Convex Hulls (2D)

O’Rourke, Chapter 3

Announcements

- For the assignments, polygon meshes (including triangulations) are represented using indirection.

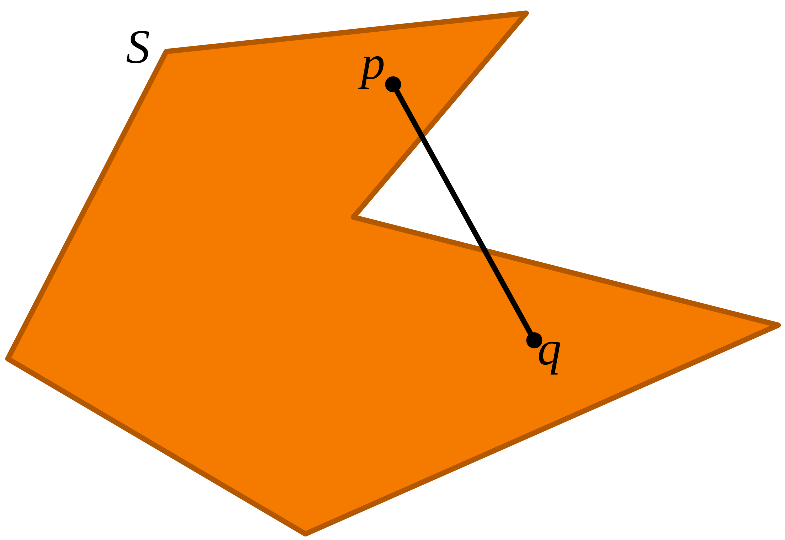
  That is, we store an array of vertices (coordinates in $\mathbb{R}^d$) and a polygon is represented as an array of indices into the vertex list.

Outline

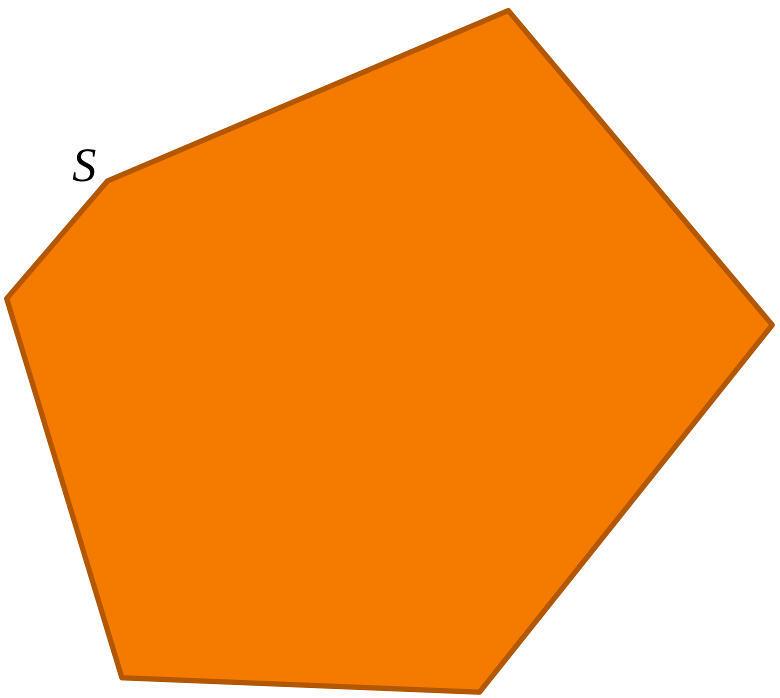
- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham’s Algorithm
  - Lower bound complexity

Convexity

A set S is *convex* if for any two points $p, q \in S$ the line segment $\overline{pq} \subset S$.



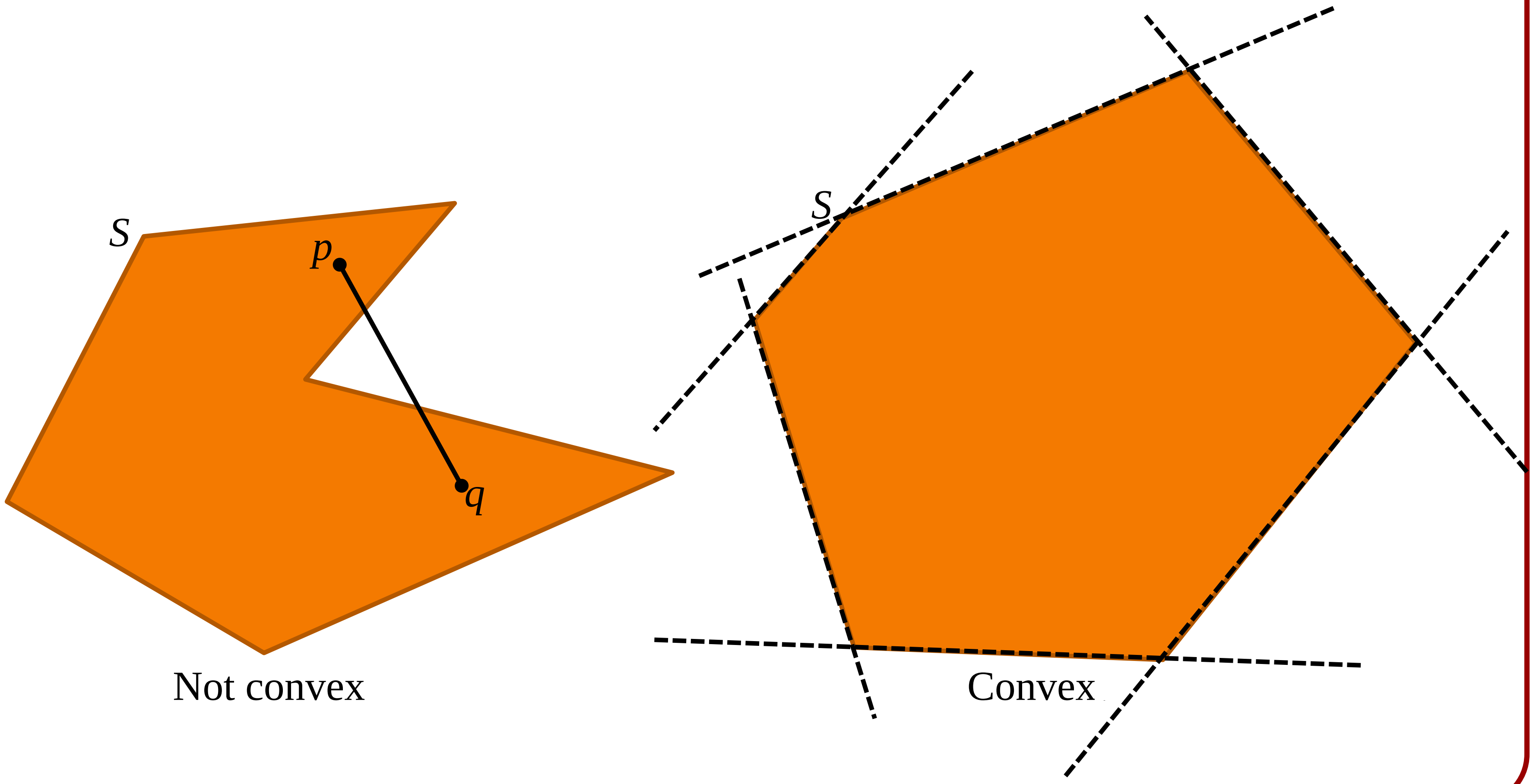
Not convex



Convex?

Convexity

A set S is *convex* if it is the intersection of (possibly infinitely many) half-spaces.



Convexity

Given points $\{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, a point $q \in \mathbb{R}^d$ is a *convex combination* of the points if q can be expressed as the linear sum:

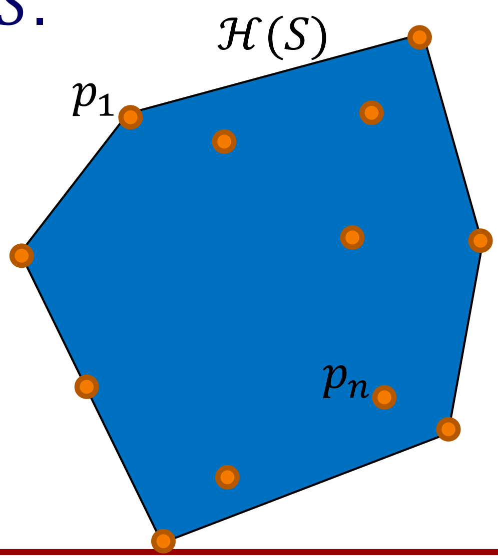
$$q = \sum_{i=1}^{n} \alpha_i \cdot p_i$$

with $\alpha_i \geq 0$ and $\alpha_1 + \cdots + \alpha_n = 1$.

Convex Hull

The *convex hull* of a set of points $S \subset \mathbb{R}^d$, denoted $\mathcal{H}(S)$, is the:

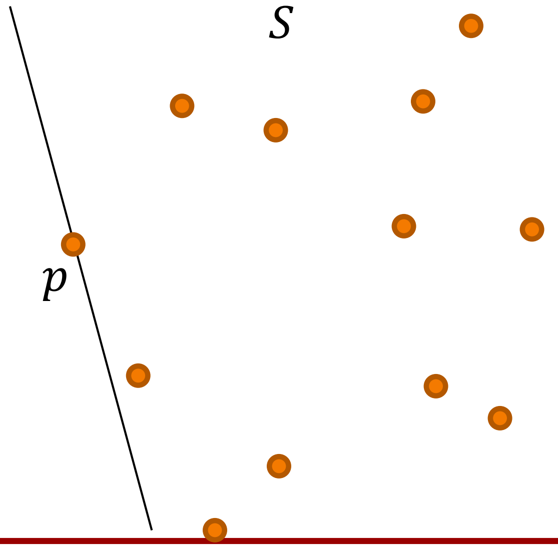
- set of all convex combinations of points in S,
- set of all convex combinations of $d + 1$ points in S,
- intersection of all convex sets C w/ $S \subset C$,
- intersection of all half-spaces H w/ $S \subset H$,
- smallest convex polygon containing S.



Convex Hull

Note:

If $p \in S \subset \mathbb{R}^2$ and p lies on a line with such that all points of S are to one side of the line, then p must be in on the boundary of the convex hull.



Convex Hull

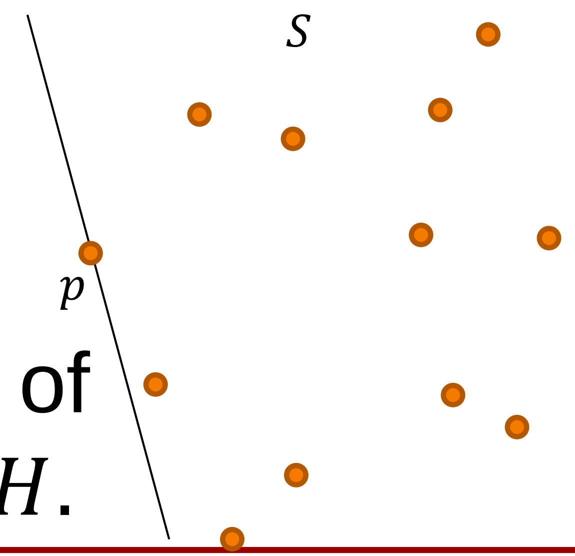
Proof:

The convex hull is the intersection of all half-spaces H w/ $S \subset H$.

Any such H must contain $p \in S$ either inside or on the boundary.

At least one such H has p on the boundary.

$\Rightarrow p$ must be on the boundary of the intersection of all such H.

