

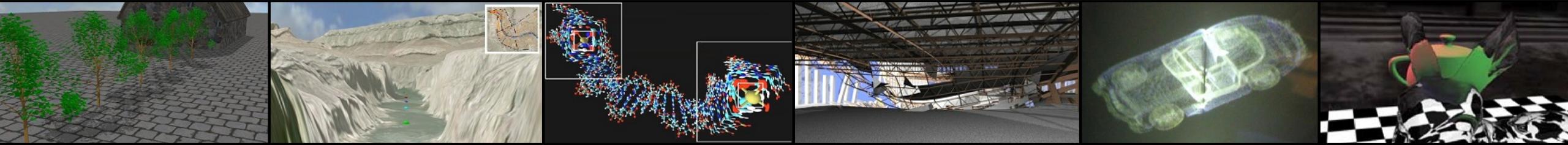
COT 4521: INTRODUCTION TO COMPUTATIONAL GEOMETRY



Convex Hull

Paul Rosen
Assistant Professor
University of South Florida

Some slides from Valentina Korzhova



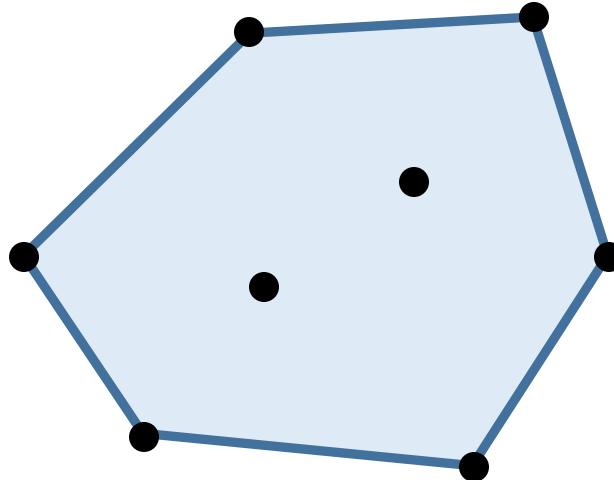
CONVEX HULL ALGORITHMS

- 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



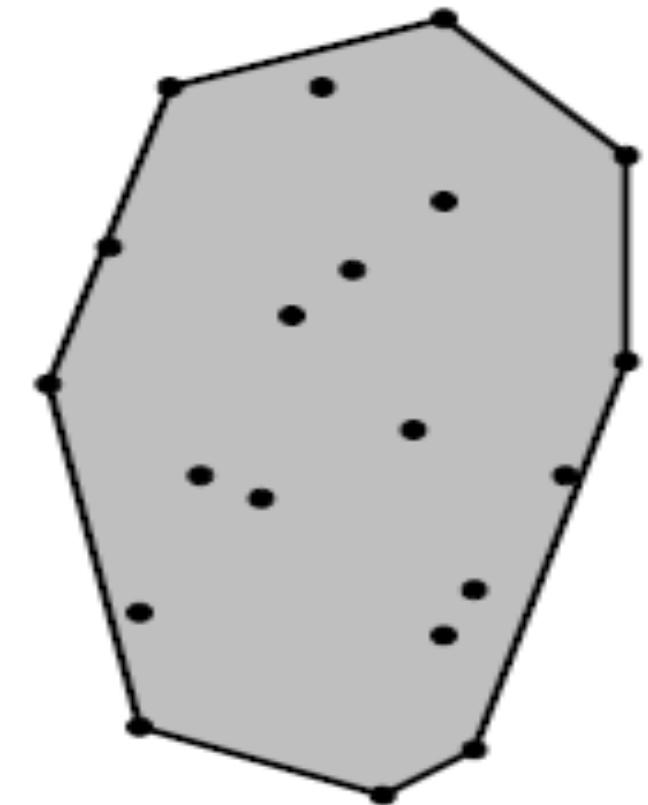
APPLICATIONS OF CONVEX HULL

- 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.



