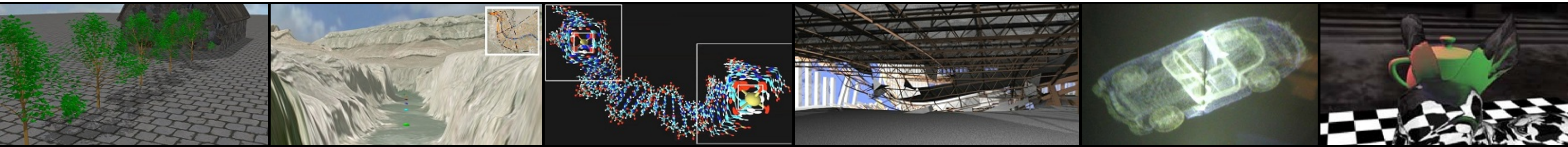
COT 4521: Introduction to Computational Geometry

<u>Convex Hull</u>

Paul Rosen
Assistant Professor
University of South Florida

Some slides from Valentina Korzhova

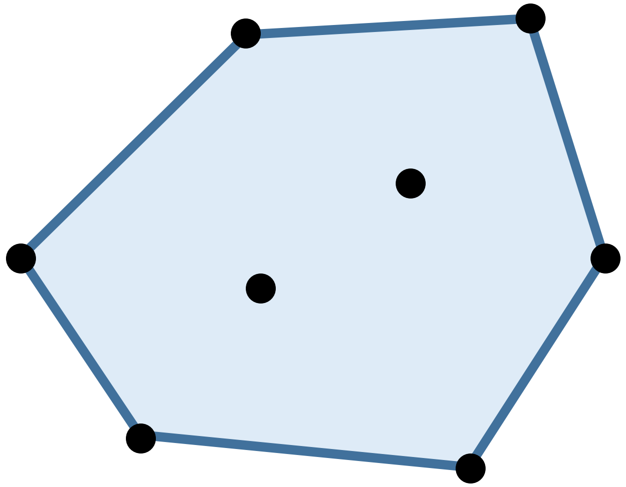


Convex Hull Algorithms

- Definitions
- Naïve Algorithms
- QuickHull
- Gift Wrapping
- Graham Scan
- Incremental
- Divide-and-Conquer

Convex hulls

- Given N distinct points on the plane, the ***convex hull*** of these points is the smallest convex polygon enclosing all of them

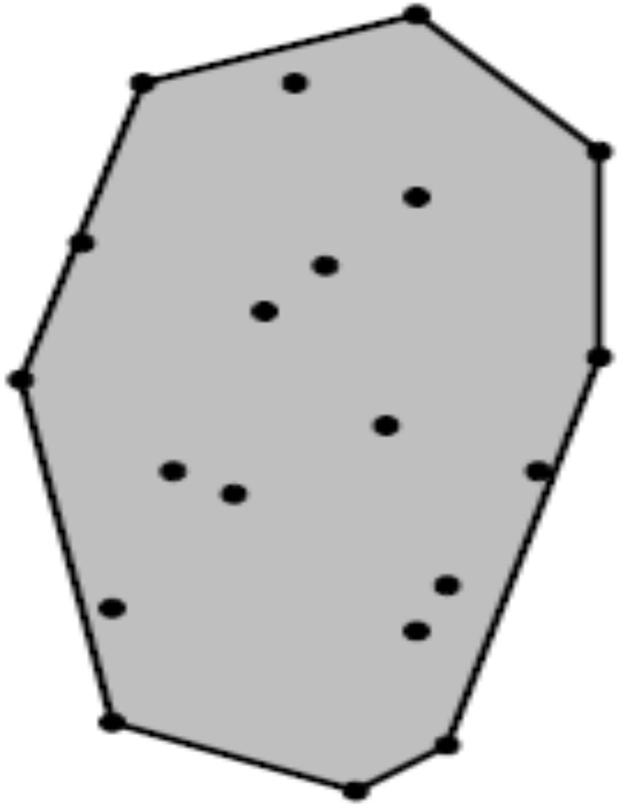


Applications of convex hull

- Fitting ranges with a line
  - Sheep and goats problem—Can you draw a straight line fence that will separate the sheep from the goats?
    - Take the convex hull of each set, if they do not intersect then you can put in a fence. If they intersect, no. (useful in data mining)
- Collision avoidance
  - Robotics problem - if the convex hulls don't run into each other than the robots won't either.
- Smallest box
  - Finding the smallest area rectangle enclosing a polygon.
- Shape analysis
  - Point shapes can be classified by the similarity of their convex shapes.

Convex hull problem

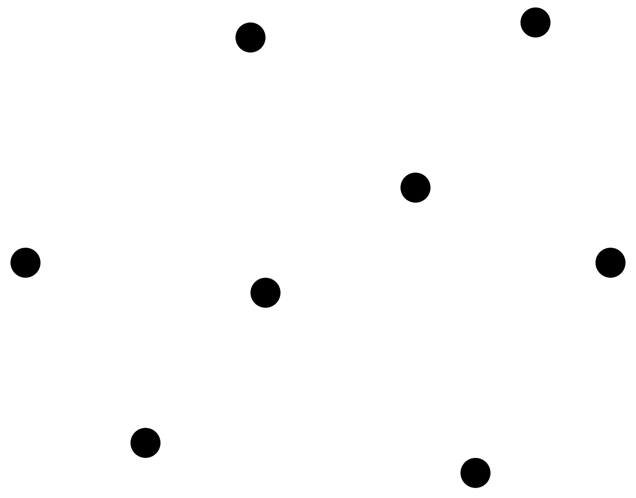
- Give an algorithm that computes the convex hull of any given set of n distinct points in the plane efficiently
- The output has at least 3 and at most n points, for $n > 2$, so it has size between $O(1)$ and $O(n)$
- The output is a convex polygon so it should be returned as a sorted sequence of the points, counterclockwise along the boundary
- Question: Is there any hope of finding an $O(n)$ time algorithm?



Definitions of convexity and convex hull

- The convex hull of a finite set of points S in the plane is the smallest convex polygon P that encloses S, smallest in the sense that there is no other polygon P' such that P $\supseteq$ P' $\supseteq$ S

- Ideas?