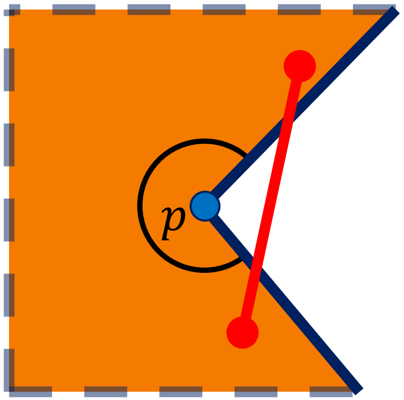


Convex Hull

Note:

If $p \in S \subset \mathbb{R}^2$ and p is a vertex of the convex hull then p must be a convex vertex.

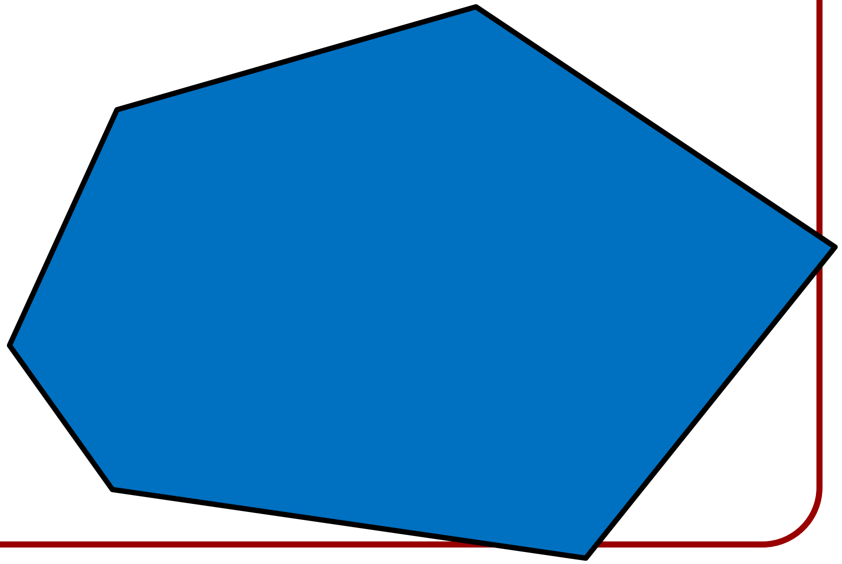
Otherwise, we could create a line segment with vertices inside of the hull but which isn't strictly interior.



Convex Hull

Claim:
If $P \subset \mathbb{R}^2$ is a polygon whose vertices are all convex, then P is convex.

We will show that if it is not convex, some vertex must be reflex.

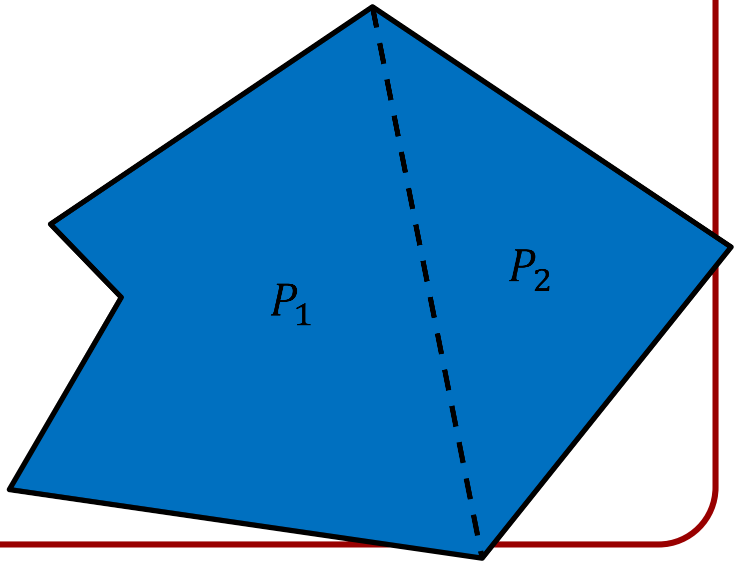


Convex Hull

Proof (by induction):

Otherwise, we could add a diagonal partitioning into sub-polygons P_1 and P_2.

⇒ By induction, each half is convex.



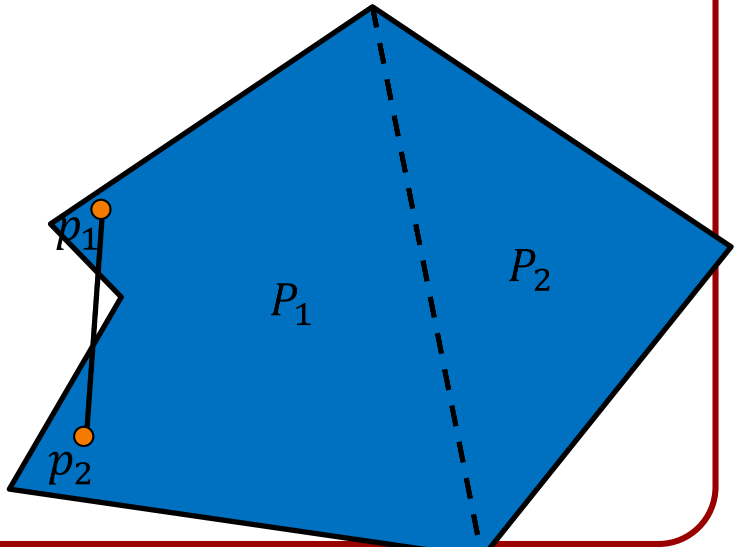
Convex Hull

Proof (by induction):

If P is not convex there must be a segment $\overline{p_1 p_2}$ between the two parts that exits P.

If p_1 and p_2 are in the same sub-polygon P_i, the line segment between them must be in the sub-polygon as well.

$\Rightarrow$ The line segment $\overline{p_1 p_2}$ cannot exit P.



Convex Hull

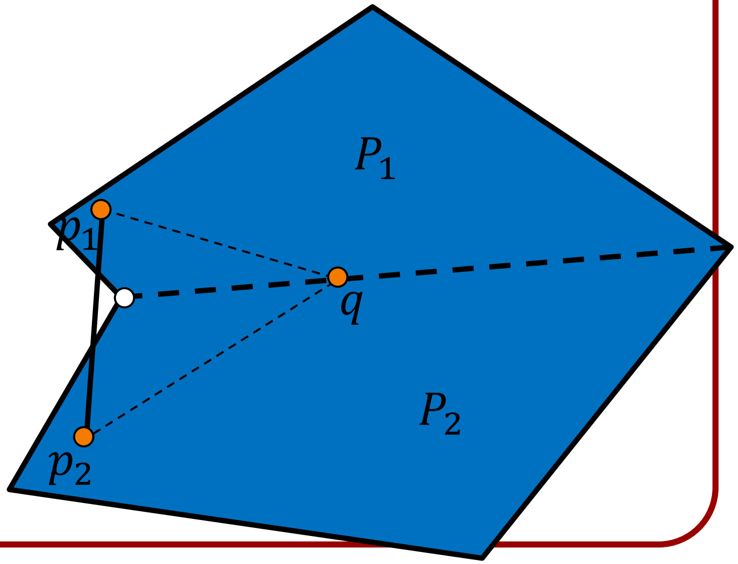
<u>Proof (by induction)</u>:

So, WLOG, $p_1 \in P_1$ and $p_2 \in P_2$.

Choose q on the diagonal.

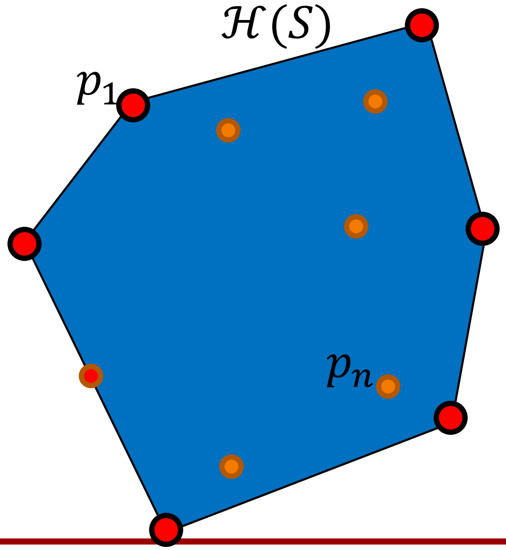
The edges $\overline{p_1 q}$ and $\overline{p_2 q}$ are entirely inside P.

$\Rightarrow$ There must be a reflex vertex inside triangle $\Delta q p_2 p_1$.



Convex Hull

The *extreme points* of a set of points $S \subset \mathbb{R}^2$ are the points which are on the convex hull and have interior angle strictly less than π.

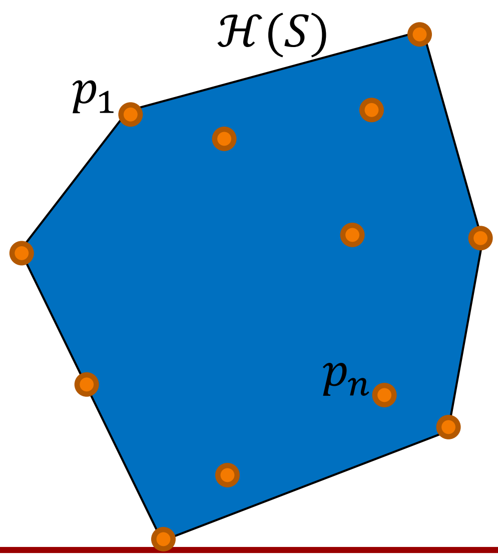


Convex Hull

Goal:

Given a set of points $S = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, compute the convex hull $\mathcal{H}(S)$ efficiently.

- Do we want all points on the hull or just the extreme ones?
- Do the output vertices need to be sorted or is the set of (extreme) vertices sufficient?

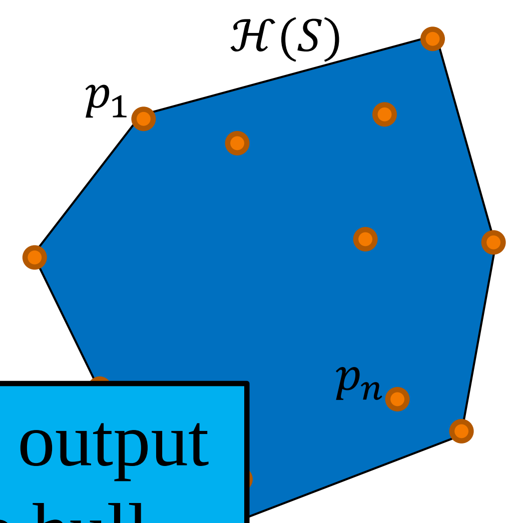


Convex Hull

<u>Goal</u>:

Given a set of points $S = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$, compute the convex hull $\mathcal{H}(S)$ efficiently.

- Do we want all points on the hull or just the extreme ones?
- Do the output vertices need to be sorted or is the set of (extreme) vertices sufficient?



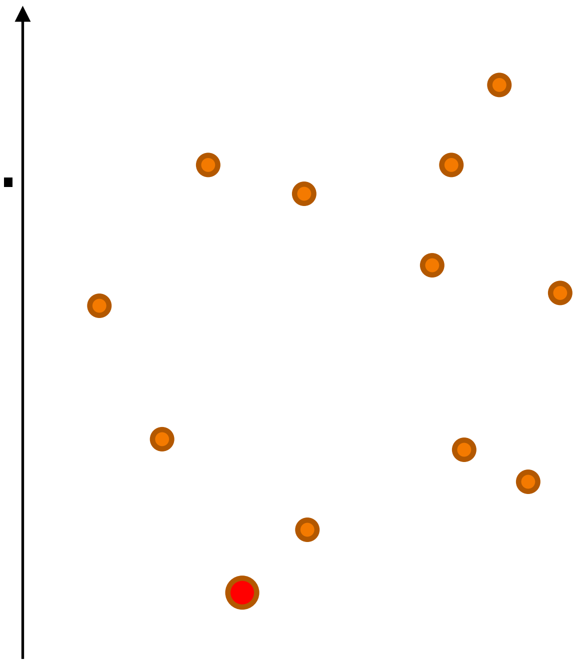
We will focus on the ordered output of the extreme points on the hull.

