CPSC 420 Lecture 2 : Today's announcements:

Reading: Cormen 3rd Edition 1029–1039

Next time: Chan's Algorithm } Wikipedia
Voronoi Diagrams }

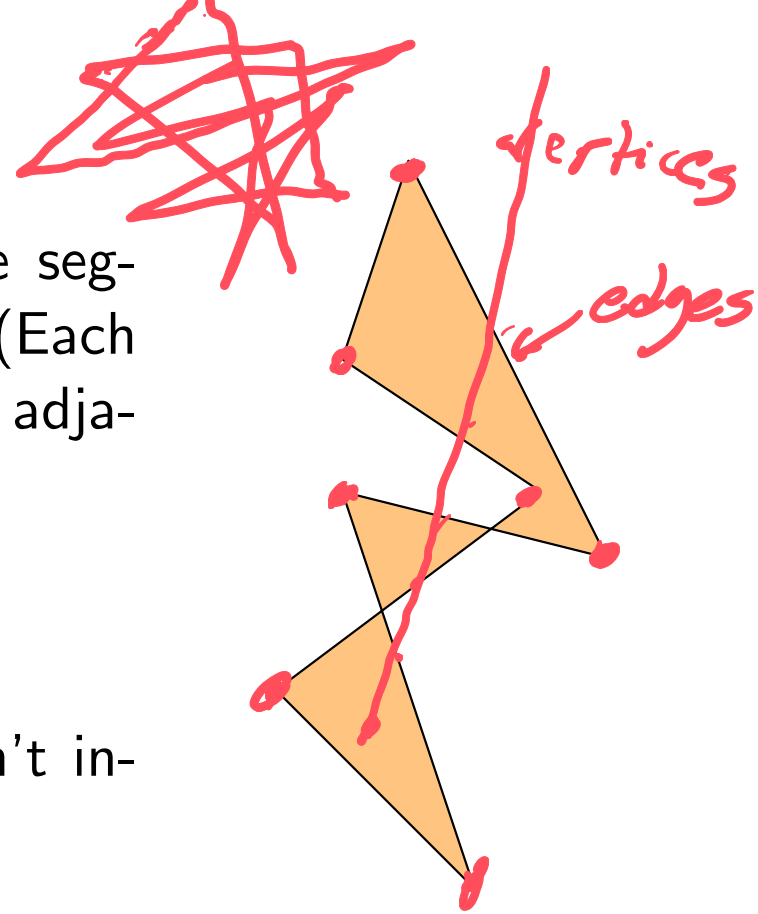
- HW1 available on Gradescope, due Jan 19, 23:59
- Examlet 1 on Jan 27 in class.

Today's Plan

- Convex hulls
  - Jarvis March
  - Graham's Scan
  - Optimal?

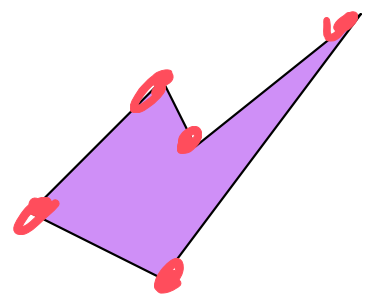
Polygon

A **polygon** is a circular sequence of line segments in the plane joined end-to-end. (Each segment endpoint is the endpoint of two adjacent segments in the sequence.)



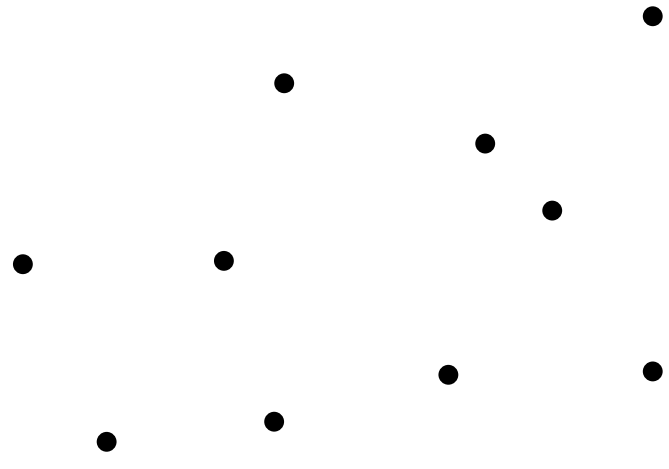
A polygon is **simple** if the segments don't intersect in a non-endpoint.

The convex hull of a finite point set P is a simple polygon.