NAÏVE ALGORITHMS FOR EXTREME POINTS

Algorithm: INTERIOR POINTS

for each i do

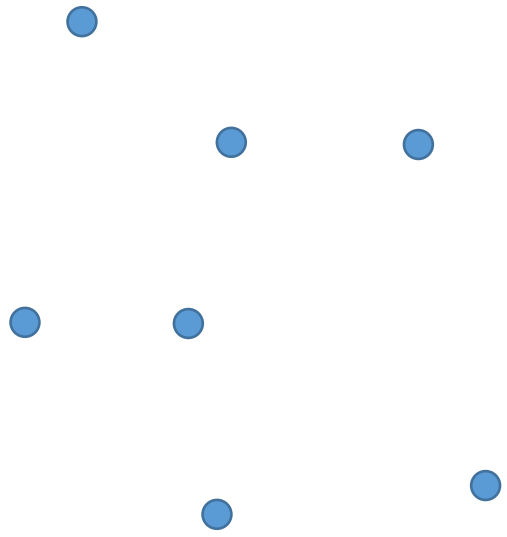
    for each $j \neq i$ do

        for each $k \neq j \neq i$ do

            for each $L \neq k \neq j \neq i$ do

                if p_L in triangle(p_i, p_j, p_k)

                    then p_L is nonextreme

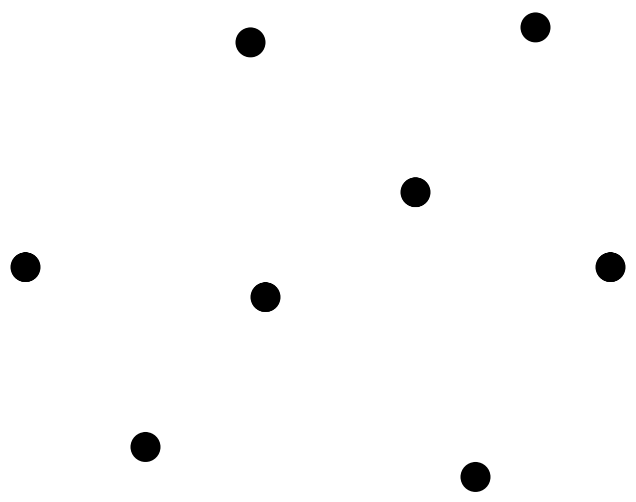


Performance? **O(n^4)**

Definitions of convexity and convex hull

- The convex hull of a set of points S is also intersection of all half spaces that contain S

- Ideas?



NAÏVE ALGORITHMS FOR EXTREME POINTS

Algorithm: EXTREME EDGES

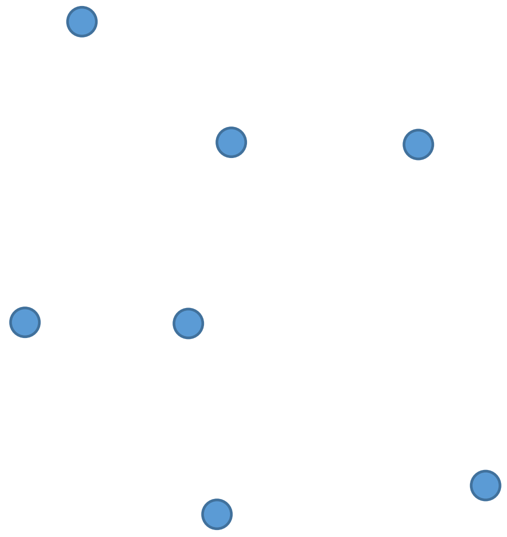
for each i do

  for each $j \neq i$ do

    for each $k \neq j \neq i$ do

      if p_k is not left or on (p_i, p_j)

        then (p_i, p_j) is not extreme



Performance? **$O(n^3)$**

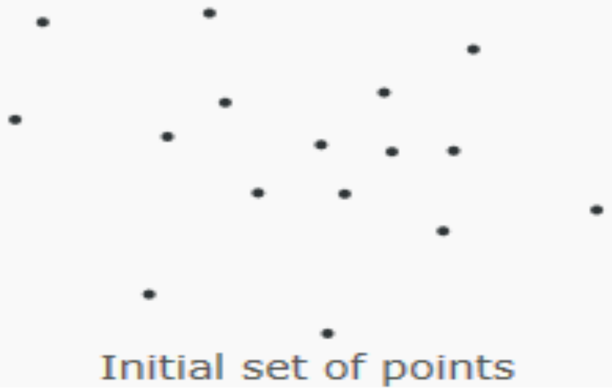
QuickHull

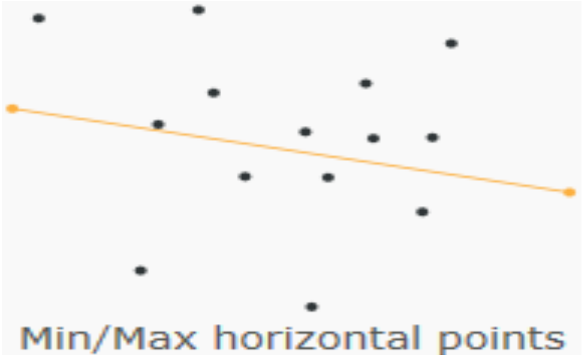
- Concentrate on points close to hull boundary
  - Named for similarity to Quicksort

- The idea is:
  - Discard many points as definitely interior to the hull
  - Concentrate on those to the hull boundary

QUICKHULL ALGORITHM

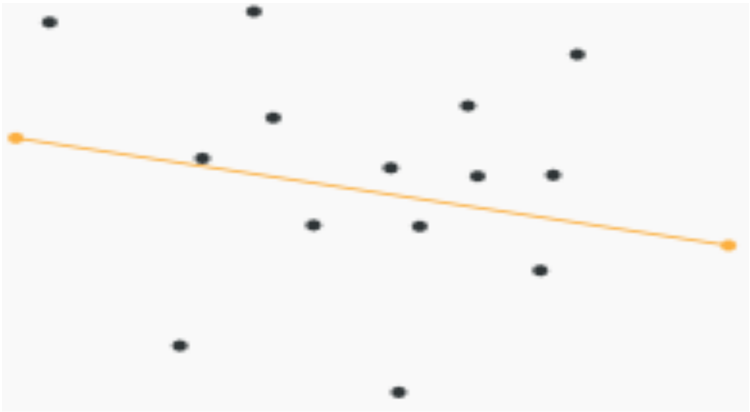
- FIND TWO DISTINCT EXTREME POINTS
  - Use the rightmost lowest and leftmost highest points x and y, which are guaranteed extreme and distinct
  - The full hull is composed of an **"upper hull"** above a line and a **"lower hull"** below a line