Convex Hull (2D)

<u>Note</u>:

If a vertex is extremal with respect to some direction, it must be on the hull.

$\Rightarrow$ We can find a hull vertex in linear time.

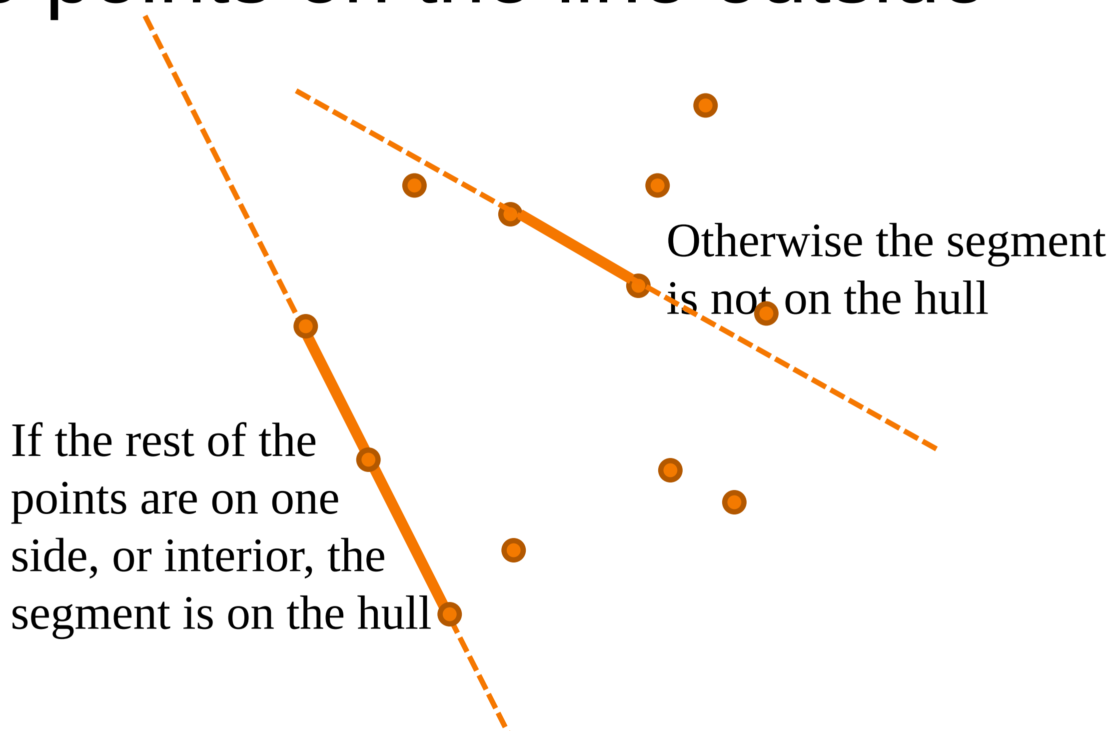


Outline

- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham's Algorithm
  - Lower bound complexity

Convex Hull (2D)

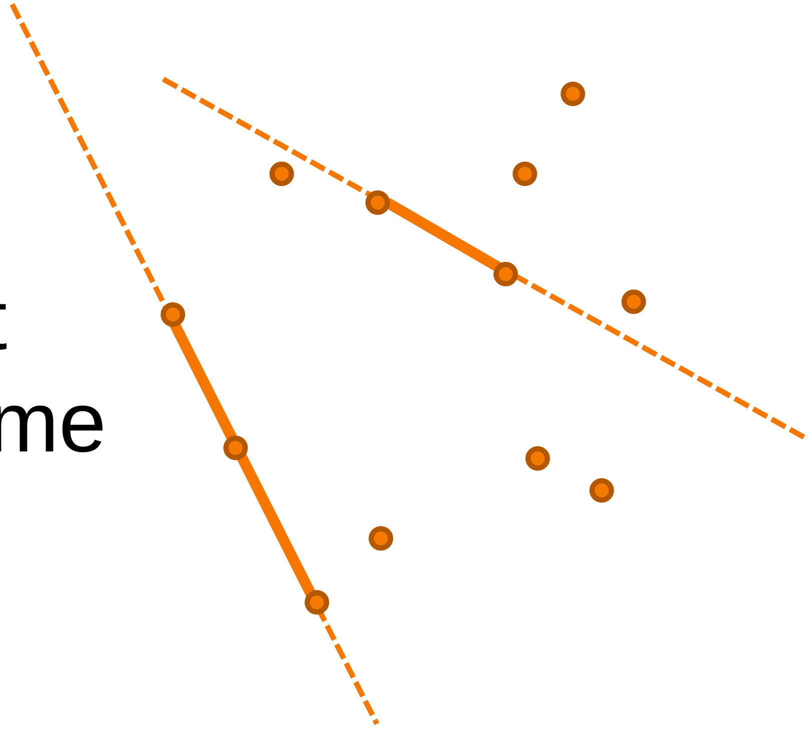
Naïve Algorithm:
For each directed edge $e \in S \times S$, check if the half-space to the right of e is empty of points (and there are no points on the line outside the segment).



Convex Hull (2D)

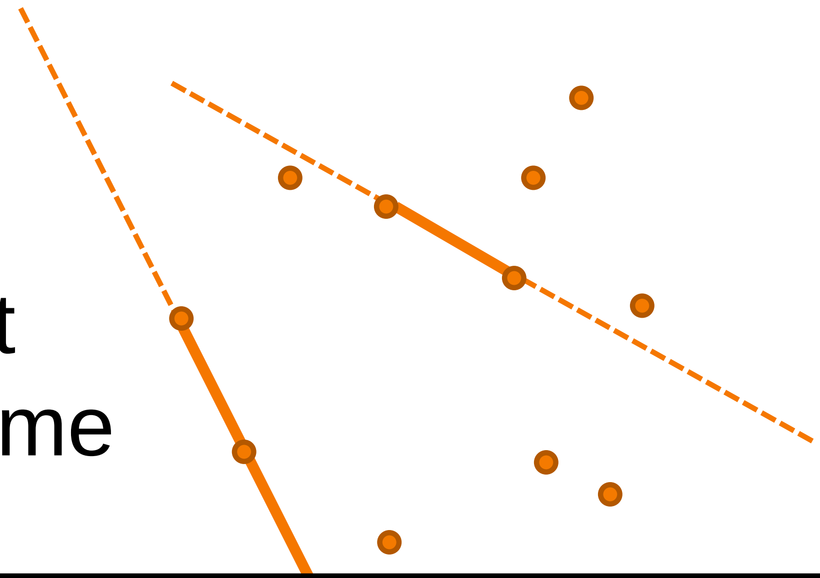
<u>Naïve Algorithm $O(n^3)$</u>:
For each directed edge $e \in S \times S$, check if the half-space to the right of e is empty of points (and there are no points on the line outside the segment).

<u>Note</u>:
The output is the set of (unordered) extreme points on the hull.



Convex Hull (2D)

<u>Naïve Algorithm $O(n^3)$</u>:
For each directed edge $e \in S \times S$, check if the half-space to the right of e is empty of points (and there are no points on the line outside the segment).



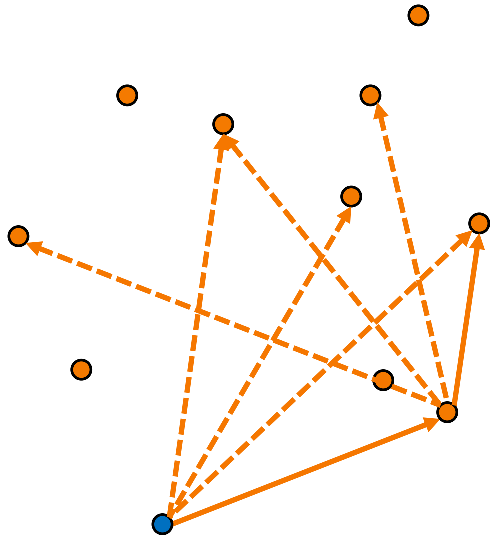
<u>Note</u>:
The output is the set of (unordered) extreme points on the hull.

If we want the ordered points, we can stitch the edges together in $\leq O(n^2)$ time in a post-processing step.

Convex Hull (2D)

Naïve Algorithm++:

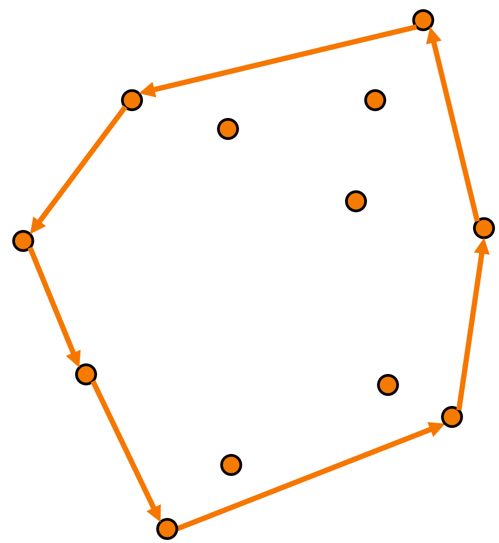
Grow the hull by starting at a hull vertex and searching for the next edge on the hull by trying all possible edges and testing if they are on the hull.



Convex Hull (2D)

Naïve Algorithm++ $O(n^2 h)^*$:
Grow the hull by starting at a hull vertex and searching for the next edge on the hull by trying all possible edges and testing if they are on the hull.

Note:
By explicitly forcing the output to be sorted, we end up with a faster algorithm.

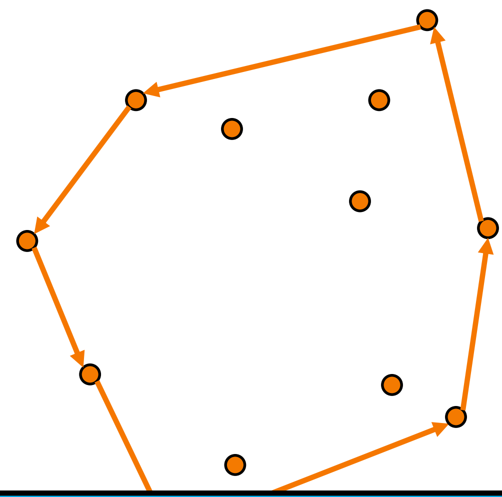


$^*h$ is the number of points on the hull.

Convex Hull (2D)

Naïve Algorithm++ $O(n^2 h)^*$:
Grow the hull by starting at a hull vertex and searching for the next edge on the hull by trying all possible edges and testing if they are on the hull.

Note:
By explicitly forcing the output to be sorted, we end up with a faster algor



This implementation is *output sensitive*.

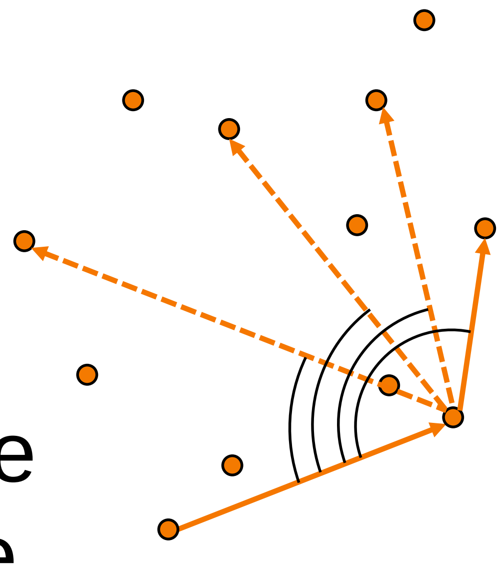
$^*h$ is the number of points on the hull.

