Computing Convex Hull (in 2D)

1389-2

Convex Set

Definition:

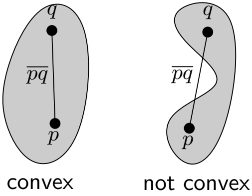
A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S.

Convex Hull:

The convex hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.

Definition:

A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S.



Convex Hull:

The convex hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.

Convex Set

Definition:

A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S.

Convex Hull:

The convex hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.

Computing CH:

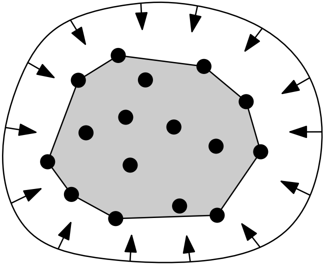
Observation:

It is the unique convex polygon whose vertices are points from P and that contains all points of P.

Computing CH:

Observation:

It is the unique convex polygon whose vertices are points from P and that contains all points of P.



Computing CH:

Problem:

given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points in the plane, compute a list that contains those points from P that are the vertices of $\mathcal{CH}(P)$, listed in clockwise order.

Problem:

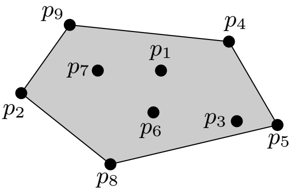
given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points in the plane, compute a list that contains those points from P that are the vertices of $\mathcal{CH}(P)$, listed in clockwise order.

Input= set of points

$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

Output= representation of the convex hull:

p_4, p_5, p_8, p_2, p_9



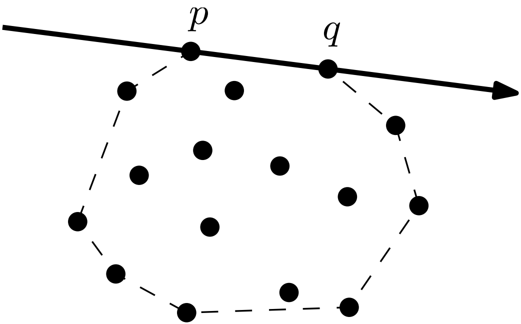
Geometry of the problem

Property:

If we direct the line through p and q such that $\mathcal{CH}(P)$ lies to the right, then all the points of P must lie to the right of this line. The reverse is also true: if all points of $P \setminus \{p, q\}$ lie to the right of the directed line through p and q, then pq is an edge of $\mathcal{CH}(P)$.

Property:

If we direct the line through p and q such that $\mathcal{CH}(P)$ lies to the right, then all the points of P must lie to the right of this line. The reverse is also true: if all points of $P \setminus \{p, q\}$ lie to the right of the directed line through p and q, then pq is an edge of $\mathcal{CH}(P)$.



First algorithm

Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
E ← ∅.
for all ordered pairs (p,q) ∈ P × P with p not equal to q
    do valid ← true
        for all points r ∈ P not equal to p or q
            do if r lies to the left of the directed line from p to q
                then valid ← false.
        if valid then Add the directed edge pq→ to E.
From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise order.
```

Clarify:

1. How do we test whether a point lies to the left or to the right of a directed line? (See Exercise 1.4)
2. How can we construct $\mathcal{L}$ from E?

First algorithm

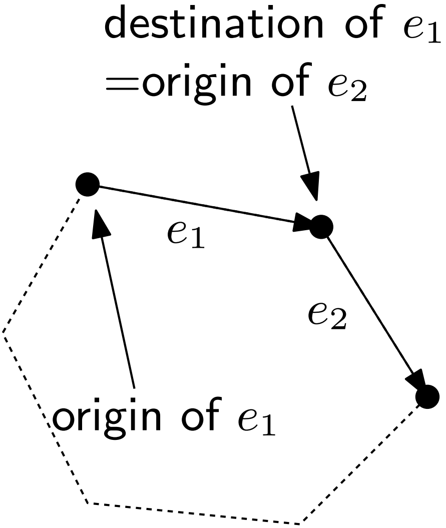
Algorithm SlowConvexHull(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
E ← ∅.
for all ordered pairs (p,q) ∈ P × P with p not equal to q
    do valid ← true
        for all points r ∈ P not equal to p or q
            do if r lies to the left of the directed line from p to q
                then valid ← false.
        if valid then Add the directed edge pq→ to E.
From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise ord
```

Clarify:

1. How do we test whether a point lies to the left or to the right of a directed line? (See Exercise 1.4)
2. How can we construct $\mathcal{L}$ from E?