QuickHull Algorithm

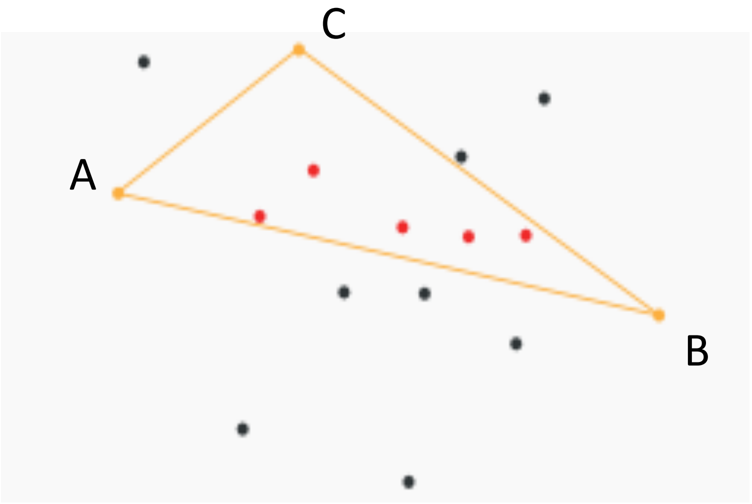
- Divide
    - The line formed by these two points is used to divide the set into two different parts
    - Everything left from this line is considered one part, everything right of it is considered another one
    - Both of these parts are processed recursively



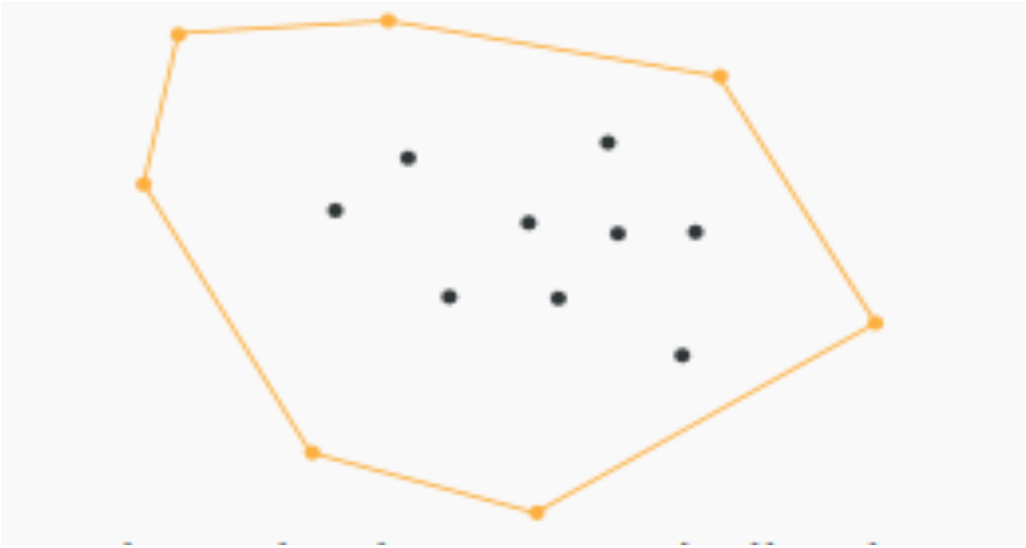
- Find extreme points in an "upper hull" and in a "lower hull"

QuickHull Algorithm

- Discard all points inside Δ
- Operate recursively on the points outside segments AC and BC

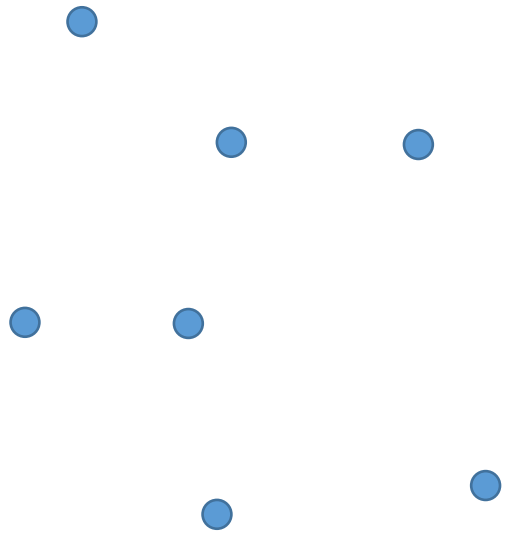


QuickHull Algorithm



QuickHull Example

- Find two distinct extreme points
- Divide
- Find extreme points in upper and lower hull
- Discard all points inside Δ
- Recurse



QuickHull Performance

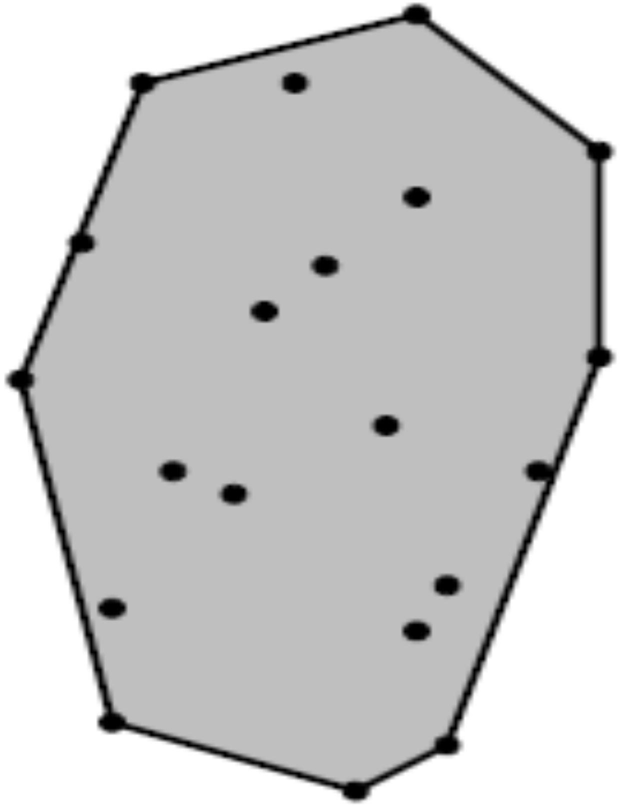
- Performance?
  - Worst case: $O(n^2)$
    - Example?
  - Average case: $\Theta(n \log n)$

Extreme points

- The extreme points of a set S of points in plane are the vertices of the convex hull at which the <u>interior angle is strictly convex, less than π</u>

- A set of points S is said to be strongly convex if it consists of only extreme points

- Ideas?