Outline

- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham’s Algorithm
  - Lower bound complexity

Convex Hull (2D)

Note:
The next edge on the hull is the one making the largest angle. (If two points make the same angle, ignore the closer one.)

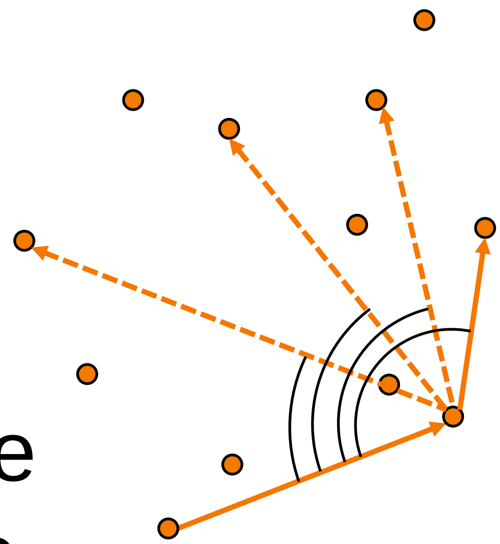


Gift Wrapping:
Grow by finding the edge making the largest angle.

Convex Hull (2D)

Note:

The next edge on the hull is the one making the largest angle. (If two points make the same angle, ignore the closer one.)

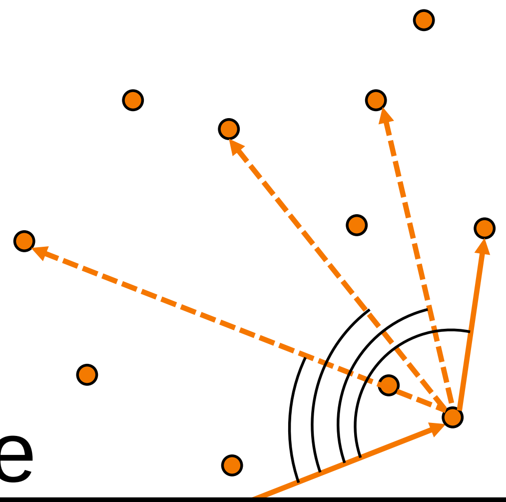


Gift Wrapping $O(nh)$:

Grow by finding the edge making the largest angle.

Convex Hull (2D)

<u>Note</u>:
The next edge on the hull is the one making the largest angle. (If two points make the same angle, ignore the closer one.)



<u>Gift Wrapping $O(nh)$</u>:
Grow by finding the edge
mak

A similar approach makes it possible to find a hull edge in linear time.

Outline

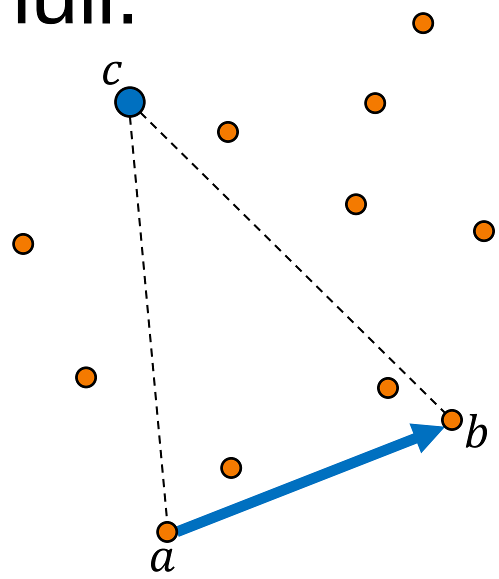
- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham's Algorithm
  - Lower bound complexity

Convex Hull (2D)

Observation:

Given a hull edge (a, b), we can find the point c furthest from the edge in linear time.

1. The point c is on the hull.
2. The triangle Δabc partitions the input into three regions:
   I. Points inside Δabc.
   II. Points to the right of $\overrightarrow{bc}$.
   III. Points to the right of $\overrightarrow{ca}$.



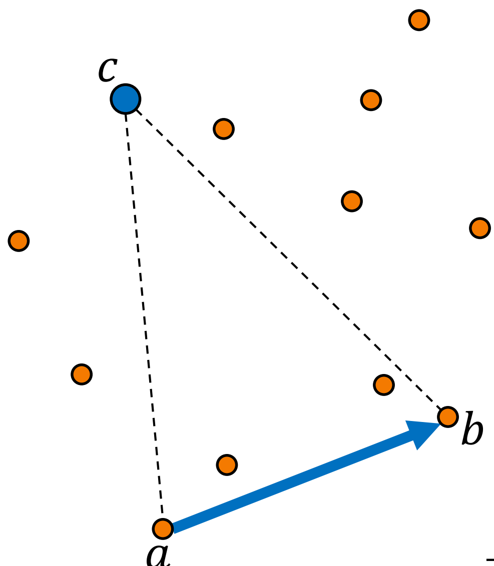
Convex Hull (2D)

Observation:

Given a hull edge (a, b), we can find the point c furthest from the edge in linear time.

$\Rightarrow$ Divide-and-conquer:

- Discard points inside Δabc
- Separately compute the half-hulls* to the right of $\overrightarrow{bc}$ and the right of $\overrightarrow{ca}$.
- Merge the two hulls.

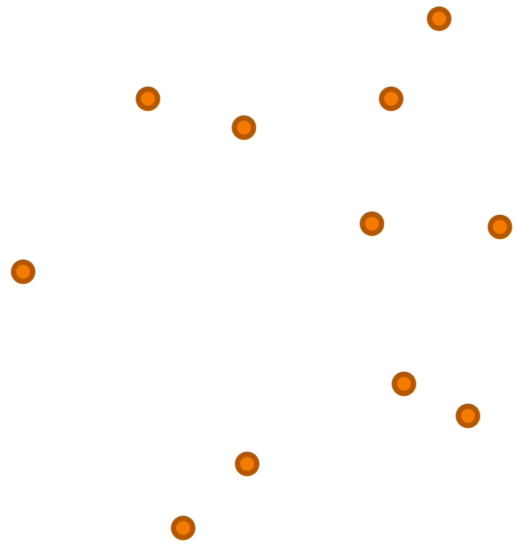


*The half hull to the right of $\overrightarrow{bc}$ is the hull of the points on or to the right of $\overrightarrow{bc}$, (i.e. including b and c) not including the edges touching either b or c.

Convex Hull (2D)

QuickHull($S \subset \mathbb{R}^2$)

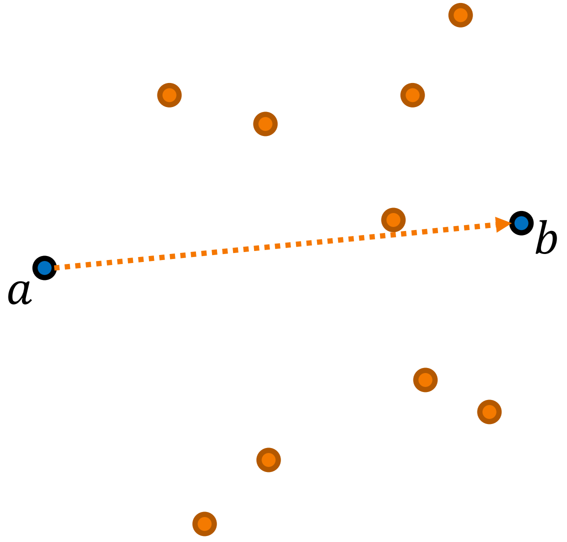
- $(a, b) \leftarrow$ HorizontalExtrema(S)
- $A \leftarrow$ Right(S , $\overrightarrow{ab}$)
- $B \leftarrow$ Right(S , $\overrightarrow{ba}$)
- $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ab}$)
- $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{ba}$)
- return $\{a\} \cup Q_A \cup \{b\} \cup Q_B$



Convex Hull (2D)

QuickHull($S \subset \mathbb{R}^2$)

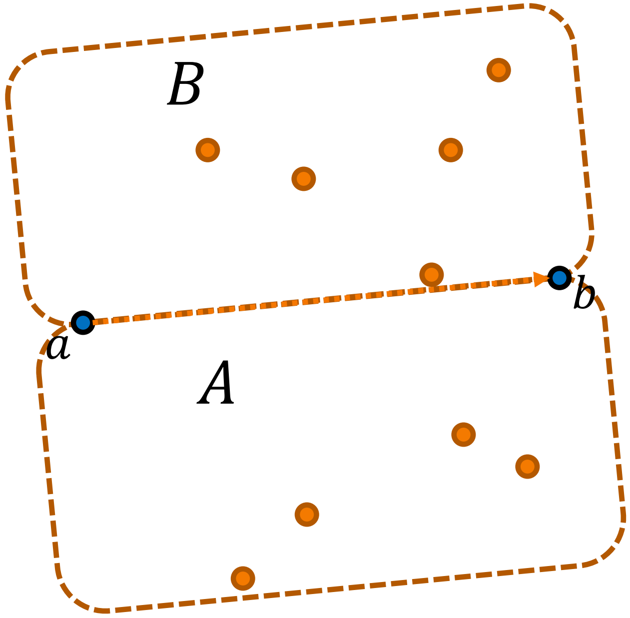
- $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow$ **HorizontalExtrema(S)**
- $A \leftarrow$ RightOn(S , $\overrightarrow{ab}$)
- $B \leftarrow$ RightOn(S , $\overrightarrow{ba}$)
- $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ab}$)
- $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{ba}$)
- return $\{a\} \cup Q_A \cup \{b\} \cup Q_B$



Convex Hull (2D)

QuickHull($S \subset \mathbb{R}^2$)

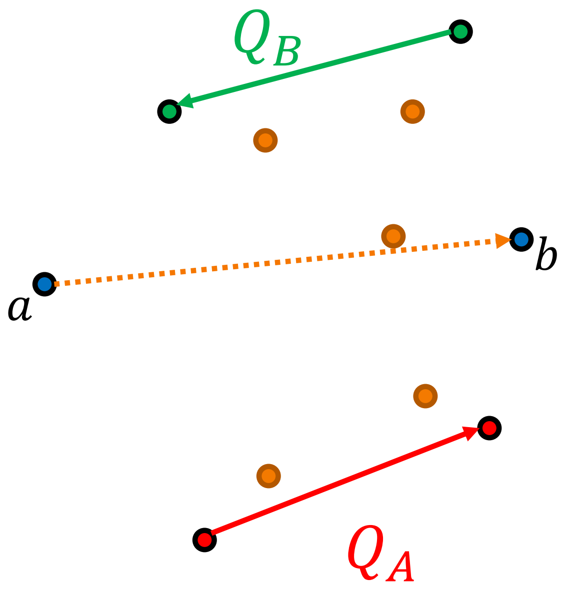
- $(a, b) \leftarrow$ HorizontalExtrema(S)
- $\boldsymbol{A} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\overrightarrow{\boldsymbol{ab}}$ **)**
- $\boldsymbol{B} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\overrightarrow{\boldsymbol{ba}}$ **)**
- $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ab}$)
- $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{ba}$)
- return $\{a\} \cup Q_A \cup \{b\} \cup Q_B$



Convex Hull (2D)

QuickHull($S \subset \mathbb{R}^2$)