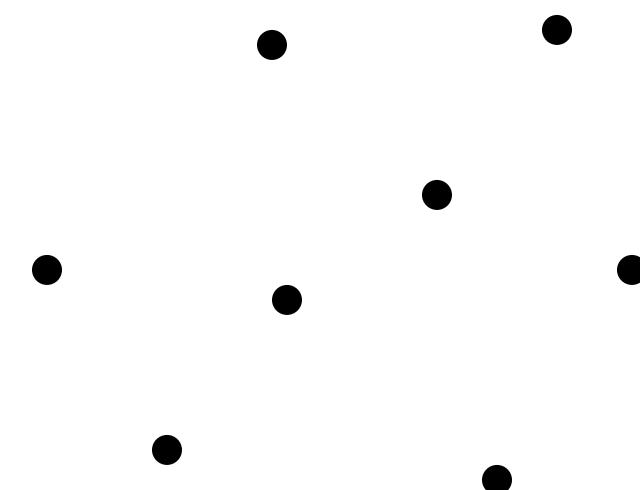
CONVEX HULL PROBLEM

- 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

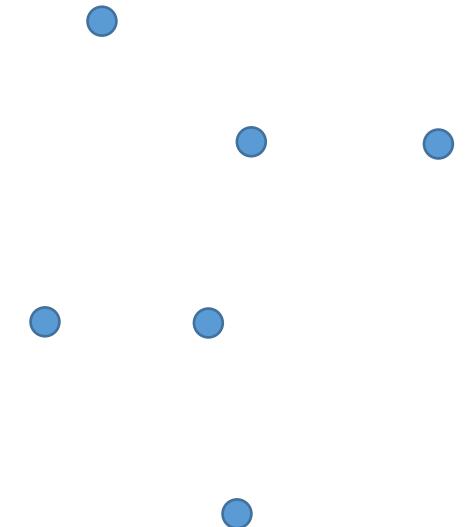
- THE CONVEX HULL OF A FINITE SET OF POINTS S IN THE PLANE IS THE SMALLEST CONVEX POLYGON P THAT ENCLOSES S , SMALLEST IN THE SENSE THAT THERE IS NO OTHER POLYGON P' SUCH THAT $P \supseteq P' \supseteq S$
- IDEAS?



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
```

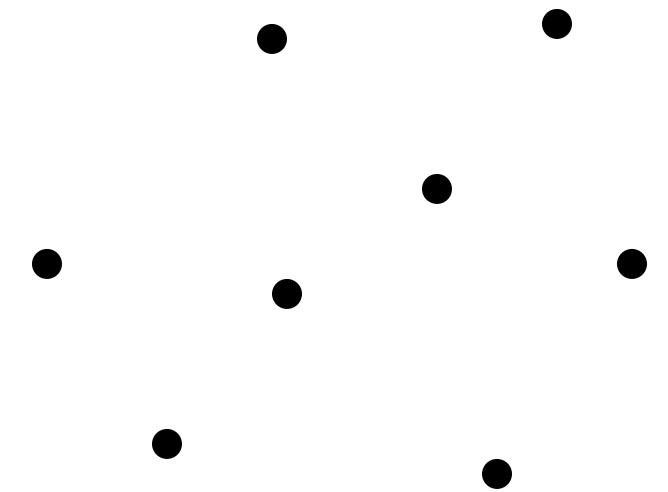


Performance? $O(n^4)$



DEFINITIONS OF CONVEXITY AND CONVEX HULL

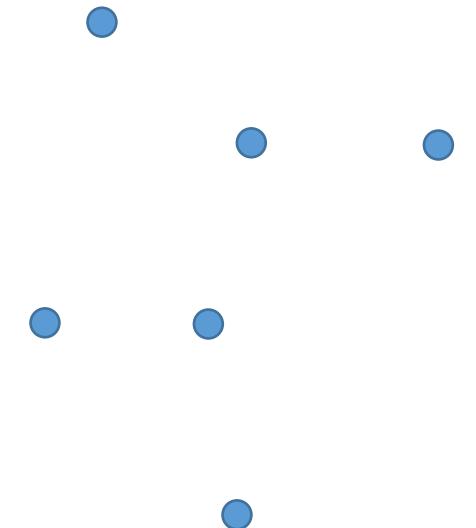
- THE CONVEX HULL OF A SET OF POINTS
 S IS ALSO INTERSECTION OF ALL HALF SPACES THAT CONTAIN S
- IDEAS?