Jarvis's Algorithm (Gift-Wrapping)

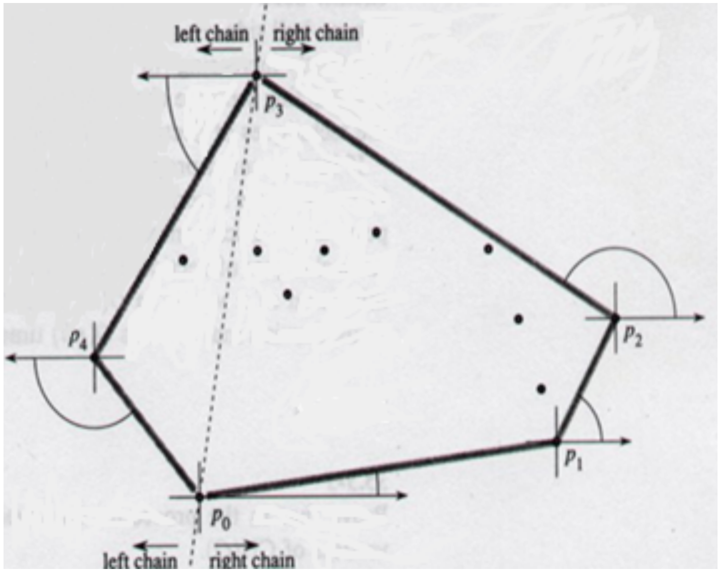
- The simplest algorithm for computing convex hulls simply simulates the process of wrapping a piece of string around the points

- This algorithm is called Jarvis's march or gift-wrapping algorithm

- Jarvis's march starts by computing the bottommost point, since we know this point must be a convex hull vertex

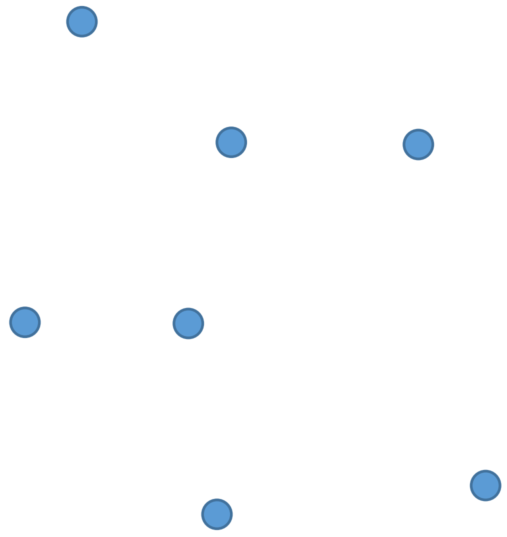
2D Gift Wrapping

- The first vertex is chosen is lowest p_0
- The next vertex, p_1, has the least polar angle with any point with respect to p_0
- Then, p_2, has the least polar angle with respect to p_1
- The right chain goes as high as the highest point p_3
- Then, left chain is constructed with respect to the negative x-axis



Example

- The first vertex is chosen is lowest p_0
- The next vertex, p_1, has the least polar angle with any point with respect to p_0
- Then, p_2, has the least polar angle with respect to p_1
- The right chain goes as high as the highest point p_3
- Then, left chain is constructed with respect to the negative x-axis



2D Gift Wrapping

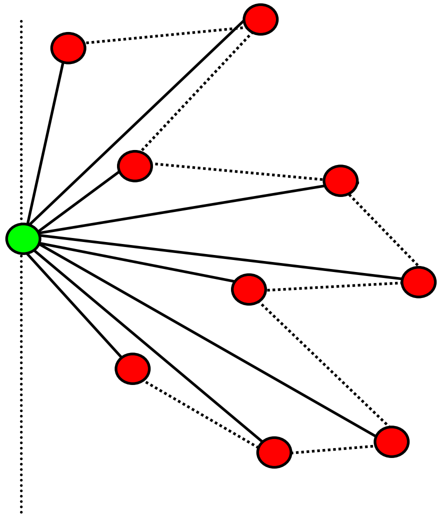
- What is the performance of Gift Wrapping?
  - Since the algorithm spends $O(n)$ time for each convex hull vertex, the worst-case running time is $O(n^2)$

- Right?
  - This naive analysis hides the fact that if the convex hull has very few vertices, Jarvis's march is extremely fast.
  - A better way to write the running time is $O(nh)$, where h is the number of convex hull vertices.

- In the worst case, $h = n$, and we get our old $O(n^2)$ time bound
  - Output-sensitive algorithm; the smaller the output, the faster the algorithm

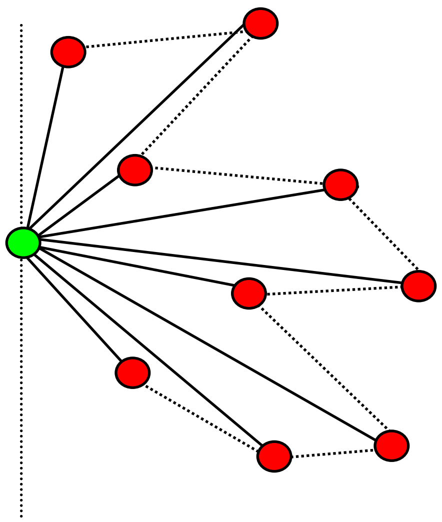
Graham's Algorithm

- Given the left most point
- Sort the points by angle, counterclockwise about that point
- Process the points in their sorted order (use data structure – stack)