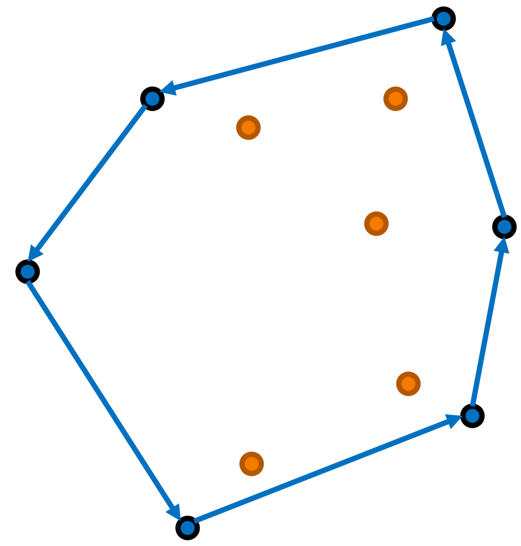
- $(a, b) \leftarrow$ HorizontalExtrema(S)
- $A \leftarrow$ RightOn(S , $\overrightarrow{ab}$)
- $B \leftarrow$ RightOn(S , $\overrightarrow{ba}$)
- $\boldsymbol{Q_A}$ $\leftarrow$ **QuickHalfHull(** $\boldsymbol{A}$ **,** $\boldsymbol{\overrightarrow{ab}}$ **)**
- $\boldsymbol{Q_B}$ $\leftarrow$ **QuickHalfHull(** $\boldsymbol{B}$ **,** $\boldsymbol{\overrightarrow{ba}}$ **)**
- return $\{a\} \cup Q_A \cup \{b\} \cup Q_B$



Convex Hull (2D)

QuickHull($S \subset \mathbb{R}^2$)

- $(a, b) \leftarrow$ HorizontalExtrema(S)
- $A \leftarrow$ RightOn(S , $\overrightarrow{ab}$)
- $B \leftarrow$ RightOn(S , $\overrightarrow{ba}$)
- $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ab}$)
- $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{ba}$)
- **return** $\{\boldsymbol{a}\} \cup \boldsymbol{Q_A} \cup \{\boldsymbol{b}\} \cup \boldsymbol{Q_B}$

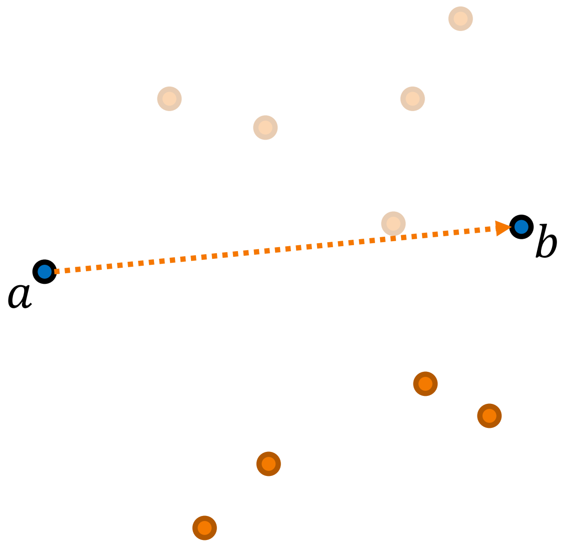


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return $\emptyset$
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOf(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOf(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

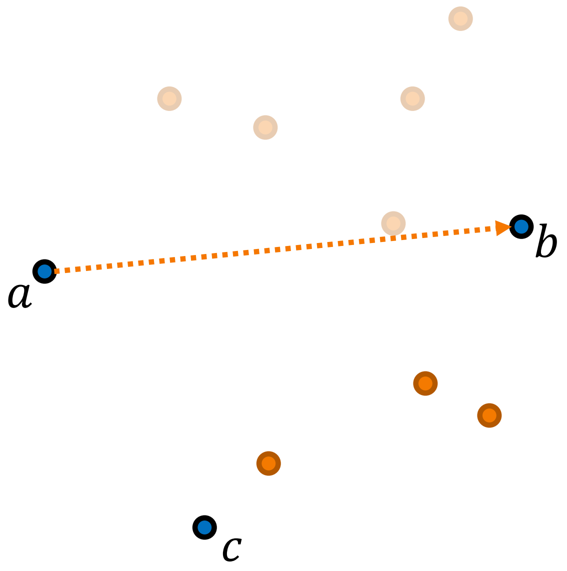


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return $\emptyset$
- else
  - » $\boldsymbol{c} \leftarrow$ **Furthest(** $\boldsymbol{S}$ **,** $\overrightarrow{\boldsymbol{ab}}$ **)**
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

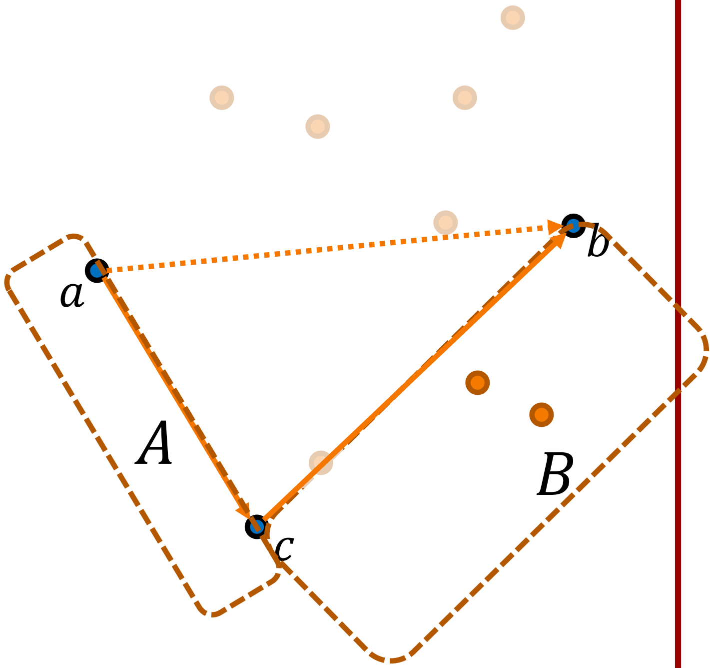


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return ∅
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $\boldsymbol{A} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\boldsymbol{\overrightarrow{ac}}$ **)**
  - » $\boldsymbol{B} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\boldsymbol{\overrightarrow{cb}}$ **)**
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

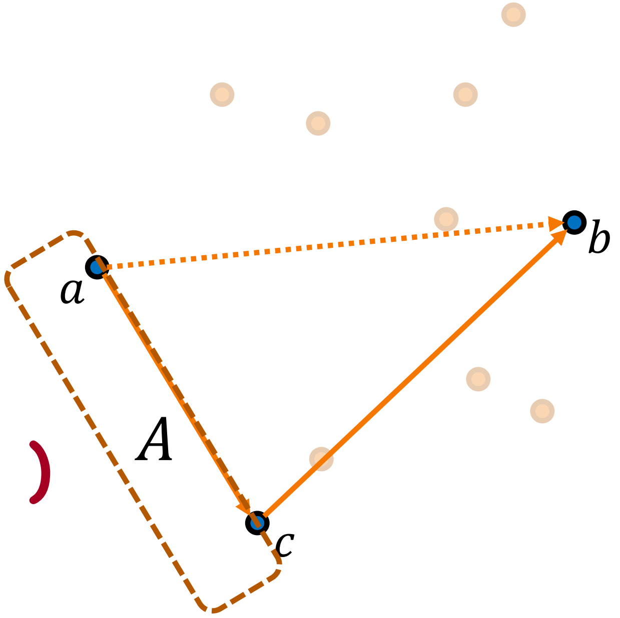


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return ∅
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » **$Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)**
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

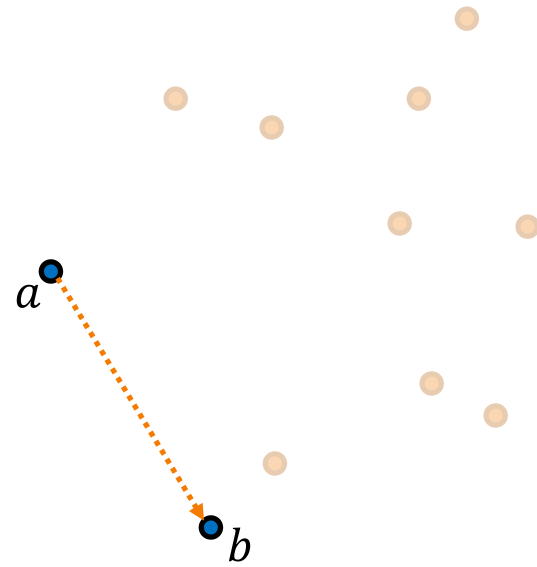


Convex Hull (2D)

(Recursion Level 1)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- **if(** $S == \{a, b\}$ **) return** $\emptyset$
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

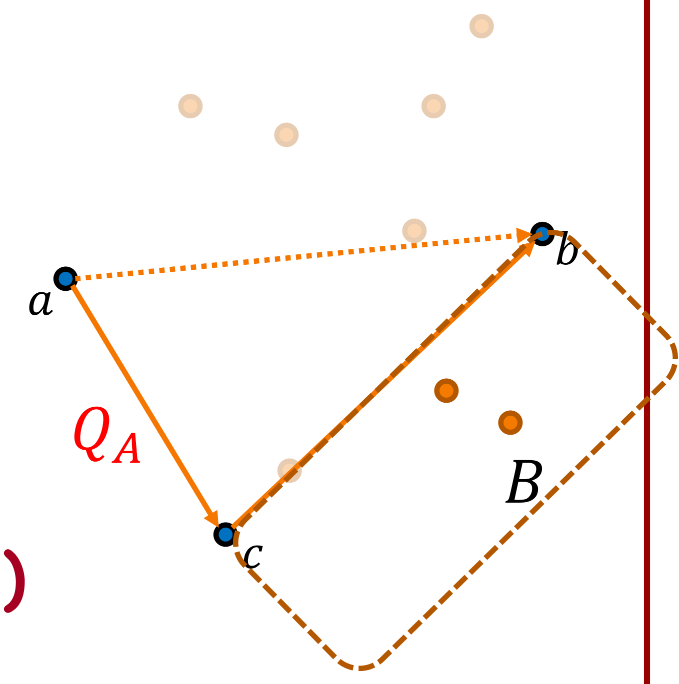


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return ∅
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » **$Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)**
  - » return $Q_A \cup \{c\} \cup Q_B$

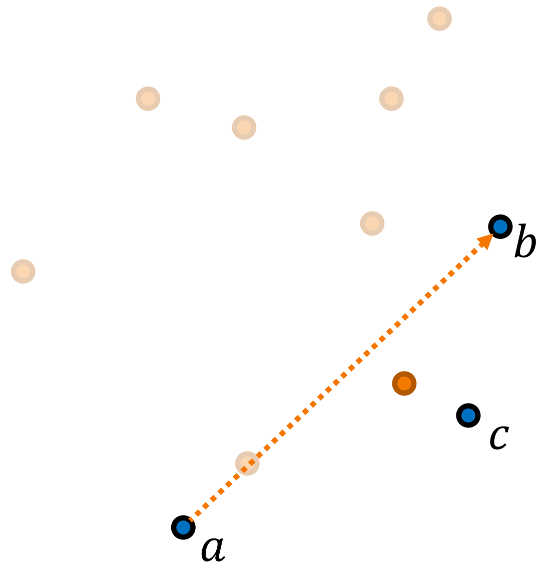


Convex Hull (2D)

(Recursion Level 1)

QuickHalfHull($S \subset \mathbb{R}^2$, $\overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return $\emptyset$
- else
  - » $\boldsymbol{c}$ ← **Furthest(** $\boldsymbol{S}$ **,** $\boldsymbol{\overrightarrow{ab}}$ **)**
  - » A ← RightOn(S , $\overrightarrow{ac}$)
  - » B ← RightOn(S , $\overrightarrow{cb}$)
  - » Q_A ← QuickHalfHull(A , $\overrightarrow{ac}$)
  - » Q_B ← QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