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? $O(n^3)$



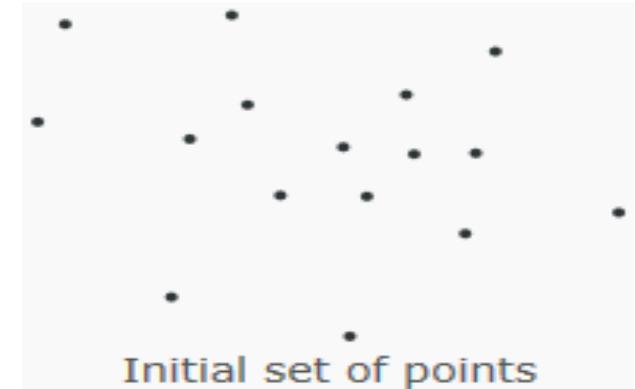
QUICKHULL

- 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

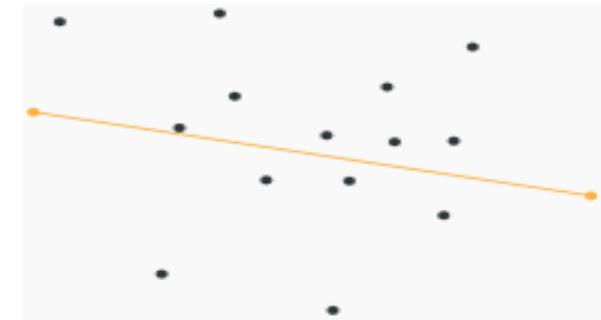


QUICKHULL ALGORITHM

- 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



Initial set of points

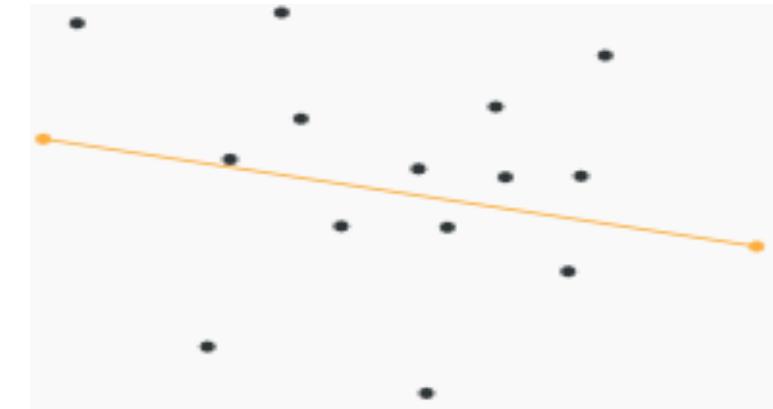


Min/Max horizontal points



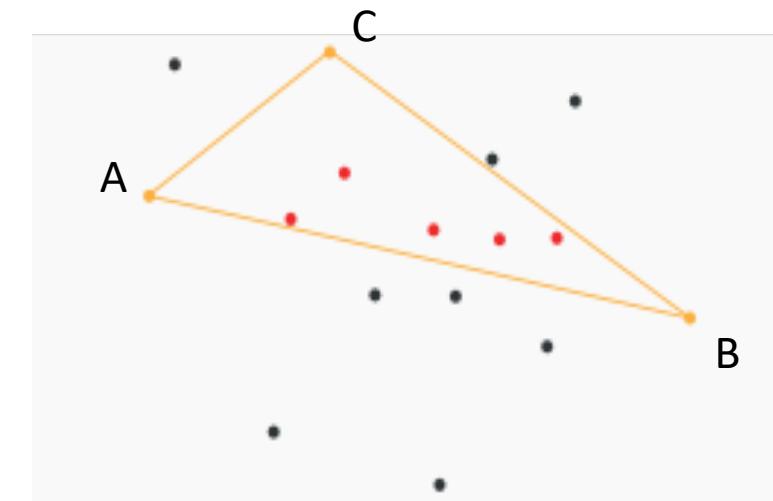
QUICKHULL ALGORITHM