Jarvis March (Gift-wrapping)

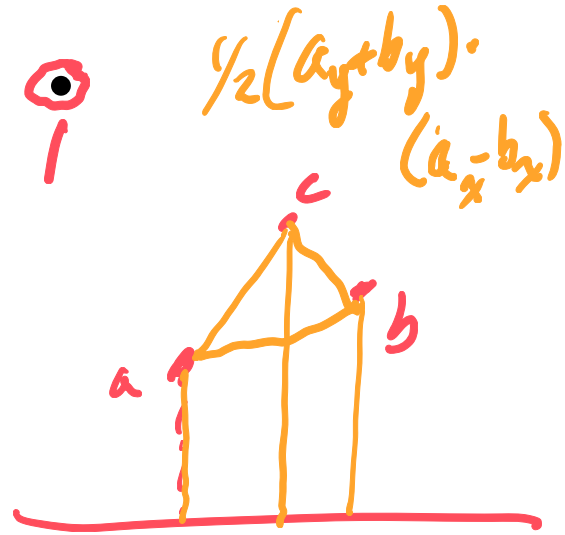
Idea: Tie a string to a point $p_1 \in P$ that is on the $CH(P)$. Rotate a taut string around the points until it "bends" at the next point on $CH(P)$. Keep going until back to p_1.



Turn test

Path $a \to b \to c$ makes a left turn at b iff

$$\det \begin{pmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{pmatrix} > 0$$

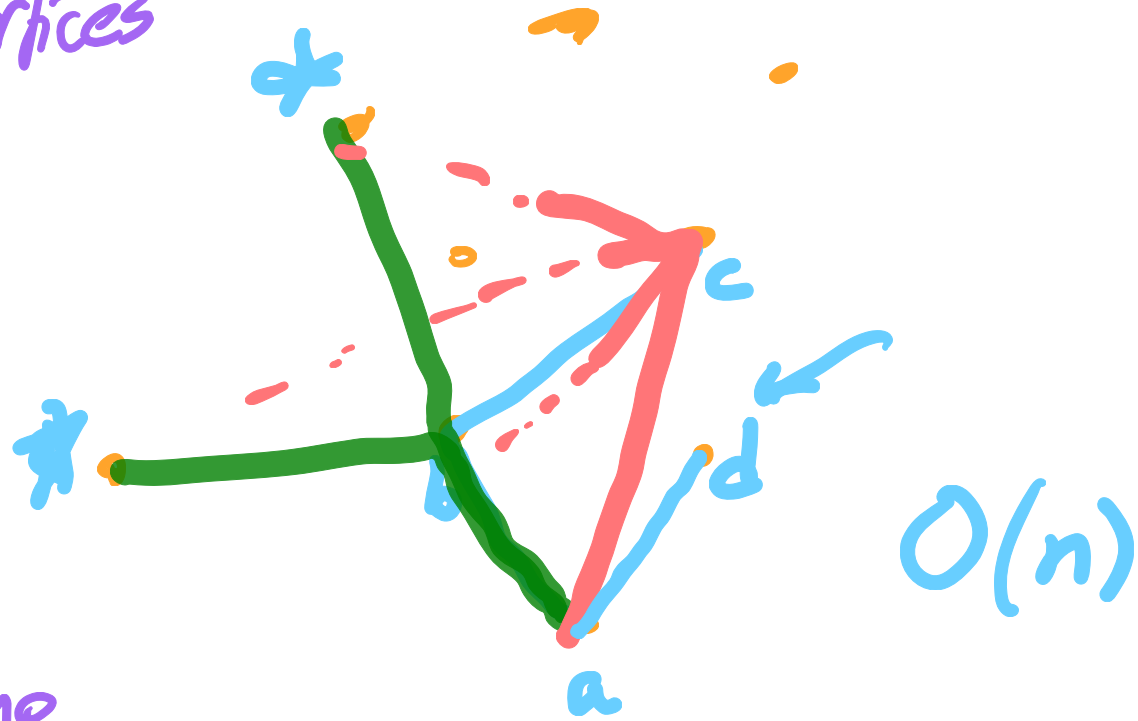


$$a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y > 0$$

How do we use `LeftTurn(a,b,c)` to wrap?