Computing $\mathcal{L}$:



Complexity of the algorithm

Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

$E \leftarrow \emptyset$.
for all ordered pairs $(p,q) \in P \times P$ with p not equal to q
    do *valid* $\leftarrow$ **true**
        for all points $r \in P$ not equal to p or q
            do if r lies to the left of the directed line from p to q
                then *valid* $\leftarrow$ **false**.
        if *valid* **then** Add the directed edge $\overrightarrow{pq}$ to E.
From the set E of edges construct a list $\mathcal{L}$ of vertices of $\mathcal{CH}(P)$, sorted in clockwise ord

Running time: $\mathcal{O}(n^3) + \mathcal{O}(n^2) = \mathcal{O}(n^3)$.

Complexity of the algorithm

Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

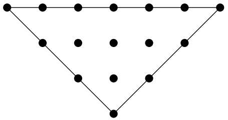
```
E ← ∅.
for all ordered pairs (p, q) ∈ P × P with p not equal to q
    do valid ← true
        for all points r ∈ P not equal to p or q
            do if r lies to the left of the directed line from p to q
                then valid ← false.
        if valid then Add the directed edge pq→ to E.
From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise ord
```

Running time: $\mathcal{O}(n^3) + \mathcal{O}(n^2) = \mathcal{O}(n^3)$.

Degenerate case (or Degeneracy)

Degenerate Case:

A point r does not always lie to the right or to the left of the line through p and q, it can also happen that it **lies on** this line. What should we do then?



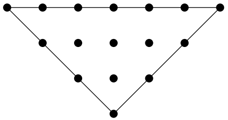
Solution:

A directed edge $\overrightarrow{pq}$ is an edge of $\mathcal{CH}(P)$ if and only if all other points $r \in P$ lie either strictly to the right of the directed line through p and q, or they lie on the open line segment $\overline{pq}$.

Degenerate case (or Degeneracy)

Degenerate Case:

A point r does not always lie to the right or to the left of the line through p and q, it can also happen that it **lies on** this line. What should we do then?



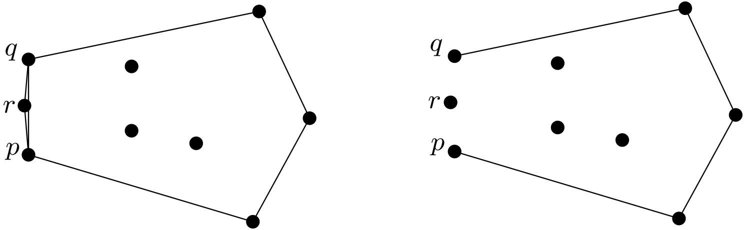
Solution:

A directed edge $\overrightarrow{pq}$ is an edge of $\mathcal{CH}(P)$ if and only if all other points $r \in P$ lie either strictly to the right of the directed line through p and q, or they lie on the open line segment $\overline{pq}$.

Robustness:

Robustness:

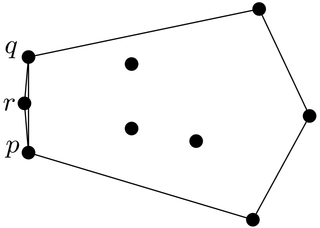
If the points are given in floating point coordinates and the computations are done using floating point arithmetic, then there will be rounding errors that may distort the outcome of tests.

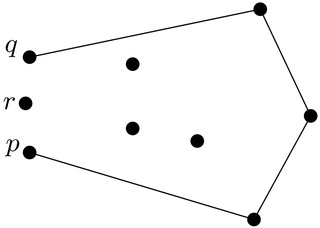


This algorithm is not robust!