- **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”**

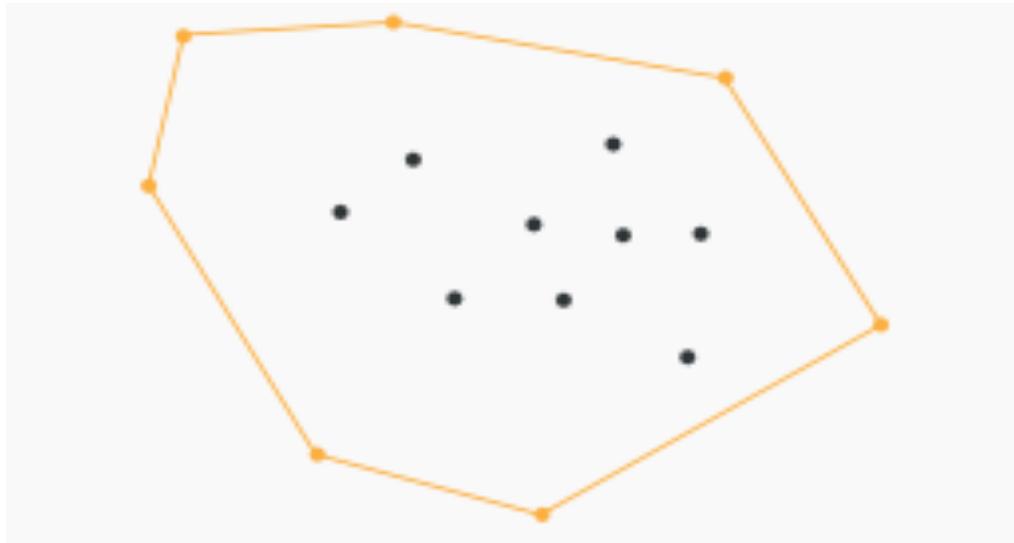


QUICKHULL ALGORITHM

- DISCARD ALL POINTS INSIDE Δ
- OPERATE RECURSIVELY ON THE POINTS OUTSIDE SEGMENTS AC AND BC

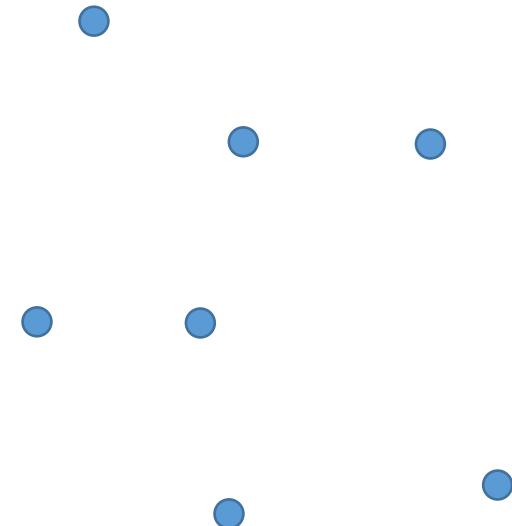


QUICKHULL ALGORITHM



QUICKHULL EXAMPLE

- FIND TWO DISTINCT EXTREME POINTS
- DIVIDE
- FIND EXTREME POINTS IN UPPER AND LOWER HULL
- DISCARD ALL POINTS INSIDE Δ
- RECURSE



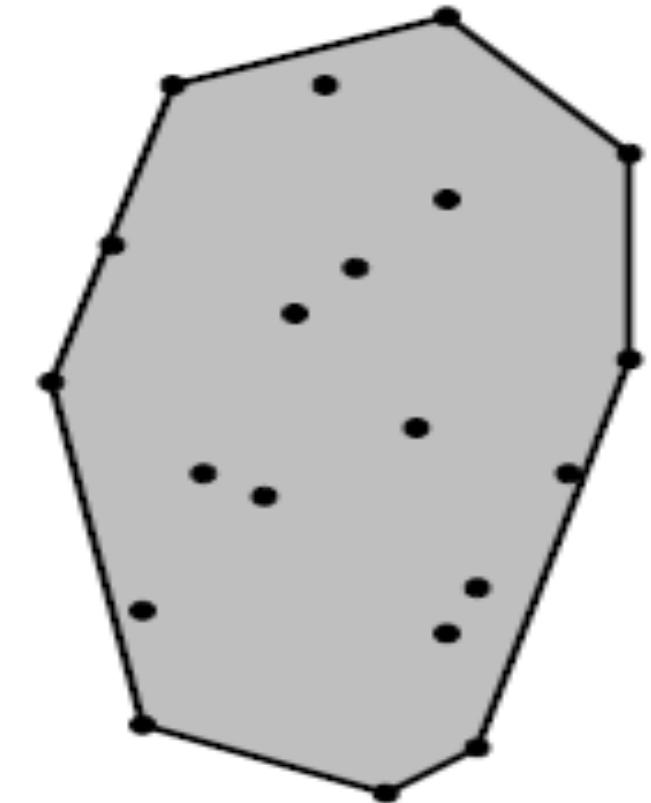
