← ⟨p_1, p_2⟩        ▷ 𝒰 is a list containing upper-hull vertices.
for i ← 3 to n do
    Append p_i to 𝒰.
    while the last three points in 𝒰 do not make a right turn
        do remove the middle of the three points.
Compute lower hull ℒ in a similar way, from left to right.
CH(P) ← ( 𝒰 concatenated to ℒ )
return CH(P)
```

Running Time:

$O(n \log n)$

$O(1)$

$\sum_{i=3}^{n}$ time to add p_i

$O(1 + \#\text{ removed points})$

$O(n)$

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane..
Output: A list of vertices of $\text{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list.
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ $\triangleright$ $\mathcal{U}$ is a list containing upper-hull vertices.
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
while the last three points in $\mathcal{U}$ do not make a right turn
do remove the middle of the three points.
5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\text{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\text{CH}(P)$

Running Time:

- Step 1: $O(n \log n)$
- Step 2: $O(1)$
- Steps 3–4: $\sum_{i=3}^{n}$ time to add p_i → $O(1 + \#\text{ removed points})$
- Steps 6–7: $O(n)$

- each point is charged only once $\rightarrow O(1)$ time per point.
- total time for step $4 = O(n) +$ total time charged to points $= O(n)$

GRAHAM-SCAN(P)
Input: set $P = \{p_1, \ldots, p_n\}$ of n points in the plane.
Output: A list of vertices of $\mathrm{CH}(P)$ in clockwise order.

1: Sort P by x-coordinate. Let $p_1, \ldots, p_n$ be the sorted list. → $O(n \log n)$
2: $\mathcal{U} \leftarrow \langle p_1, p_2 \rangle$ ▷ $\mathcal{U}$ is a list containing upper-hull vertices. → $O(1)$
3: **for** $i \leftarrow 3$ to n **do**
4: Append p_i to $\mathcal{U}$.
  while the last three points in $\mathcal{U}$ do not make a right turn
    do remove the middle of the three points.

→ $\sum_{i=3}^{n}$ time to add p_i → $O(1 + \#\text{ removed points})$

5: Compute lower hull $\mathcal{L}$ in a similar way, from left to right.
6: $\mathrm{CH}(P) \leftarrow$ ($\mathcal{U}$ concatenated to $\mathcal{L}$)
7: **return** $\mathrm{CH}(P)$

→ $O(n)$

Running Time:

$O(n \log n)$
$O(1)$
$\sum_{i=3}^{n}$ time to add p_i
$O(n)$ +

$O(n \log n)$

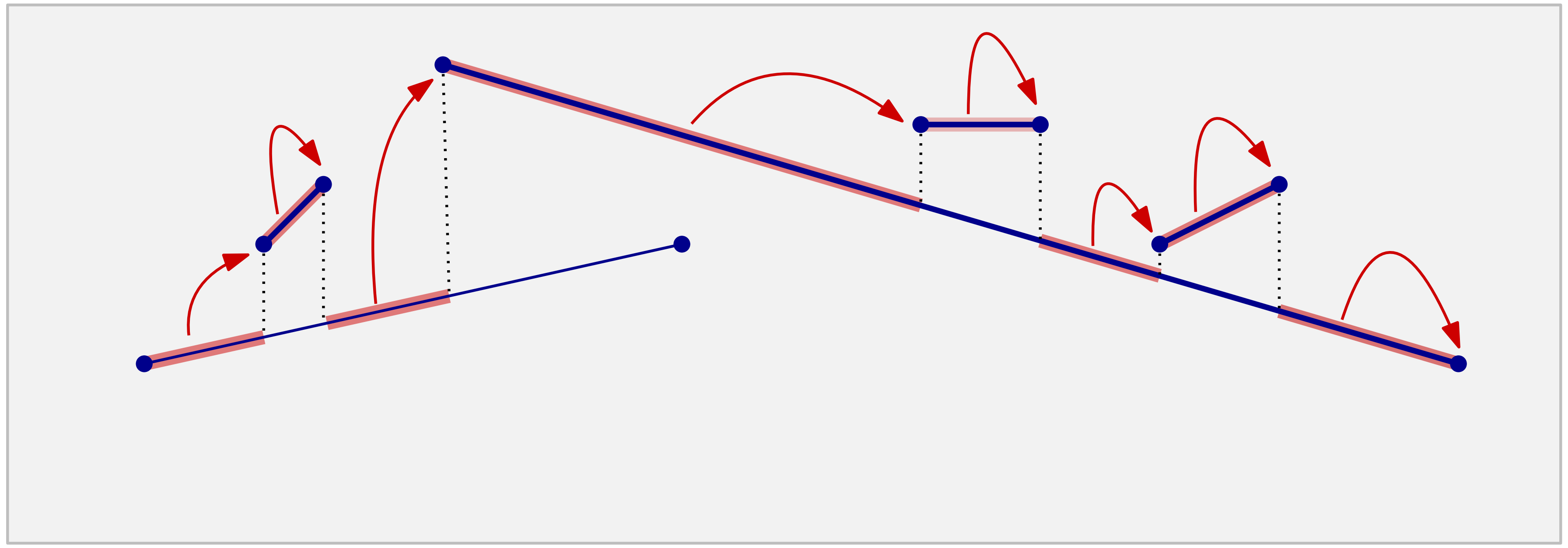
Charging Schemes:

- Suppose we want to bound the size of a set S_1 of "objects" and that we know the size of a certain set S_2 of objects (possibly of a different type).