Robustness:

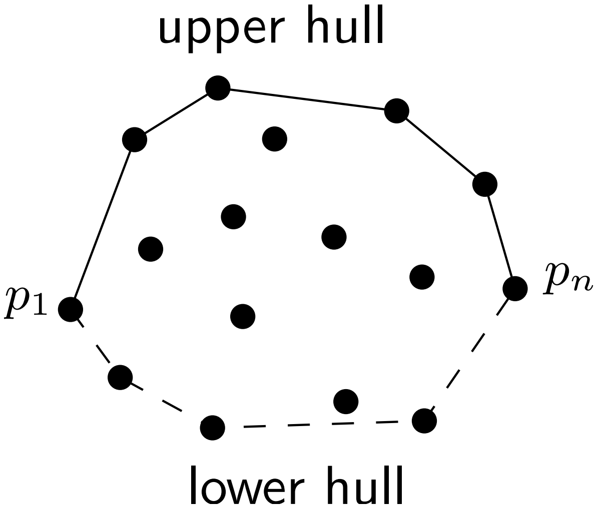
If the points are given in floating point coordinates and the computations are done using floating point arithmetic, then there will be rounding errors that may distort the outcome of tests.

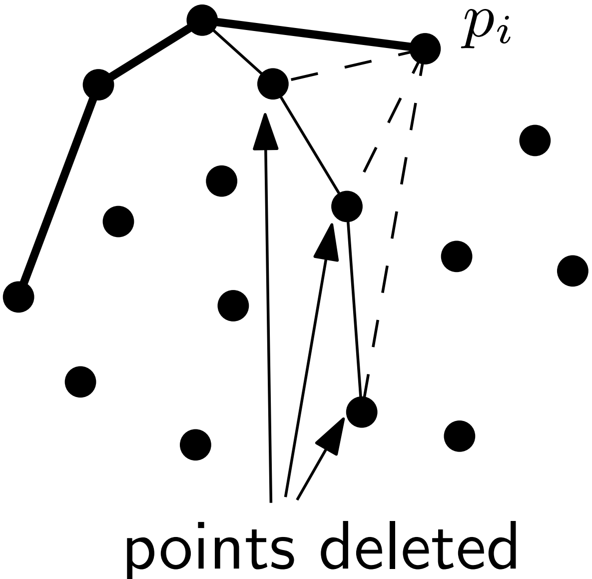




This algorithm is not robust!

2nd algorithm: incremental




p_i
points deleted

2nd algorithm: incremental

Algorithm CONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

1. Sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$.
2. Put the points p_1 and p_2 in a list $\mathcal{L}_{\text{upper}}$, with p_1 as the first point.
3. **for** $i \leftarrow 3$ **to** n
4.   **do** Append p_i to $\mathcal{L}_{\text{upper}}$.
5.   **while** $\mathcal{L}_{\text{upper}}$ contains more than two points **and** the last three points in $\mathcal{L}_{\text{upper}}$ do not make a right turn
6.   **do** Delete the middle of the last three points from $\mathcal{L}_{\text{upper}}$.
7. Put the points p_n and p_{n-1} in a list $\mathcal{L}_{\text{lower}}$, with p_n as the first point.
8. **for** $i \leftarrow n-2$ **downto** 1
9.   **do** Append p_i to $\mathcal{L}_{\text{lower}}$.
10.   **while** $\mathcal{L}_{\text{lower}}$ contains more than 2 points **and** the last three points in $\mathcal{L}_{\text{lower}}$ do not make a right turn
11.   **do** Delete the middle of the last three points from $\mathcal{L}_{\text{lower}}$.
12. Remove the first and the last point from $\mathcal{L}_{\text{lower}}$ to avoid duplication of the points where the upper and lower hull meet.
13. Append $\mathcal{L}_{\text{lower}}$ to $\mathcal{L}_{\text{upper}}$, and call the resulting list $\mathcal{L}$.
14. **return** $\mathcal{L}$

2nd algorithm: incremental

Special cases:

1. Two points have same x-coordinate.
2. Three points on a line

Solution:

Use the lexicographic order.

Solution:

not a right turn

2nd algorithm: incremental

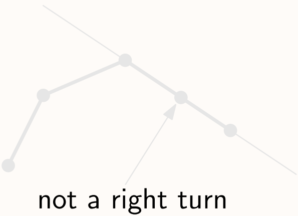
Special cases:

1. Two points have same x-coordinate.
2. Three points on a line

Solution:

Use the lexicographic order.