Jarvis March Algorithm

h = # hull vertices



$O(n)$

Running time for finding Hull = $O(nh)$

Jarvis March Algorithm

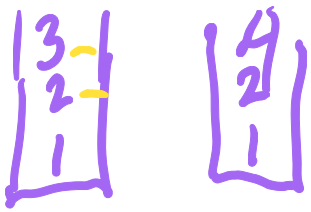
```
INPUT: P[0...n-1] array of 2D points
OUTPUT: Array of convex hull vertices in ccw order.

If n < 3 return distinct points in P
a0 = index of point with min y-coord in P
h = 0, a = a0, b = (any index not a0) mod n
while b != a0
  for c = 0 to n-1
    if (c != a and c != b and
        not LeftTurn(P[a],P[b],P[c])) then
      b = c
  H[h++] = P[b], a = b
return H
```

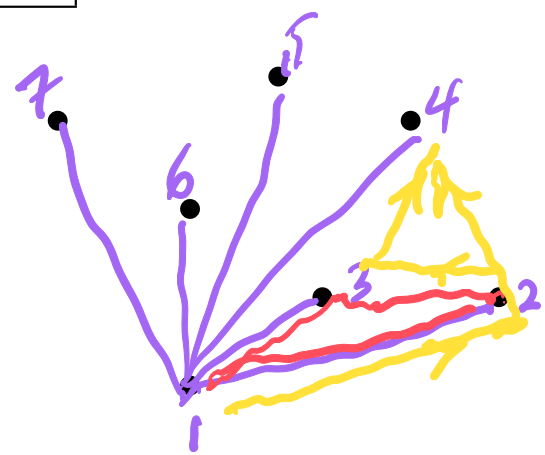
Running time: $O(nh)$ where h is number of vertices of convex hull.

Graham's Scan

1. Find point p_1 in P with smallest y-coord (break ties using smaller x-coord)
2. Sort remaining points in P by ccw angle around p_1. Let $p_2, p_3, \ldots, p_n$ be these points in order. (What about ties?)
3. Start with $p_1 p_2 p_3$ as candidate hull. Put them on a stack S.
4.

```
for i = 4 to n
    while not LeftTurn(S[top-1], S[top], p_i)
        pop(S)
    push p_i onto S
return S
```





Graham's Scan Run Time

1. Finding p_1 takes $\Theta(n)$ time
2. Sorting by angle takes $\Theta(n \log n)$ time
3. Put $p_1 p_2 p_3$ on a stack S takes $O(1)$ time
4.

```
for i = 4 to n
    while not LeftTurn(S[top-1], S[top], p_i)
        pop(S)
    push p_i onto S
return S
```

One iteration of for-loop causes $< n$ pops $\Rightarrow$ $O(n^2)$ time (not tight)

But, over all iterations, #pushes $< n$ and #pops $<$ #pushes

So total time taken by for-loop is $\Theta(n)$

$O(n \log n)$

Faster Convex Hull?

Is this the fastest algorithm for Convex Hull?

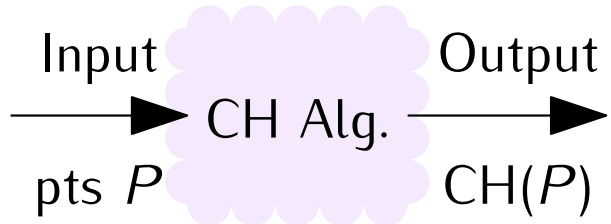
How powerful is our computer?

- It can multiply, add, subtract, compare two real numbers in one step.
- It cannot wrap a string around n objects in linear time.

Is this the fastest algorithm using an Algebraic Decision Tree model computer for Convex Hull?

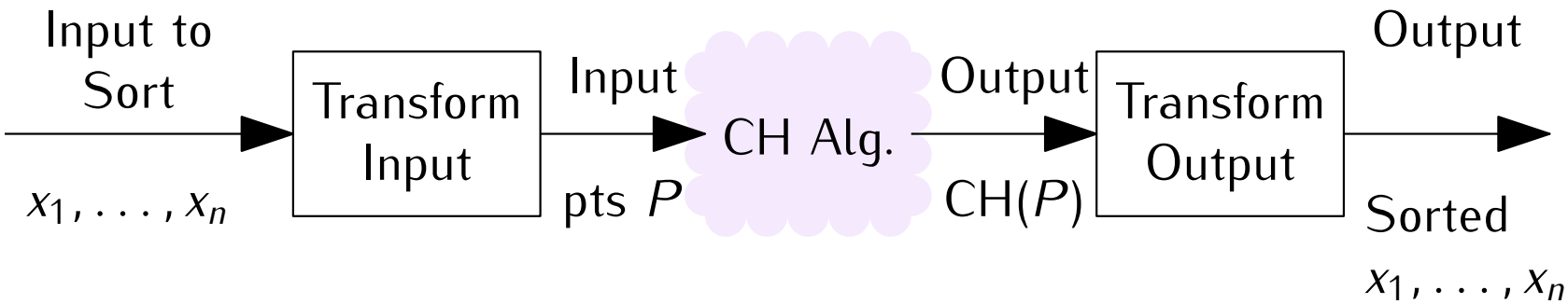
Faster Convex Hull?