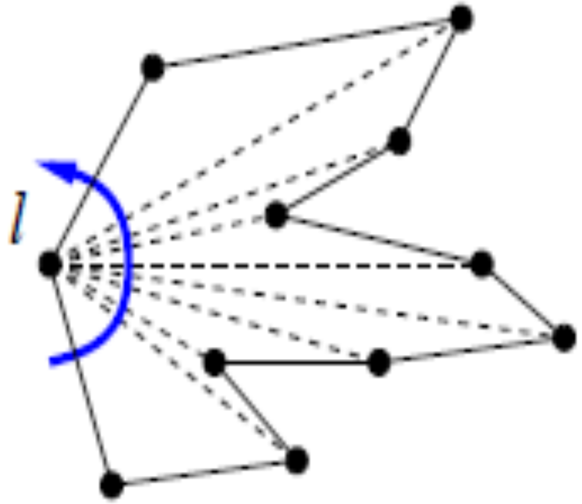
GRAHAM'S ALGORITHM

- POINTS SORTED ANGULARLY PROVIDE "STAR-SHAPED" STARTING POINT
- PREVENT "DENTS" AS YOU GO VIA CONVEXITY TESTING



Graham's Algorithm

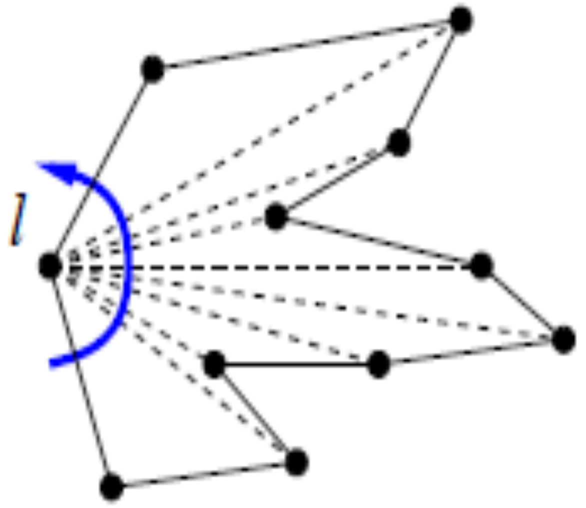
- Start Graham's scan by finding the leftmost point *l*
- Then we sort the points in counterclockwise order around *l*.
  - We can do this in O(n log n) time with any comparison-based sorting algorithm (quicksort, merge sort, heap sort, etc.).
  - To compare two points p and q, we check whether the triple *l*; p; q is oriented clockwise or counterclockwise.
- Once the points are sorted, we connected them in counterclockwise order, starting and ending at *l*.
- The result is a simple polygon with n vertices.



GRAHAM'S ALGORITHM

- USING A STACK DATA STRUCTURE, PLACE L AND THE 2 POINTS AFTER L ONTO THE STACK

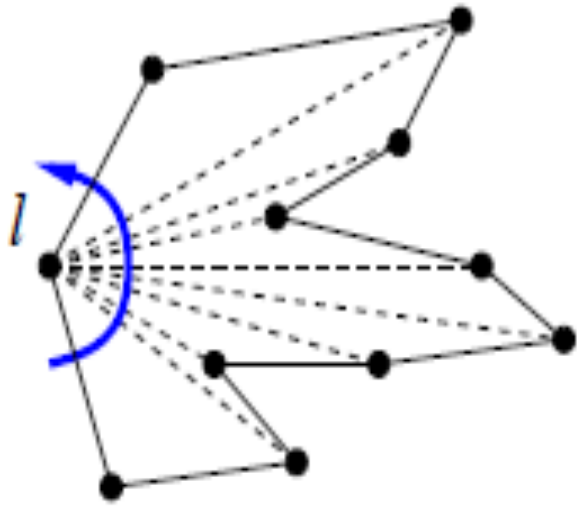
- REPEAT UNTIL L
  - Place the the next element on the stack
  - Check for convexity



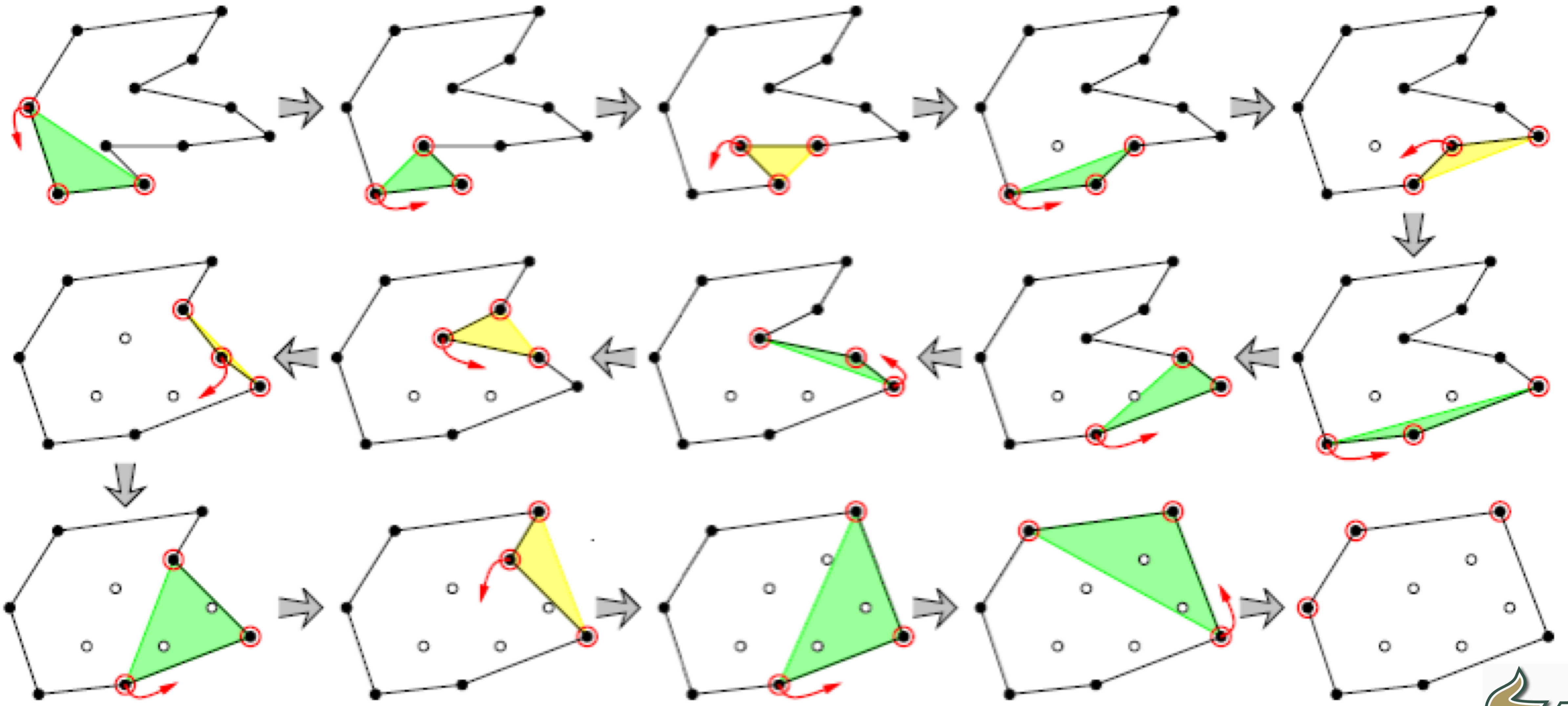
Graham's Algorithm

- Check for convexity
  - Pop the top 3 elements from the stack—p; q; r of the polygon (a initially, these are *the* 2 vertices after *l*)

  - We now apply the following two rules:
    - If p; q; r are in counterclockwise order—Push p; q; r back onto the stack
    - If p; q; r are in clockwise order—Remove q from the polygon by pushing p and q back onto the stack. Repeat convexity check.