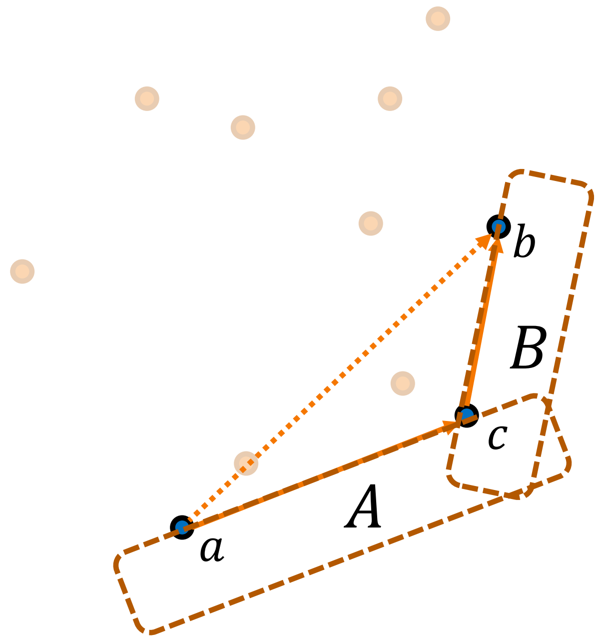


Convex Hull (2D)

(Recursion Level 1)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return ∅
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $\boldsymbol{A} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\overrightarrow{\boldsymbol{ac}}$ **)**
  - » $\boldsymbol{B} \leftarrow$ **RightOn(** $\boldsymbol{S}$ **,** $\overrightarrow{\boldsymbol{cb}}$ **)**
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » return $Q_A \cup \{c\} \cup Q_B$

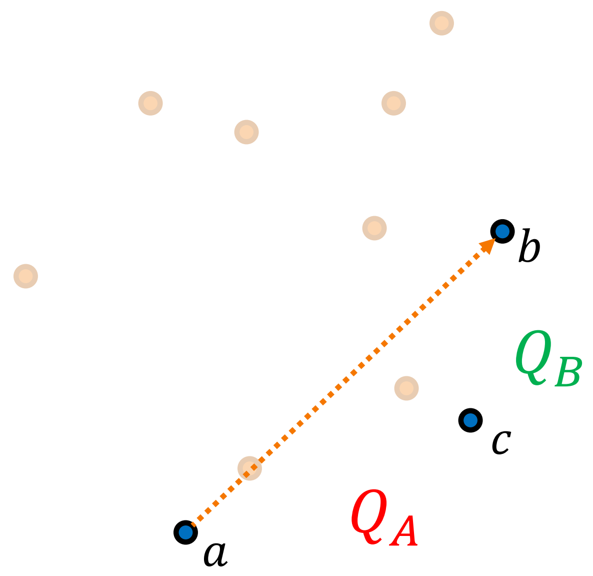


Convex Hull (2D)

(Recursion Level 1)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return ∅
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $\boldsymbol{Q_A} \leftarrow$ **QuickHalfHull(** $\boldsymbol{A}$ **,** $\boldsymbol{\overrightarrow{ac}}$ **)**
  - » $\boldsymbol{Q_B} \leftarrow$ **QuickHalfHull(** $\boldsymbol{B}$ **,** $\boldsymbol{\overrightarrow{cb}}$ **)**
  - » return $Q_A \cup \{c\} \cup Q_B$

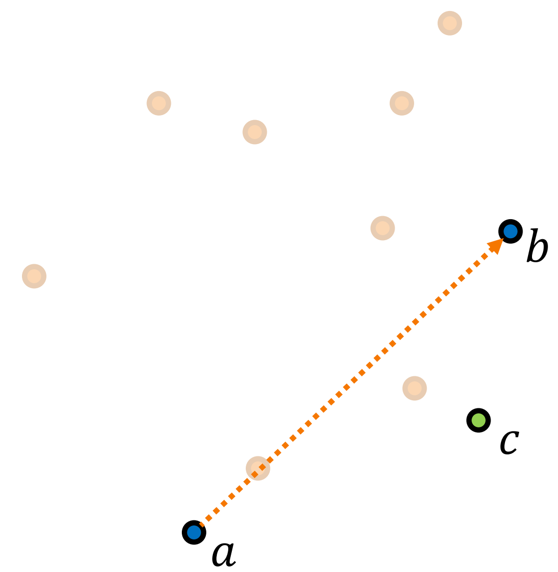


Convex Hull (2D)

(Recursion Level 1)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return Ø
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » **return $Q_A \cup \{c\} \cup Q_B$**

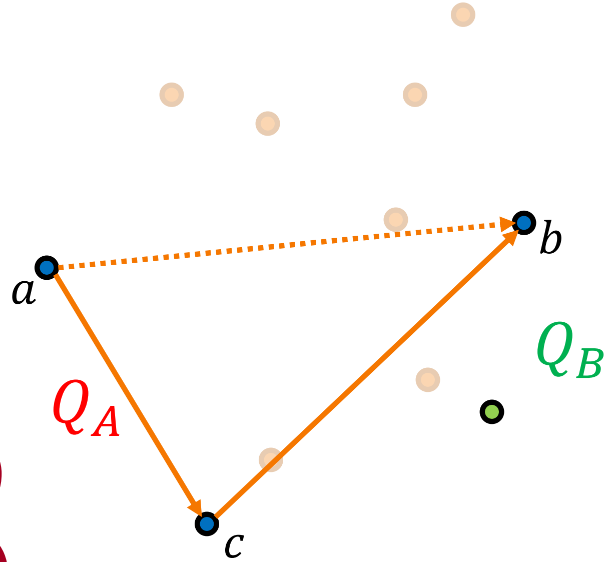


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2$, $\overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return $\emptyset$
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » **$Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)**
  - » **$Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)**
  - » return $Q_A \cup \{c\} \cup Q_B$

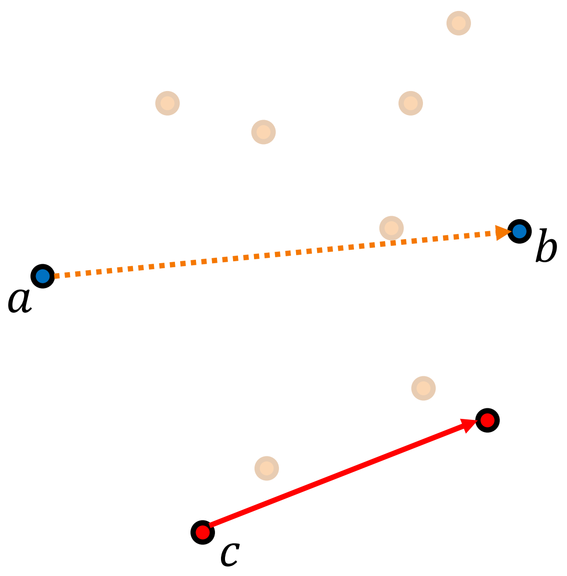


Convex Hull (2D)

(Recursion Level 0)

QuickHalfHull($S \subset \mathbb{R}^2, \overrightarrow{ab} \in S \times S$)

- if($S == \{a, b\}$) return $\emptyset$
- else
  - » $c \leftarrow$ Furthest(S , $\overrightarrow{ab}$)
  - » $A \leftarrow$ RightOn(S , $\overrightarrow{ac}$)
  - » $B \leftarrow$ RightOn(S , $\overrightarrow{cb}$)
  - » $Q_A \leftarrow$ QuickHalfHull(A , $\overrightarrow{ac}$)
  - » $Q_B \leftarrow$ QuickHalfHull(B , $\overrightarrow{cb}$)
  - » **return** $\boldsymbol{Q_A \cup \{c\} \cup Q_B}$

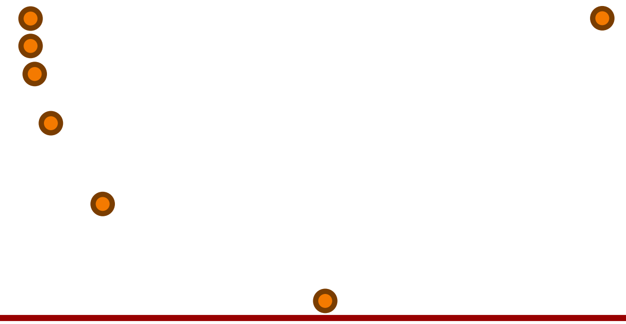


Convex Hull (2D)

QuickHull Complexity:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.

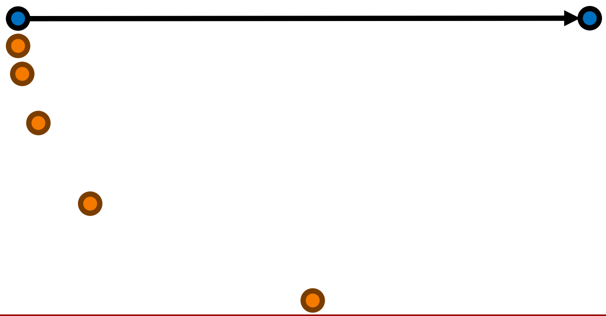


Convex Hull (2D)

QuickHull Complexity:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.

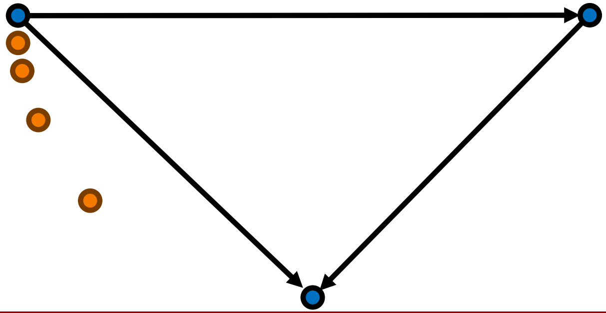


Convex Hull (2D)

<u>QuickHull Complexity</u>:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.

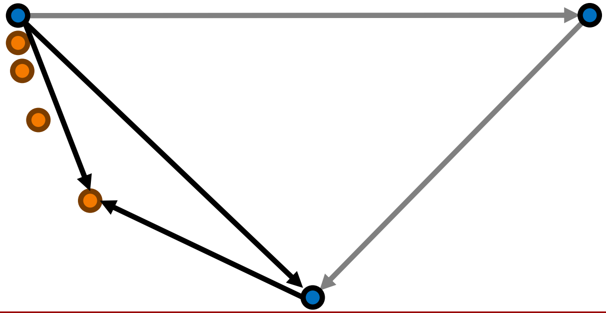


Convex Hull (2D)

<u>QuickHull Complexity</u>:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.

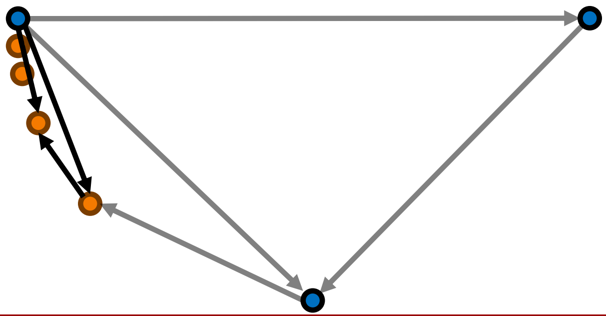


Convex Hull (2D)

QuickHull Complexity:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.



Convex Hull (2D)

QuickHull Complexity:

Like QuickSort:

- In the worst case, the complexity can be $O(n^2)$.
- In practice it is $O(n \log n)$.
- The implementation is output sensitive.

Does it extend to higher dimensions?