QUICKHULL PERFORMANCE

- 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 INTERIOR ANGLE IS STRICTLY CONVEX, LESS THAN π
- A SET OF POINTS S IS SAID TO BE STRONGLY CONVEX IF IT CONSISTS OF ONLY EXTREME POINTS
- IDEAS?



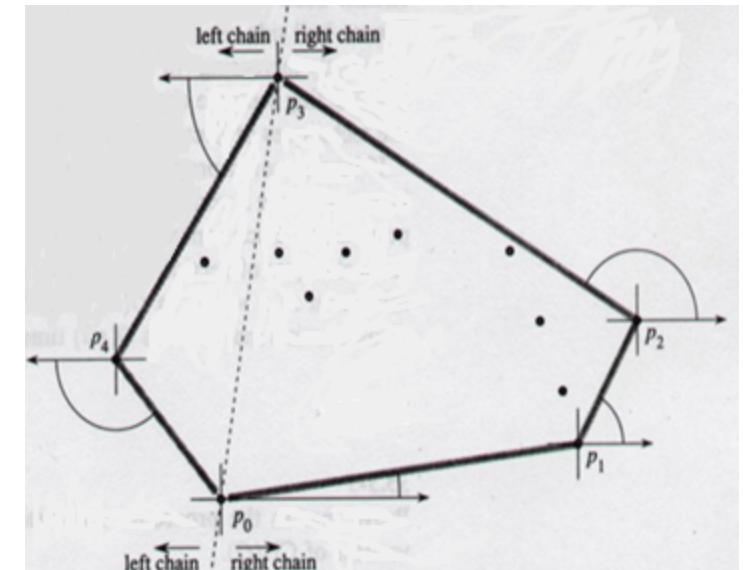
JARVIS'S ALGORITHM (GIFT-WRAPPING)

- 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



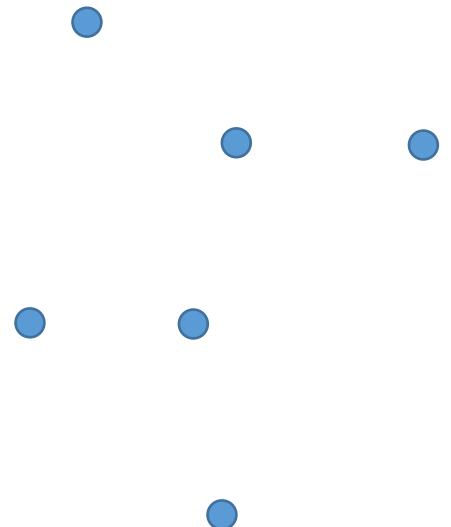
2D GIFT WRAPPING

- 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