Graham's Algorithm Example

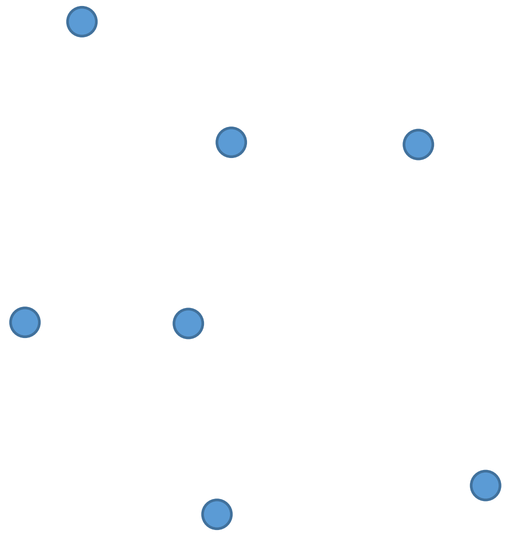


Graham Scan Example

- PLACE L AND THE 2 POINTS AFTER L ONTO THE STACK
- REPEAT UNTIL L
  - Place the the next element on the stack
  - Check for convexity
    - Pop the top 3 elements from the stack—p; q; r of the polygon (a initially, these are *the* 2 vertices after *l*)

      If p; q; r are in counterclockwise order—Push p; q; r back onto the stack

      If p; q; r are in clockwise order—Remove q from the polygon by pushing p and q back onto the stack. Repeat convexity check.



Complexity of Graham's Algorithm

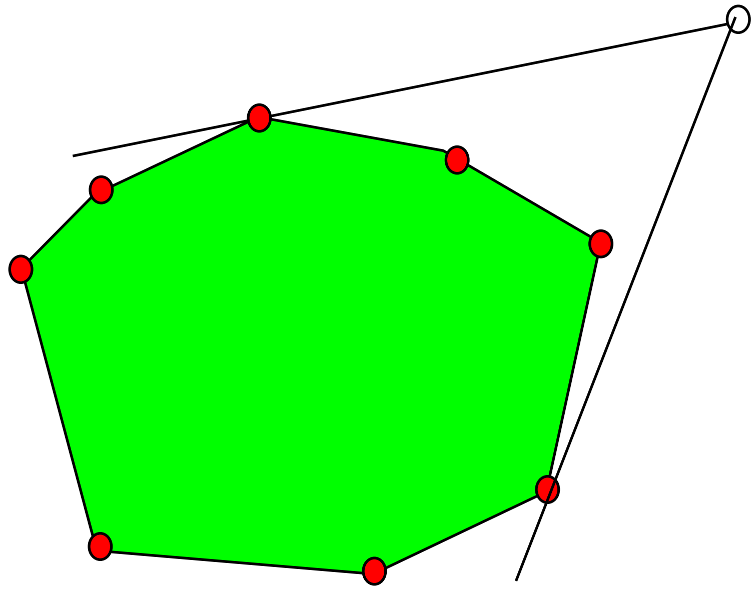
- Sorting phase: O(n log n)

- Scanning phase
  - Whenever a point moves forward, it moves onto a vertex that hasn't seen a point before (except the last time)—So the first rule is applied n - 2 times.
  - Whenever a point moves backwards, a vertex is removed from the polygon—So the second rule is applied exactly n - h times, where h is as usual the number of convex hull vertices.
  - Since each counterclockwise test takes constant time, the scanning phase takes O(n) time altogether.

- Overall: O(n log n)

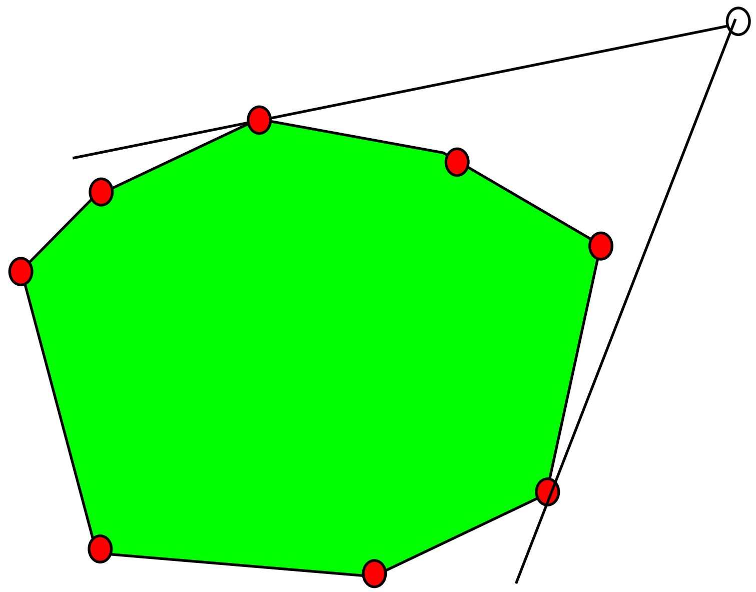
2D Incremental

- Add points, one at a time
  - update hull for each new point by finding tangent points
- Key step becomes adding a single point to an existing hull.



How do we find tangets?

- Naïvely?
- Optimal?



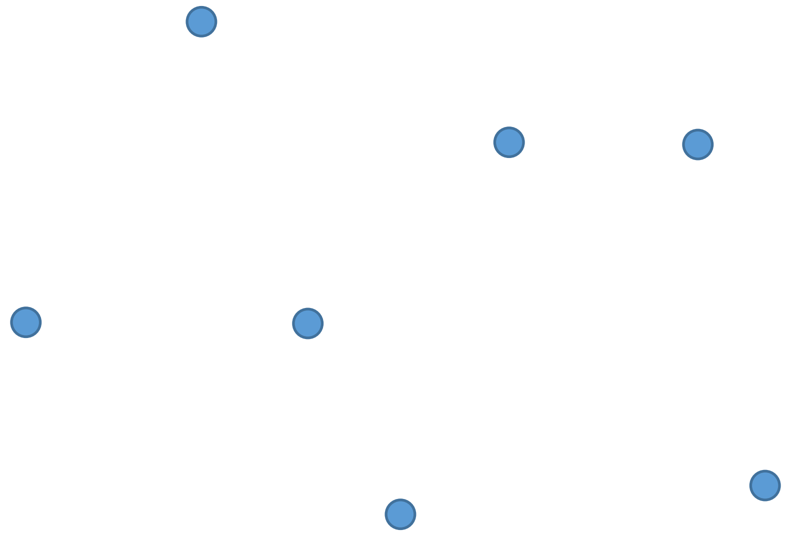
2D Incremental Optimal

- Preorder the points by their x coordinate, so that $p \notin Q$ at each step
- connecting the rightmost point of the hull to the point being added with two segments, called bridges
- Use the rules similar to Graham scan determine the new convex hull

2D Incremental Optimal

- Expand the bridges
  - As long as there is a clockwise turn at the endpoint of either bridge, remove that point from the circular sequence of vertices and connect its two neighbors.
  - As soon as the turns at both endpoints of both bridges are counter-clockwise, stop.
- At that point, the bridges lie on the upper and lower common tangent lines of the two sub-hulls.