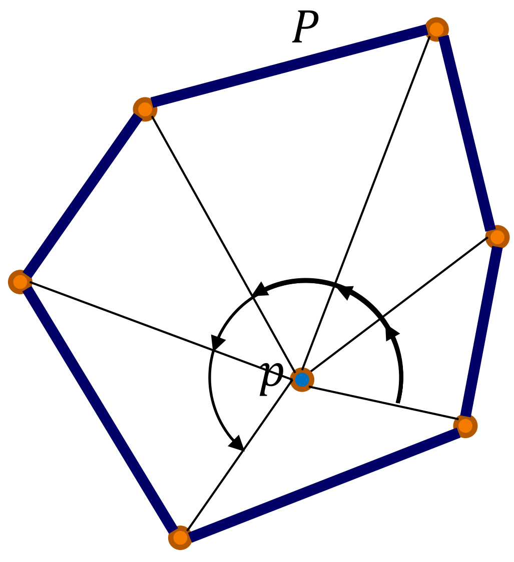
Outline

- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham's Algorithm
  - Lower bound complexity

Convex Hull (2D)

Graham's Observation:

If $P \subset \mathbb{R}^2$ is a convex polygon and $p \in P$ is a point in the interior of the polygon, then the angle of the line segments between p and the ordered vertices of P is monotonic.



Convex Hull (2D)

Graham's Observation:

WLOG assume p and v_i lie on a vertical line with p below v_i.

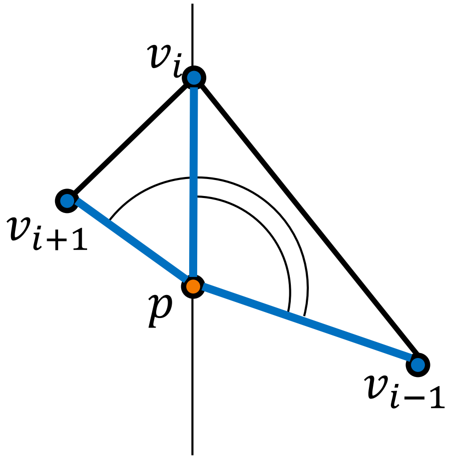
Since the polygon is convex, p is to the left of $\overrightarrow{v_i v_{i+1}}$.

$\Rightarrow v_{i+1}$ is to the left of the vertical.

Since the polygon is convex, p is to the left of $\overrightarrow{v_{i-1} v_i}$.

$\Rightarrow v_{i-1}$ is to the right of the vertical.

$\Rightarrow$ The angles $\angle p v_{i-1}$, $\angle p v_i$, $\angle p v_{i+1}$ increase monotonically.



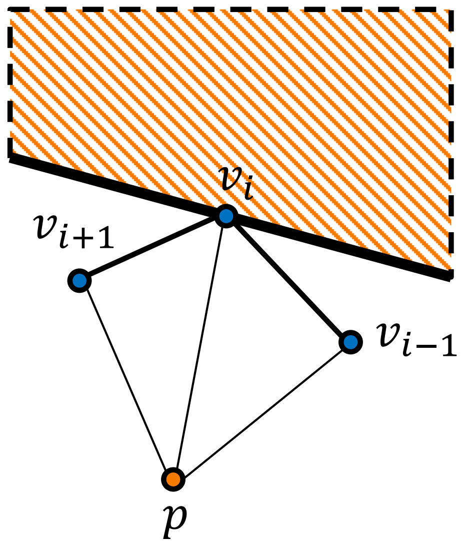
Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ PointOnHull(S)
- $H \leftarrow$ SortByAngle(S , p)
- while(RemoveReflexVertex(H)){ }
- return H

Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)
- $p \leftarrow$ PointOnHull(S)
- $H \leftarrow$ SortByAngle(S , p)
- while(RemoveReflexVertex(H)){ }
- return H



Note:

At every iteration, the vertices of H are sorted by angle relative to p.

$\Rightarrow$ Hull vertices can never be removed because the angle between the previous and next vertex is always convex.

Convex Hull (2D)

GraphamScan($S \subset \mathbb{R}^2$)

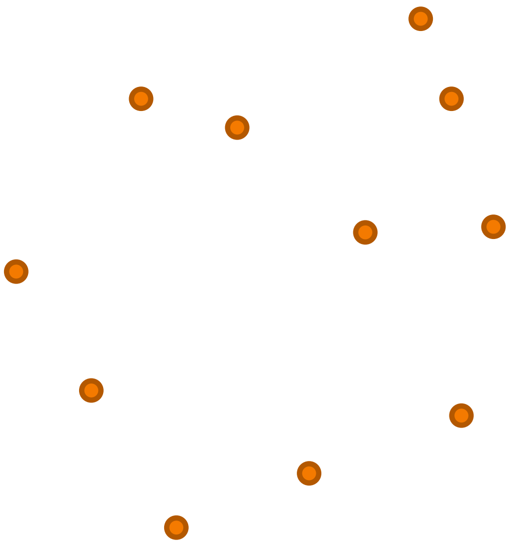
- $p \leftarrow$ PointOnHull(S)
- $H \leftarrow$ SortByAngle(S , p)
- while(RemoveReflexVertex(H)){ }
- return H

Correctness:

- The output polygon has only convex vertices.
  $\Rightarrow$ It's convex.
  $\Rightarrow H \subset \mathcal{H}(S)$.
- All hull vertices are in H.
  $\Rightarrow \mathcal{H}(S) \subset H$.

Convex Hull (2D)

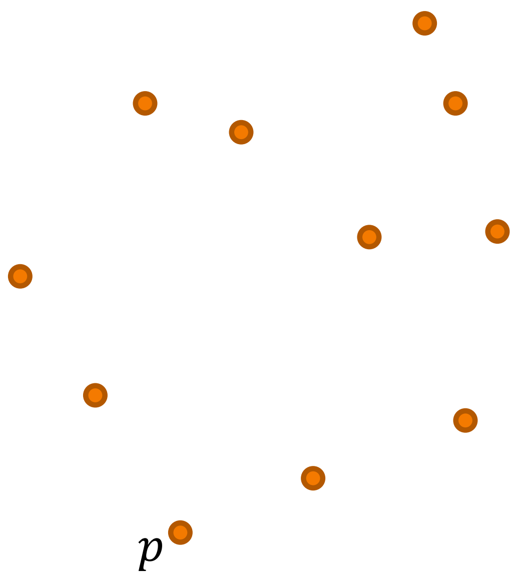
GrahamScan($S \subset \mathbb{R}^2$)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S)

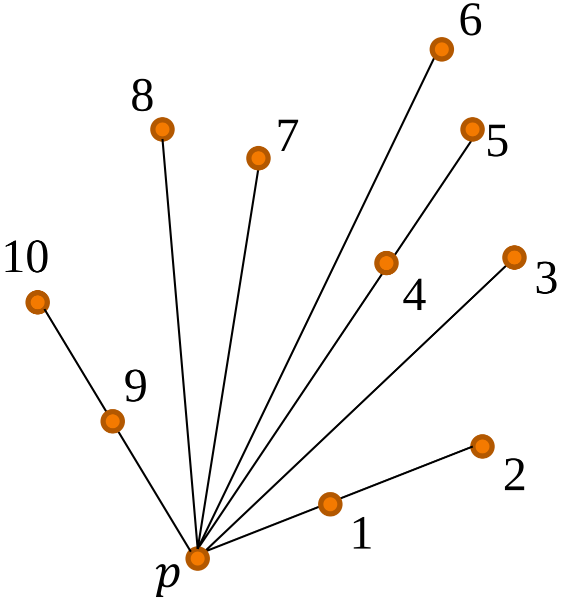


Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)

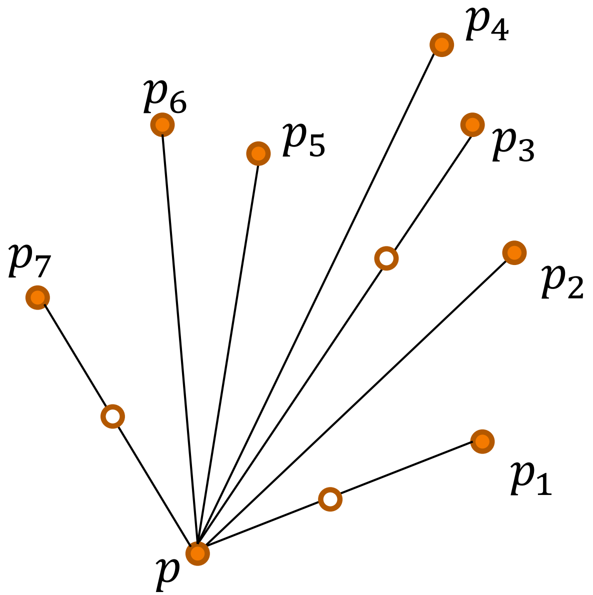
<u>Note</u>:
Since p is bottom-most, angles are in the range $[0, \pi)$ and points p_i and p_j can be sorted by testing if p_j is left of $\overrightarrow{pp_i}$.



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$

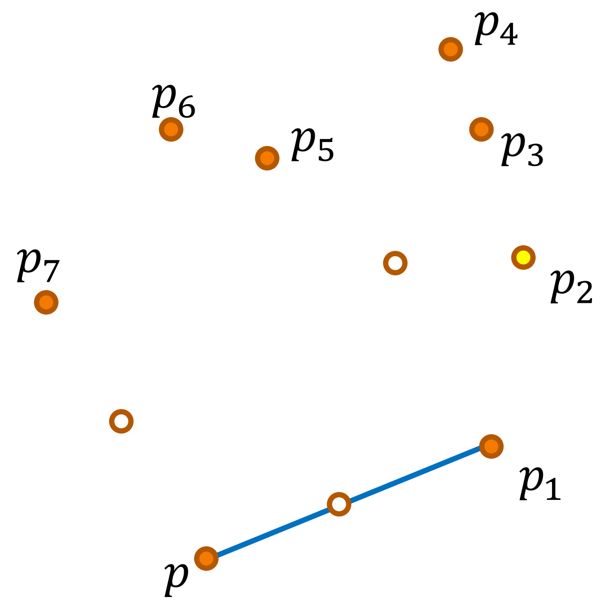


Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$

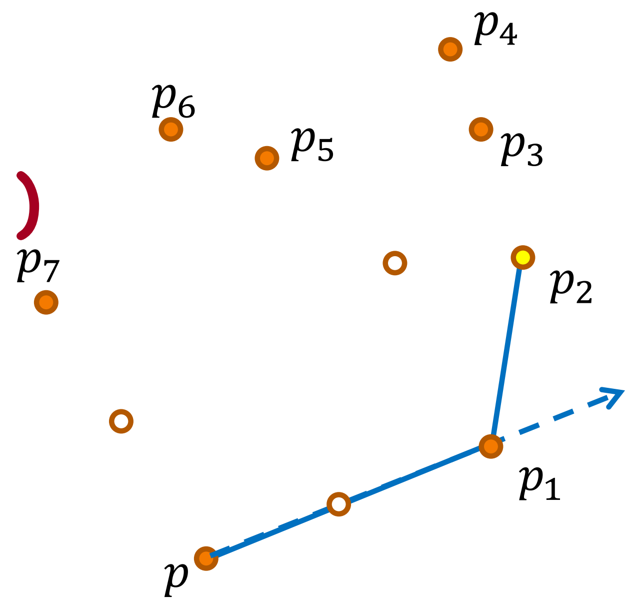
<u>Note</u>:
Since p is bottom-(right)-most, vertices are sorted by angle in $(0, \pi]$, and non-extreme points are removed, $\overline{pp_1}$ is on the hull.