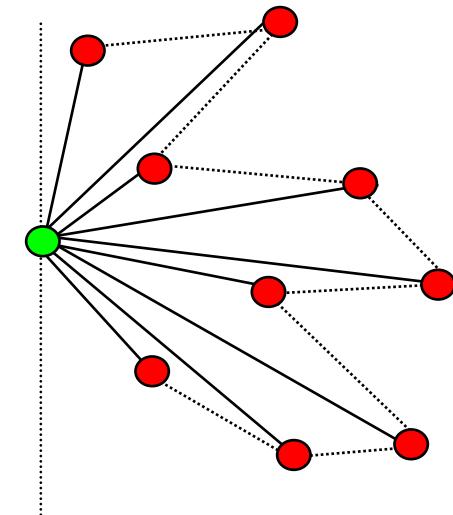
2D GIFT WRAPPING

- 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



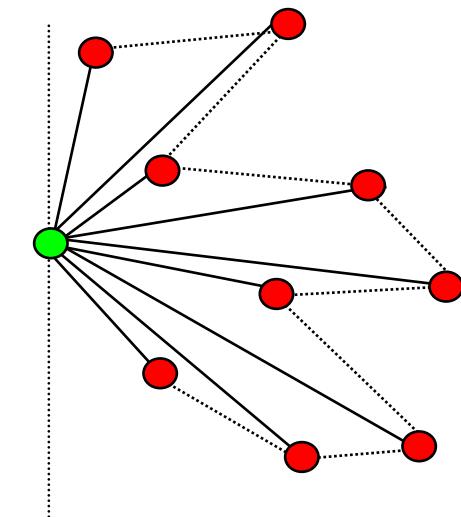
GRAHAM'S ALGORITHM

- 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)



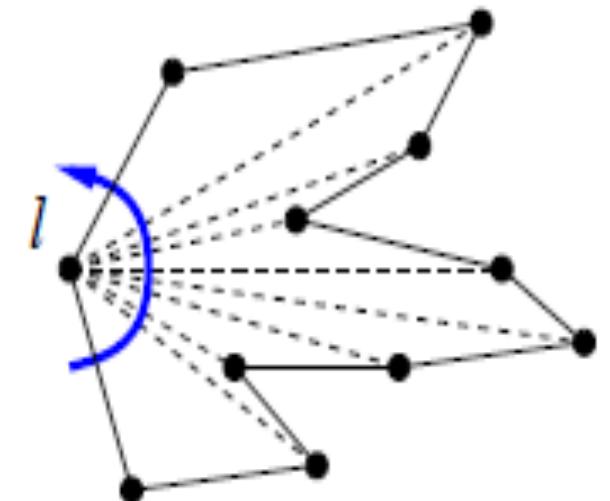
GRAHAM'S ALGORITHM

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



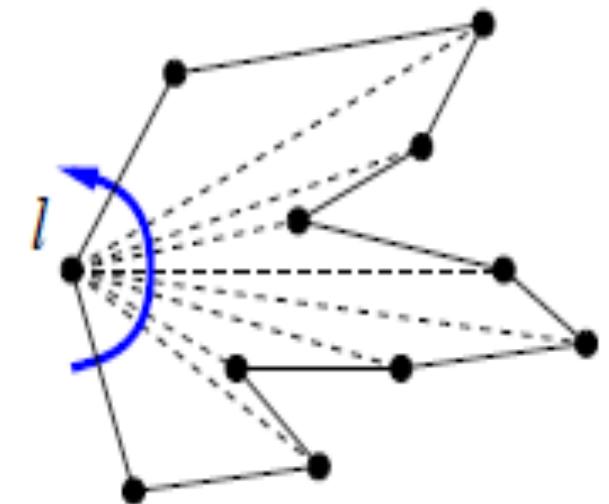
GRAHAM'S ALGORITHM

- 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.