- A charging scheme would assign (charge) each object in S_1 to an object in S_2 ...
- ...and then you need to prove an upper bound c on the number of objects in S_1 that can be charged to any single given object in S_2.
- ...so that you can conclude that $|S_1| \leqslant c \cdot |S_2|$.

Theorem. The upper envelope of a set S of n disjoint segments consists of $O(n)$ pieces.

Proof. (piece of upper envelope = maximal piece of segments in S with no other segment above it)

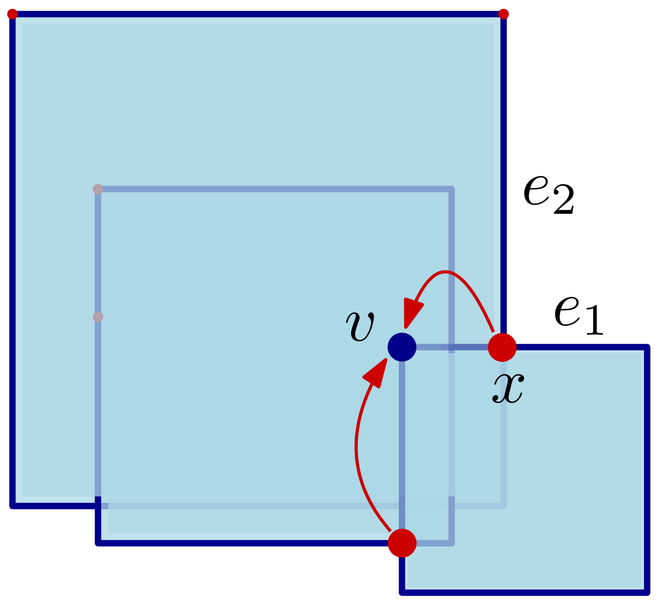
Let e be a piece of the upper envelope. If the right endpoint of e is the right endpoint r_i of a segment $s_i \in S$, then we charge e to r_i. Otherwise the right endpoint of e is immediately below the left endpoint ℓ_j of some segment $s_j \in S$, and we charge e to ℓ_j. This way every endpoint gets charged at most one piece of the upper envelope. Hence, there are at most $2n = O(n)$ pieces. ∎



Theorem. The boundary of the union of a set S of n squares has $O(n)$ vertices.

Proof.

Union vertices are of two types: corners of squares and intersection points of edges of two squares. The number of vertices of the former type is obviously at most $4n = O(n)$. To bound vertices of the latter type we use a charging scheme, as follows.



Let $x = e_1 \cap e_2$ be an intersection of edges e_1 and e_2, such that x is a vertex of the union $\bigcup S$. Let e_1 be an edge of square $\sigma_1 \in S$ and let e_2 be an edge of square $\sigma_2 \in S$. Assume without loss of generality that σ_2 is at least as large as σ_1. Then one of the endpoints of e_1 (which is a corner v of σ_1) must lies inside σ_2. We charge x to v. Note that xv lies inside the interior of σ_2 (except for its endpoint x).

We claim that any corner v of a square σ is charged at most twice, once for each edge incident to v. Indeed, If v is charged by an intersection x lying on an edge e incident to v, then the segment xv lies entirely in the interior of the union $\bigcup S$. Since e contributes to the union boundary immediately after point x, we know that for any point $y \in e$ that lies behind x the segment yv does *not* lie in the interior of $\bigcup S$. Hence, x is the only point along e that can charge v.

Since any corner is charged at most twice, we can conclude that the total number of intersection points that are a vertex of the union is at most $2 \cdot 4n = O(n)$. ∎

Summary

Theorem. The convex hull of n points in the plane can be computed in $O(n \log n)$ time and this bound is asymptotically optimal in the worst case.

Theorem. The convex hull of n points in the plane can be computed in $O(n \log n)$ time and this bound is asymptotically optimal in the worst case.

- In fact, Graham's scan computes the convex hull of n points in the plane in $O(n)$ time after sorting the points by x-coordinate.

Theorem. The convex hull of n points in the plane can be computed in $O(n \log n)$ time and this bound is asymptotically optimal in the worst case.

- In fact, Graham's scan computes the convex hull of n points in the plane in $O(n)$ time after sorting the points by x-coordinate.
- There is also an algorithm that runs in $O(nh)$ time, where h is the number of convex-hull vertices.

Theorem. The convex hull of n points in the plane can be computed in $O(n \log n)$ time and this bound is asymptotically optimal in the worst case.

- In fact, Graham's scan computes the convex hull of n points in the plane in $O(n)$ time after sorting the points by x-coordinate.
- There is also an algorithm that runs in $O(nh)$ time, where h is the number of convex-hull vertices.
- And even an algorithm that runs in $O(n \log h)$ time.