Solution:



2nd algorithm: incremental

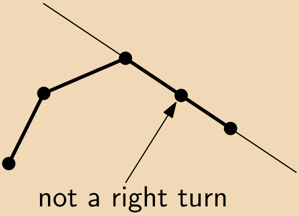
Special cases:

1. Two points have same x-coordinate.
2. Three points on a line

Solution:

Use the lexicographic order.

Solution:



Robustness:

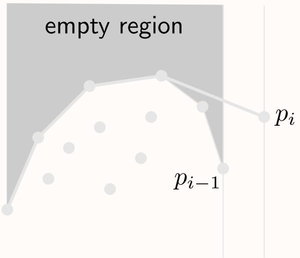
What does the algorithm do in the presence of rounding errors in the floating point arithmetic?

- When such errors occur, it can happen that a point is removed from the convex hull although it should be there, or that a point inside the real convex hull is not removed. But the structural integrity of the algorithm is unharmed: it will compute a closed polygonal chain.
- The only problem that can still occur is that, when three points lie very close together, a turn that is actually a sharp left turn can be interpreted as a right turn. This might result in a dent in the resulting polygon.

Proof of correctness:

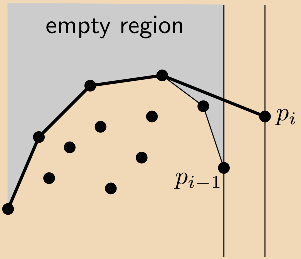
Theorem: The convex hull of a set of n points in the plane can be computed in $\mathcal{O}(n \log n)$ time.

Proof of correctness:



Theorem: The convex hull of a set of n points in the plane can be computed in $\mathcal{O}(n \log n)$ time.

Proof of correctness:



Proof of correctness:

Theorem: The convex hull of a set of n points in the plane can be computed in $\mathcal{O}(n \log n)$ time.

Time Complexity:

- Sorting: $\mathcal{O}(n \log n)$.
- The for-loop is executed a linear number of times.
- For each execution of the for-loop the while-loop is executed at least once. For any extra execution a point is deleted from the current hull.
- So the time complexity for computing upper hull and lower hull is $\mathcal{O}(n)$.
- Total running time: $\mathcal{O}(n \log n)$.

Lower bound:

An $\Omega(n \log n)$ lower bound is known for the convex hull problem.

Proof of correctness:

Theorem: The convex hull of a set of n points in the plane can be computed in $\mathcal{O}(n \log n)$ time.

Time Complexity:

- Sorting: $\mathcal{O}(n \log n)$.
- The for-loop is executed a linear number of times.
- For each execution of the for-loop the while-loop is executed at least once. For any extra execution a point is deleted from the current hull.
- So the time complexity for computing upper hull and lower hull is $\mathcal{O}(n)$.
- Total running time: $\mathcal{O}(n \log n)$.

Lower bound:

An $\Omega(n \log n)$ lower bound is known for the convex hull problem.

Other algorithms:

Algorithm	Speed	Discovered By
Brute Force	$\mathcal{O}(n^4)$	[Anon, the dark ages]
Gift Wrapping	$\mathcal{O}(nh)$	[Chand & Kapur, 1970]
Graham Scan	$\mathcal{O}(n \log n)$	[Graham, 1972]
Jarvis March	$\mathcal{O}(nh)$	[Jarvis, 1973]
QuickHull	$\mathcal{O}(nh)$	[Eddy, 1977], [Bykat, 1978]
Divide-and-Conquer	$\mathcal{O}(n \log n)$	[Preparata & Hong, 1977]
Monotone Chain	$\mathcal{O}(n \log n)$	[Andrew, 1979]
Incremental	$\mathcal{O}(n \log n)$	[Kallay, 1984]
Marriage-before-Conquest	$\mathcal{O}(n \log h)$	[Kirkpatrick & Seidel, 1986]

n: number of points

h: number of points on the boundary of convex hull

Higher dimensions:

- The convex hull can be defined in any dimension.
- Convex hulls in 3-dimensional space can still be computed in $\mathcal{O}(n \log n)$ time (Chapter 11).
- For dimensions higher than 3, however, the complexity of the convex hull is no longer linear in the number of points.

END.