2D Incremental Optimal Example



Optimal incremental algorithm

- the total work over life of the algorithm for finding tangent lines is $O(n)$
- This then provides an O(n log n) algorithm

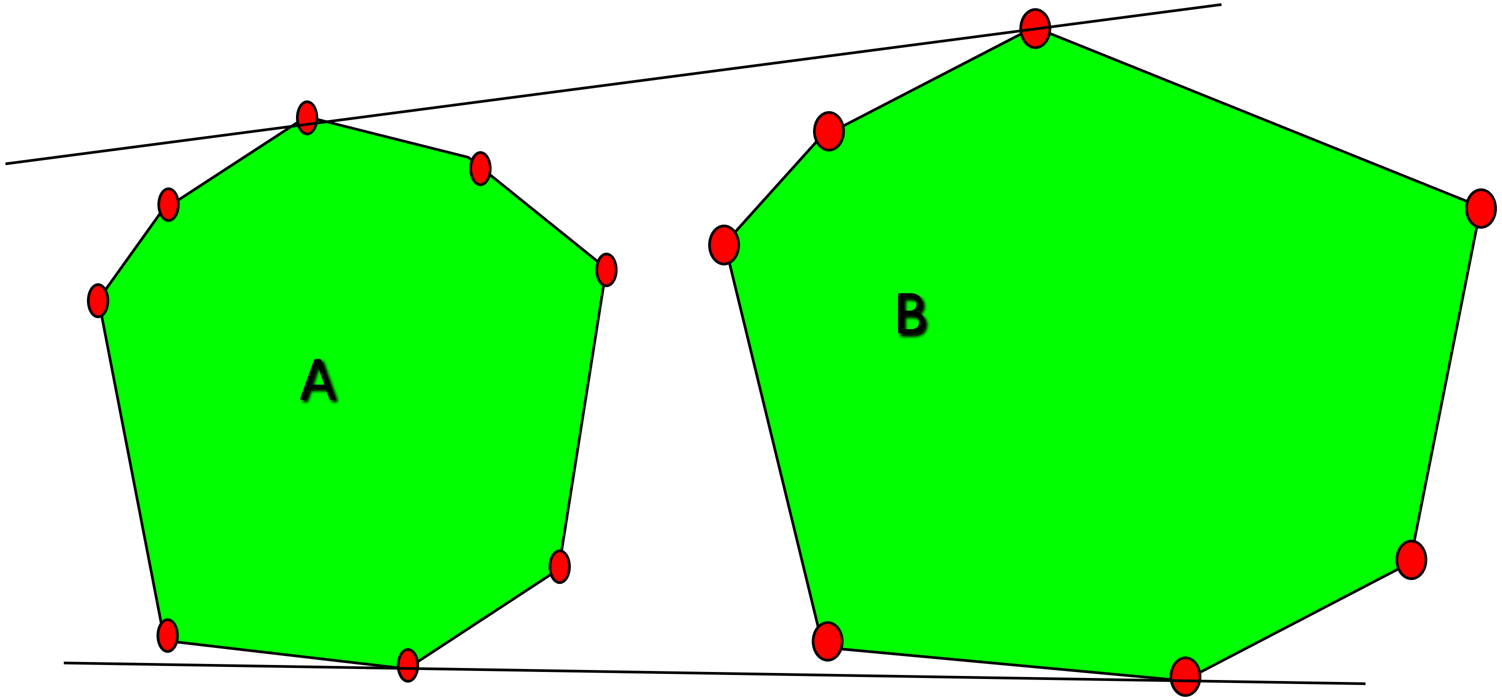
The divide-and-conquer design paradigm

- Divide the problem (instance) into subproblems, each of size n/b
- Conquer the subproblems by solving them recursively.
- Combine subproblem solutions.
    - Runtime is f(n)

Divide and Conquer

- Idea of Divide and Conquer algorithm:
  - Start by choosing a pivot point p.
  - Partitions the input points into two sets L and R, containing the points to the left of p, including p itself, and the points to the right of p, by comparing x-coordinates.
  - Recursively compute the convex hulls of L and R. Finally, merge the two convex hulls into the final output.

