Given points in d-dimensional space, find a good “container” = convex polytope. Many applications, e.g. collision detection, pattern recognition, motion planning . . .

In 2D, imagine putting a rubber band around the points
In 3D, wrap with shrink-wrap

More formally:

A set is **convex** if for every two points \( p, q \) in the set, all points on the line segment \( pq \) are also in the set.

The **convex hull** of set \( S \) is the intersection of all convex sets that contain \( S \).

Note that the convex hull of \( S \) is convex. The fact that the convex hull of a set of points \( S \) is a convex polytope whose vertices are points of \( S \) requires a proof, which we will do later.

https://brilliant.org/wiki/convex-hull/
Convex Hull Algorithms in 2D

Almost any algorithmic paradigm will work, so this problem is a great one for Algorithms courses. See [Zurich notes, Chapter 4].

**Incremental Algorithm** — add points one by one in sorted order by x coordinate

**Example**
Incremental Algorithm — add points one by one in sorted order by x coordinate

general situation

We have:
- \( H_{i-1} = \text{CH}(p_1, \ldots, p_{i-1}) \)
  as a doubly linked list
- \( p_{i-1} \) is a vertex of \( H_{i-1} \)

We want:
- add \( p_i \) to get \( H_i \)
- \( p_i \) is joined to:
  - \( p_u \) by upper bridge
  - \( p_l \) by lower bridge
**Incremental Algorithm** — add points one by one in sorted order by x coordinate

- starting from $p_{i-1}$ scan forward (clockwise) to find $p_l$

- starting from $p_{i-1}$ scan backward (counterclockwise) to find $p_u$

invariant: the line segment from $p_i$ to the current vertex is outside the CH (true initially for line segment $p_ip_{i-1}$)

How to stop the scan

If $p_s$ is above line $p_ip_r$, then $p_e < p_r$ and lower bridge is $p_ip_e$

else scan moves to $p_s$ and $p_r$ will be removed.
Run time

Adding one point

amount of work is like # vertices we remove
could be $\Theta(n)$

So is it $O(n^2)$?

Amortized analysis

we delete a vertex at most once through the course of the algorithm

So total is $\Theta(n)$

+ sorting $\Theta(n \log n)$

Total: $\Theta(n \log n)$. 
**Graham’s Algorithm**

Another sorting-base approach.
1. Sort the points radially around some point $X$ inside the convex hull.

2. Let $p_i =$ point of min $x$-coordinate (then max $y$)
   
   add the points in the (cyclic) sorted order
   
   repeatedly remove the 2nd last point if it is not convex

To find $X$: take average of 3 input points not collinear.

To sort the points radially around $X$: Do not compute angles, use sidedness test.

Runtime: $O(n \log n) + O(n)$
Divide and Conquer Algorithm

Divide the points in two by a vertical line (easy if we sort by x coordinate).
Recurse on each side.
Then combine the two sides.

To combine
Find upper & lower bridge
Start with line segment e
from max x on left
to min x on right
Walk up triangle by triangle
to find upper bridge
Walk down    lower
Divide and Conquer Algorithm

Runtime

\[ T(n) = 2T\left(\frac{n}{2}\right) + cn \]

solves \( T(n) = O(n \log n) \)

+ initial sort \( O(n \log n) \)

Total: \( O(n \log n) \).

Exercise: can you get \( O(n \log n) \) by dividing arbitrarily? (into sets of size \( \frac{n}{2} \))

don't do it by cases!
Lower Bound

There is an \( \Omega(n \log n) \) lower bound on computing the ordered convex hull in 2D on a RAM (Random Access Machine) with +,-,x.

**Proof.** Reduce sorting to finding the convex hull.

map points to parabola
\[ x \rightarrow (x, x^2) \]

Convex Hull of these points (in order) gives sorted order.

input points we want to sort

Note: even finding the (unsorted) CH vertices takes \( n \log n \) (needs different proof)
Output sensitive algorithm

Idea:

Express the run time as a function of input size, $n$, and output size, $h$.