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

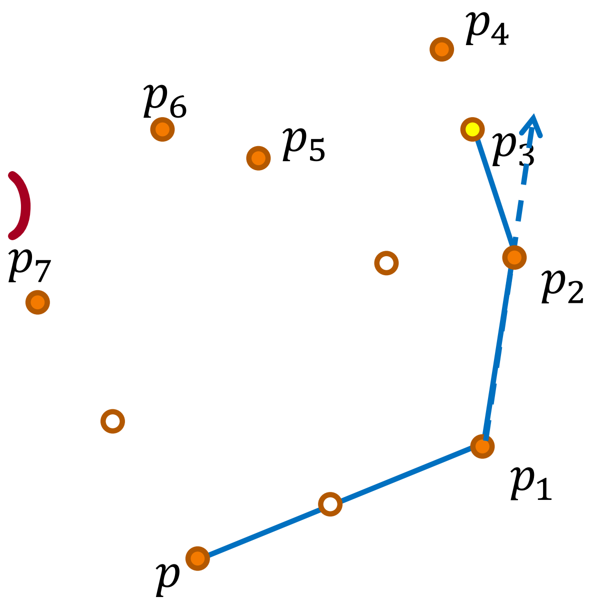
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

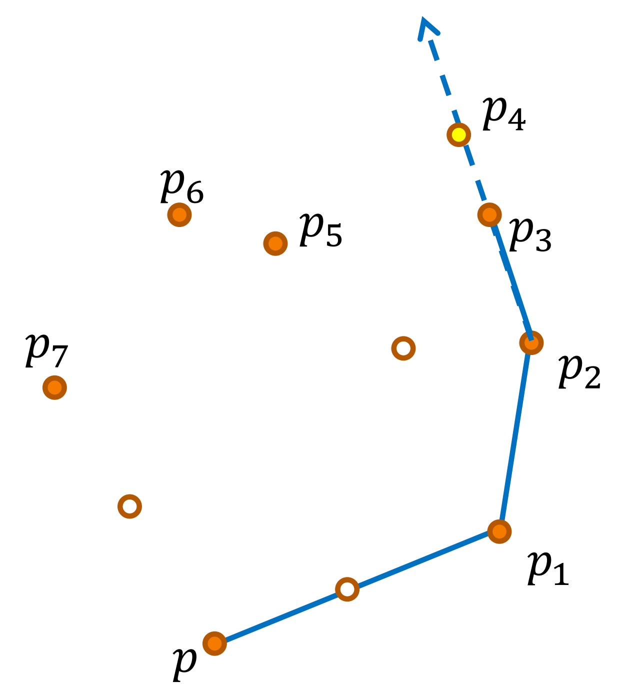
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

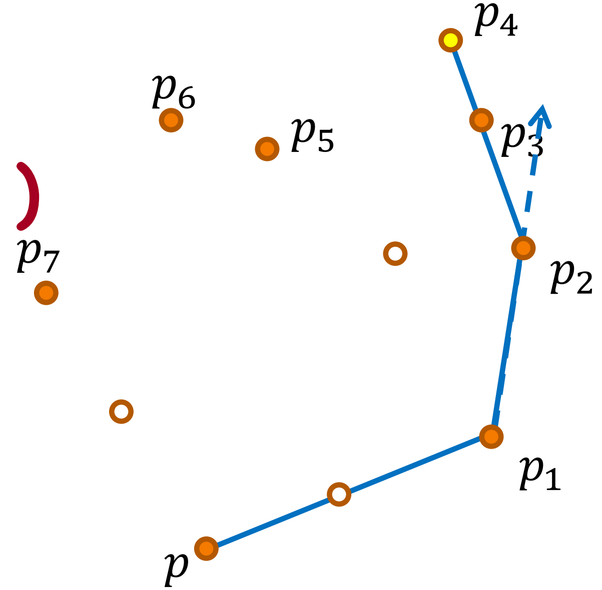
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - – Push(p_i , Q)
    - – $i \leftarrow i + 1$
  - » **else**
    - – **Pop(Q)**



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

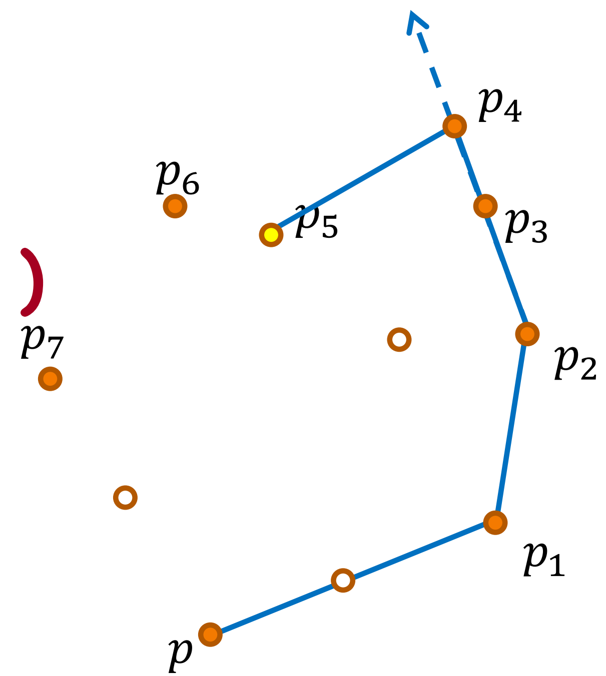
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q))
    - – Push(p_i , Q)
    - – $i \leftarrow i + 1$
  - » else
    - – Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

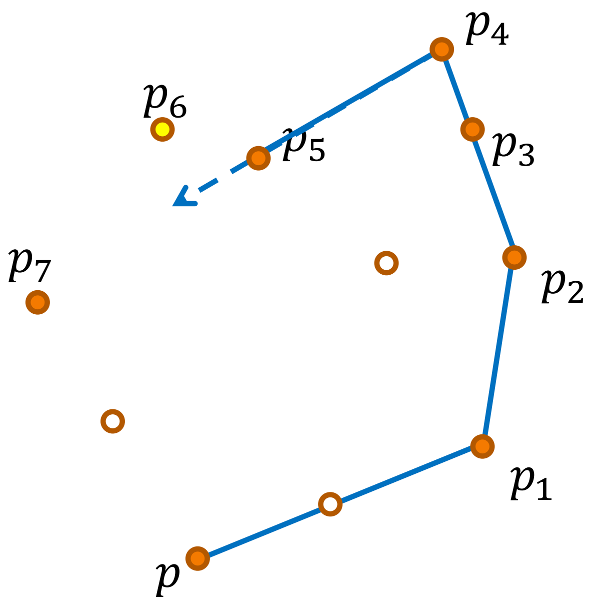
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » **if(Left(p_i , LastEdge(Q))**
    - **Push(p_i , Q)**
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

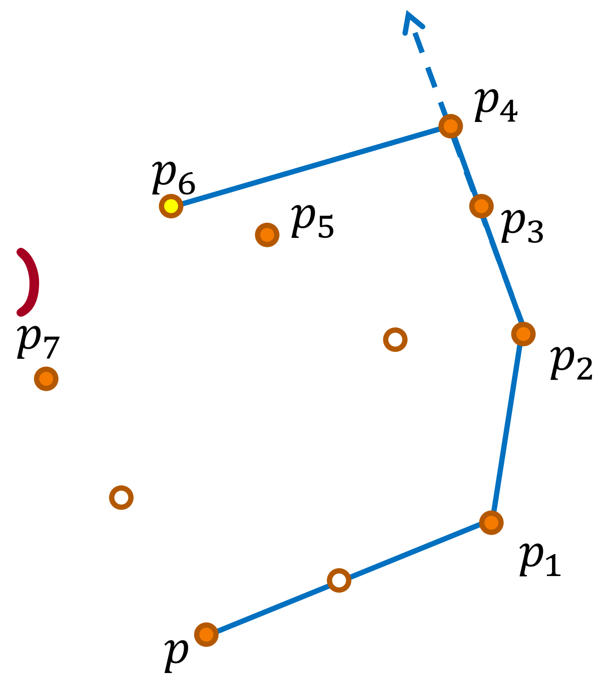
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » **else**
    - **Pop(Q)**



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

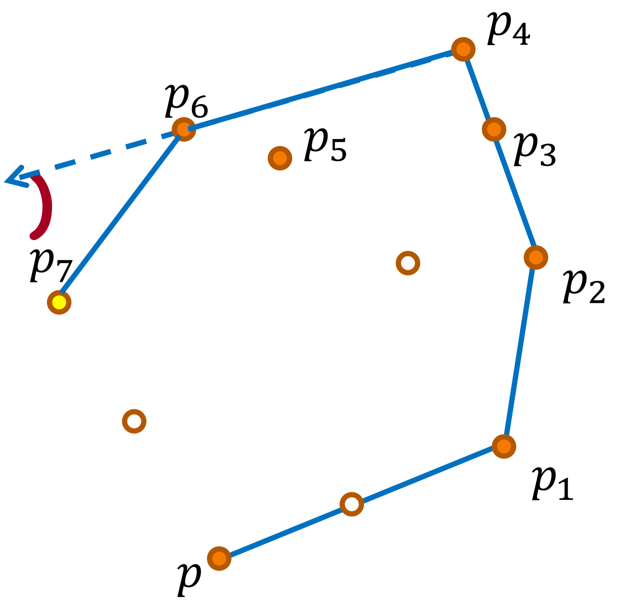
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

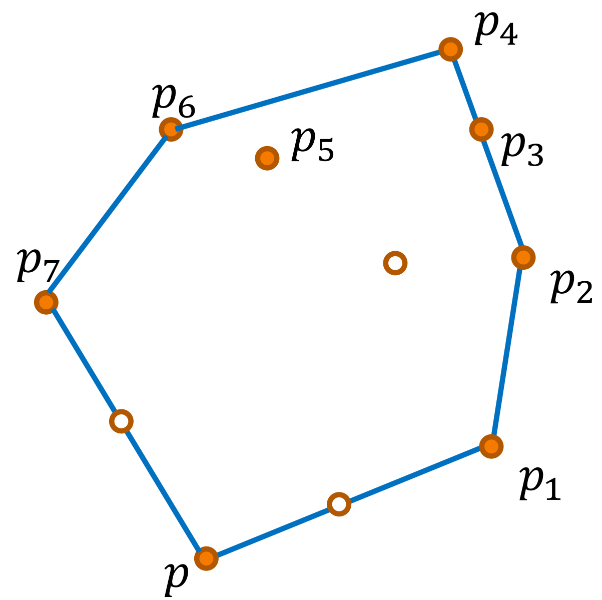
- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - – Push(p_i , Q)
    - – $i \leftarrow i + 1$
  - » else
    - – Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S)
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$)
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)



Convex Hull (2D)

GrahamScan($S \subset \mathbb{R}^2$)

- $p \leftarrow$ BottommostRightmost(S) $O(n)$
- $\tilde{S} \leftarrow$ SortByAngleAndLength(p , $S - \{p\}$) $O(n \log n)$
- if(angle(p_i) == angle(p_{i+1})): $\tilde{S} \leftarrow \tilde{S} - \{p_i\}$ $O(n)$
- $i \leftarrow 2$
- $Q \leftarrow \{p, p_1\}$
- while($i < |\tilde{S}|$)
  - » if(Left(p_i , LastEdge(Q)))
    - Push(p_i , Q)
    - $i \leftarrow i + 1$
  - » else
    - Pop(Q)

$O(n)$

Outline

- Convex Hulls
- Algorithms
  - Naïve Implementation(s)
  - Gift Wrapping
  - Quick Hull
  - Graham’s Algorithm
  - Lower bound complexity

Lower Bound Complexity

Recall:

Sorting n numbers has lower bound complexity $O(n \log n)$.

Approach:

We will show that computing the 2D hull has the same complexity by reducing sorting to the problem of computing the convex hull.

Lower Bound Complexity

Sorting → Convex Hull Reduction (Shamos):

Given a set of points $\{x_i\} \subset \mathbb{R}$:

- Choose a function $f(x)$ w/ $f''(x) > 0$
- Lift the points onto the curve
- Compute the convex hull
- Return the points on the lower hull, starting w/ the left-most.