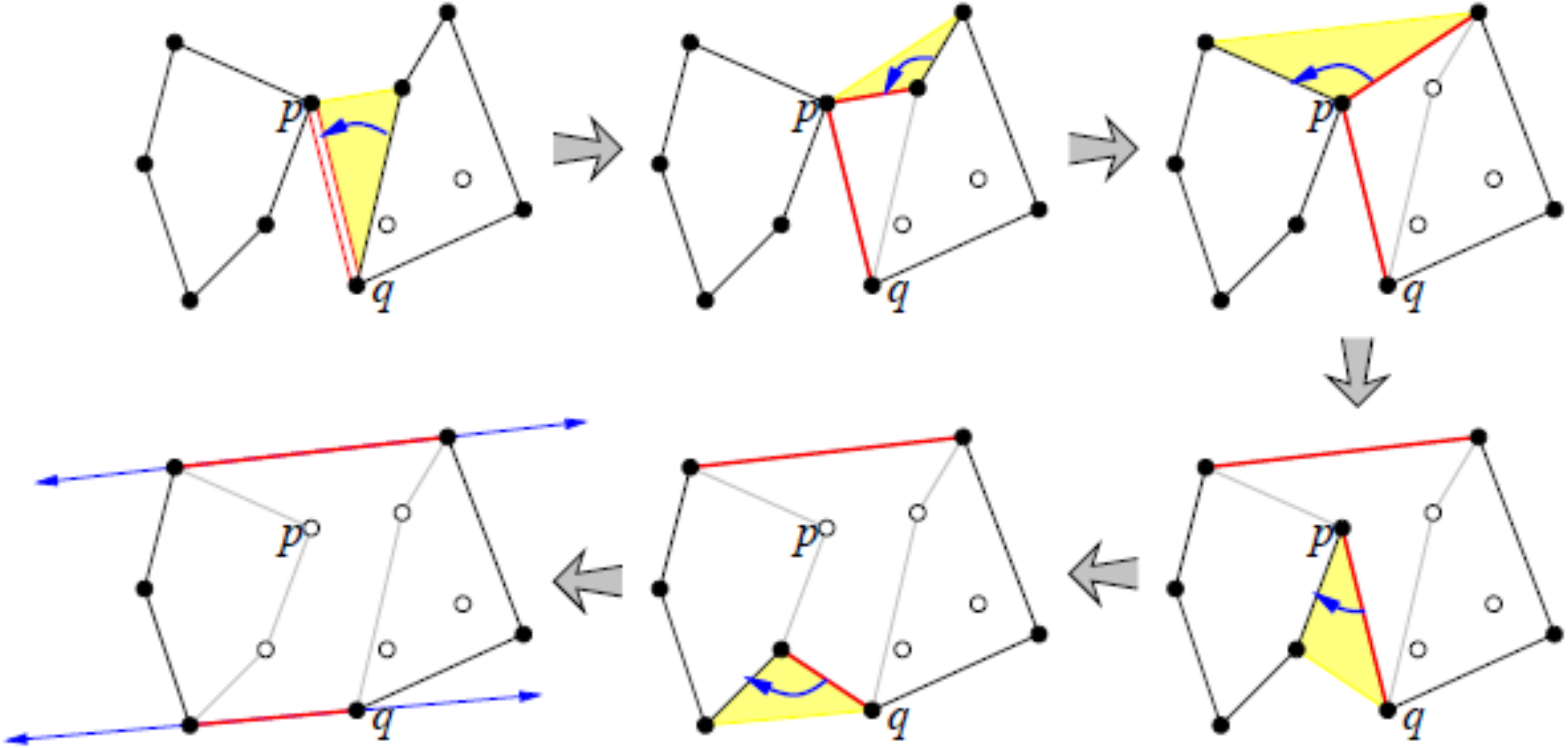
MERGING, THOUGHTS?



Divide and Conquer (Merging)

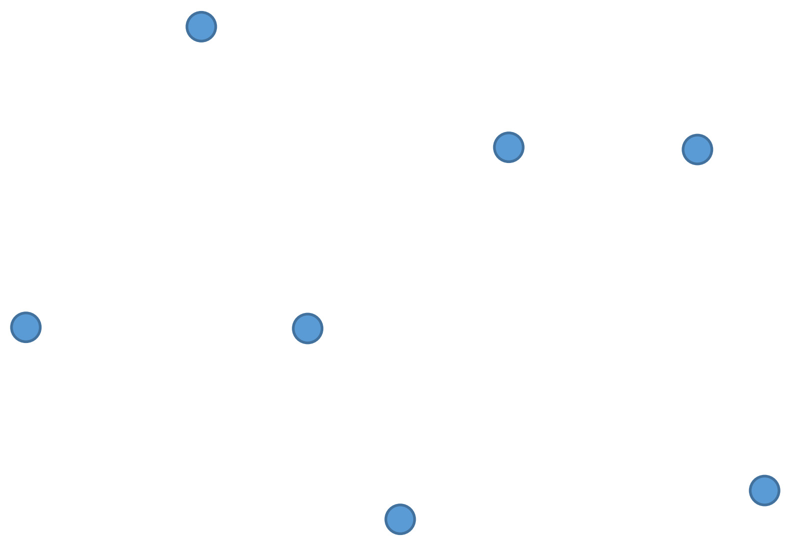
- Find the rightmost point of the hull of L with the leftmost point of the hull of R
  - Call these points p and q, respectively.
- connecting pq with bridges
- Use the rules of graham scan/Incremental to isolate the merged convex hull

Divide and Conquer (Merge Example)



Merging the left and right subhulls.

Divide and Conquer Example

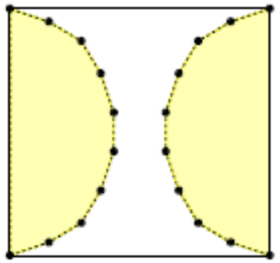


Analysis of Divide and Conquer

- Merging the two sub-hulls takes O(n) time.
- The running time is given by the recurrence $T(n) = O(n)+T(k)+T(n - k)$, just like quicksort, where k the number of points in R.

Analysis of Divide and Conquer

- Just like quicksort, if we use a naive deterministic algorithm to choose the pivot point p, the worst-case running time of this algorithm is $O(n^2)$.
- If we choose the pivot point randomly, the expected running time is $O(n \log n)$.
- There are inputs where this algorithm is clearly wasteful (at least, clearly to us). If we're really unlucky, we'll spend a long time putting together the sub-hulls, only to throw almost every thing



A set of points that shouldn't be divided and conquered.

Analysis of Divide and Conquer

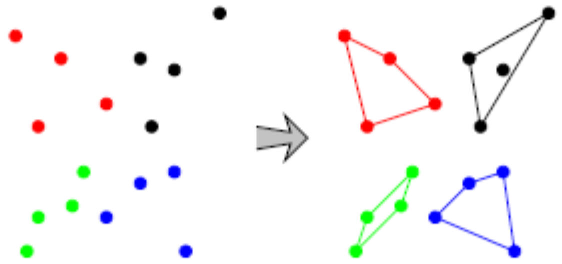
- Merge sort style implementation
  - Initial x sorting takes $O(N\ logN)$ time.
  - Divide the set of points into two sets A and B takes O(N) time
  - O(N) for merging (computing tangents).
  - Recurrence solves to T(N) = O(N log N).

Chan's algorithm (output-sensitive CH)

- The running time of this algorithm, which was discovered by Timothy Chan in 1996, is $O(n \log h)$.
- Chan's algorithm is a combination of divide-and-conquer and gift-wrapping.

CHAN'S ALGORITHM (OUTPUT-SENSITIVE CH)

- FIRST SUPPOSE A `LITTLE BIRDIE' TELLS US THE VALUE OF H;
- CHAN'S ALGORITHM STARTS BY SHATTERING THE INPUT POINTS INTO N/H ARBITRARY SUBSETS, EACH OF SIZE H, AND COMPUTING THE CONVEX HULL OF EACH SUBSET USING (SAY) GRAHAM'S SCAN.
- THIS MUCH OF THE ALGORITHM REQUIRES $O(\left(\frac{n}{h}\right)\ h \log h) = O(n \log h)$ TIME.



Shattering the points and computing subhulls in $O(n \log h)$ time.