Gift-Wrapping (or “Jarvis’s March”)

$p_1 := \text{min } x \text{ then max } y$

$l_1 := \text{vertical ray through } p_1$

“wrap” line (or ray) $l_1$ by rotating through $p_1$ until it hits first point $p_2$
How to “wrap”

Each “wrap” is like finding a min

Compare \( P_k \) and \( P_e \) by testing is \( P_e \) above/below \( P_k \)

Time for one wrap step is \( O(n) \) (like finding min)

Runtime of Gift Wrapping Algorithm:

How many wrap steps? \( h \) until we wrap around to \( P_i \)

Total time is \( O(n \cdot h) \)

Worst case \( h = n \) this is \( O(n^2) \)

Worse than \( O(n \log n) \)

But if \( h \) is small \( O(n \cdot h) \) is better than \( O(n \log n) \)
So, what is the best algorithm in terms of $n$ and $h$?

$O(n \log h)$ algorithm — first developed by Kirkpatrick and Seidel 1986, uses linear time median finding.

improved by Timothy Chan, 1996.

**Chan’s Algorithm**

Assume $h$ is known (will fix this later)

$m \leftarrow h$

Partition points in \( \lceil \frac{n}{m} \rceil \) subsets of \( \leq m \) points (arbitrarily)

Find $CH$ of each subset, use e.g. Graham or incremental

Time so far \( O \left( \frac{n}{m} \cdot m \log m \right) = O(n \log m) \)
Next run Gift Wrapping, but for the wrap step, don’t check all $n$ points.

We only need to check the most extreme (= min. radially) point of each of the $\lceil \frac{n}{m} \rceil$ convex hulls.

How to find the most extreme point of a convex hull:

- Use binary search on $a \rightarrow b$
- Have a subchain containing extreme pt.
- Test midpoint $c$
- Decide which subchain $a, c$ versus $c, b$.

Runtime: $O(\log m)$
Time for one wrap step: \( O\left(\frac{n}{m} \log m\right) \)

- find extreme in one set
- look at all \( \frac{n}{m} \) sets.

How many wrap steps should we do?

If we did \( h \) wrap steps, we’d have CH

- cost \( O(h \frac{n}{m} \log m) + O(n \log m) \)

- good if \( h = m \) but bad if \( m \) is too small \( O(n \cdot h) \).

How do we find the right \( m \)?

Try different values of \( m \).

- do \( m \) wrap steps \( O(m \frac{n}{m} \log m) + O(n \log m) \)

If we don’t get back = \( O(n \log m) \)

- to \( P \), we know \( m \) was too small.

If we try any \( m \geq h \) we get the CH.
How do we find the right $m$? And don't try too many values of $m$.

A related problem:

Search in a sorted but unbounded array of distinct natural numbers. (in a bounded array $A[1 \ldots k]$ we can search in log $k$ steps.)

use doubling trick

try $i=1, 2, 4, 8, \ldots$

$A[2^i] < x < A[2^{i+1}]$

$log x + 1$ steps to do this

Then use binary search in $A[2^i, 2^{i+1}]$

another $log x$.
How do we find the right $m$?

Try an increasing sequence of values of $m$ until we get one bigger than $h$ (i.e., one where the algorithm finds the CH)

```
try m = 2, 4, 8, 16 ...

m = 2^i

work to do this

2^i \geq h \quad i \geq \log h
```

```
\text{log h}
\sum_{i=1}^{\log h} n \log 2^i = n \sum_{i=1}^{\log h} i = n \left(\frac{\log h \cdot (\log h + 1)}{2}\right) \quad \text{Too big, want } n \log h
```

```
\text{log log h}
\sum_{i=1}^{\log \log h} n \log (2^{2^i}) = n \sum_{i=1}^{\log \log h} 2^i = n \sum_{i=1}^{\log \log h} \log \log h

= n \log h \quad \text{Right bound}
```

Summary

- algorithms for Convex Hull in the plane

References

- [CGAA] Section 1.1
- [Zurich notes] Chapter 4
- [O’Rourke] Chapter 3

Next: convex hull in higher dimensions.