Suppose there exists a really fast Convex Hull algorithm.



Faster Convex Hull?

Suppose there exists a really fast Convex Hull algorithm.



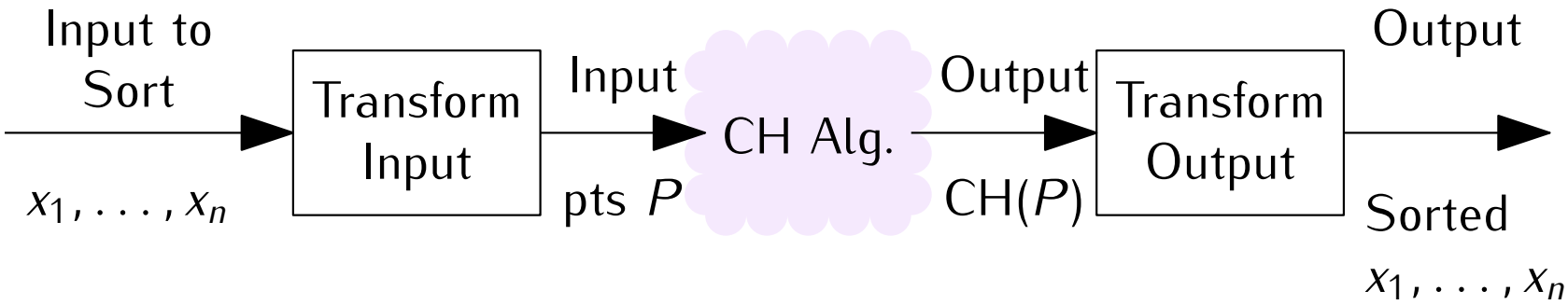
Create an alg. that transforms an input to the sorting problem into an input to CH problem... Make this really fast.

let CH Alg. do all the hard work...

and then transform CH output into the answer to the sorting problem. Make this really fast.

Faster Convex Hull?

Suppose there exists a really fast Convex Hull algorithm.

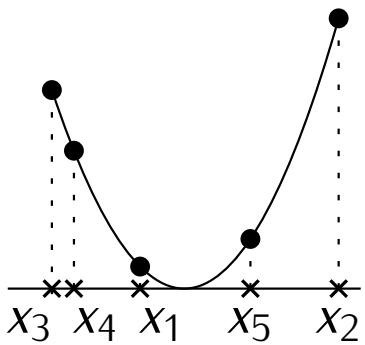


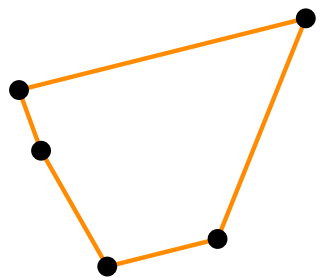
Create an alg. that transforms an input to the sorting problem into an input to CH problem... Make this really fast.

let CH Alg. do all the hard work...

and then transform CH output into the answer to the sorting problem. Make this really fast.

for all i:
$x_i \to (x_i, x_i^2)$





Find hull pt w/ min x-coord
Output x-coords in ccw order

Transform Input

Transform Output

Why did we do this?

We already have good algorithms for sorting. Why make this complicated sorting algorithm?

Why did we do this?

We already have good algorithms for sorting. Why make this complicated sorting algorithm?

Because we know Sorting complexity is $\Omega(n \log n)$.
(i.e. fastest alg. for sorting takes $\geq cn \log n$ steps for large n)

We've just constructed a sorting algorithm that takes time

$$T(n) = T_I(n) + T_{CH}(n) + T_O(n)$$

We know $T(n) \geq cn \log n$ (sorting complexity).
We know $T_I(n) \leq c_I n$ and $T_O(n) \leq c_O n$ for some constants c_I and c_O (from the input and output transformations).
That means (since $T_{CH}(n) = T(n) - T_I(n) - T_O(n)$)

$$T_{CH}(n) \geq cn \log n - c_I n - c_O n \in \Omega(n \log n).$$

This holds for **any** convex hull algorithm, so the complexity of convex hull is $\Omega(n \log n)$.