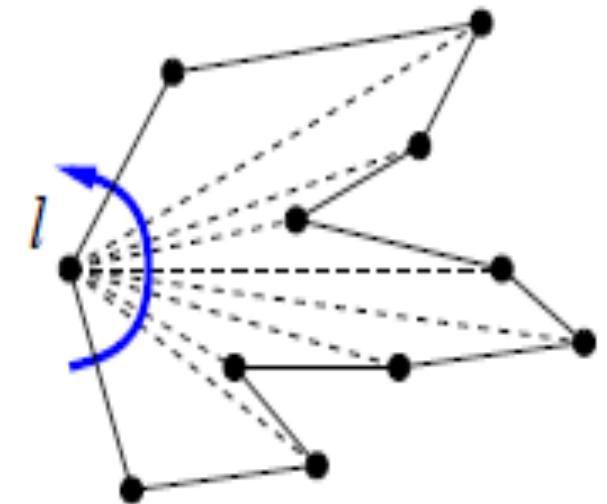
GRAHAM'S ALGORITHM

- 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

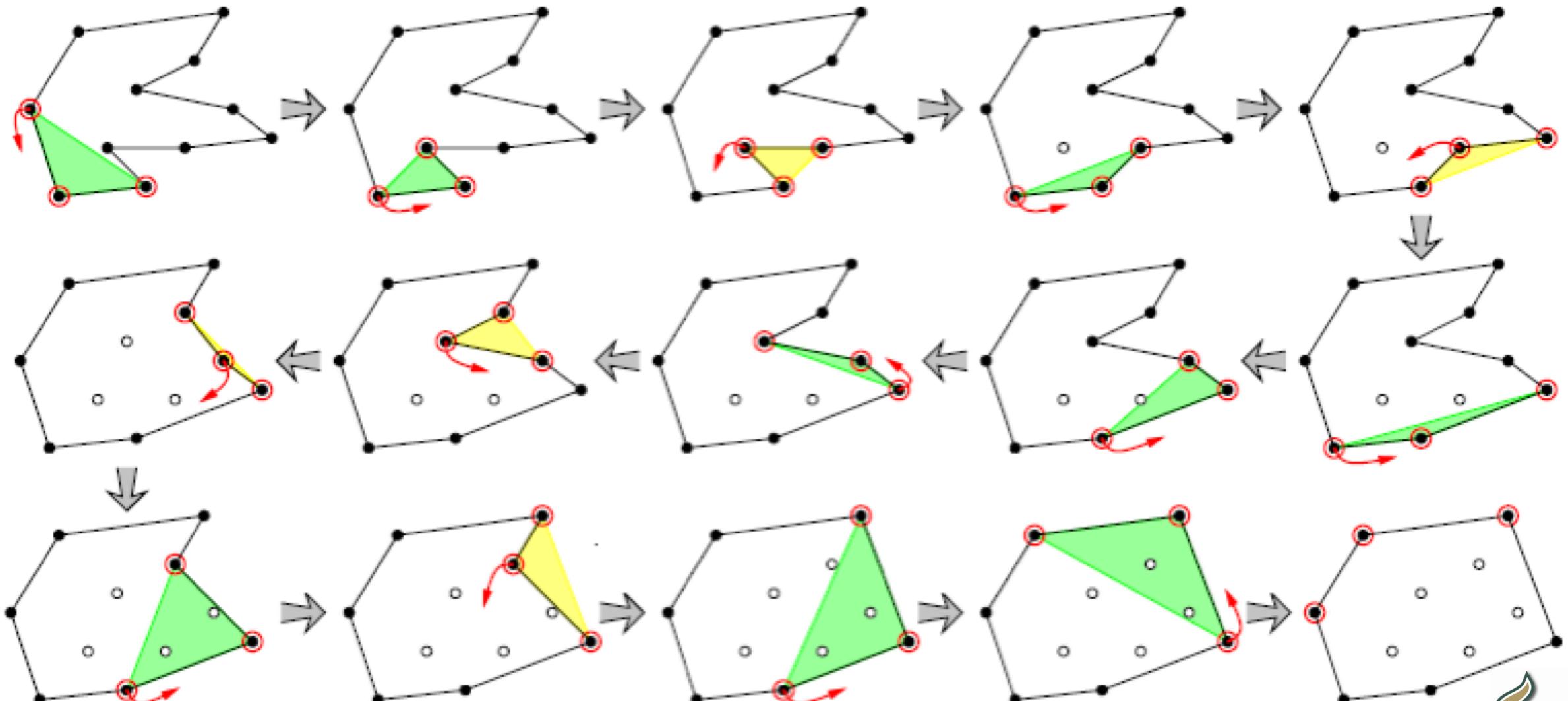


GRAHAM'S ALGORITHM

- **CHECK FOR CONVEXITY**
 - Pop the top 3 elements from the stack— $p; q; r$ of the polygon (at initially, these are the 2 vertices after I)
 - 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.

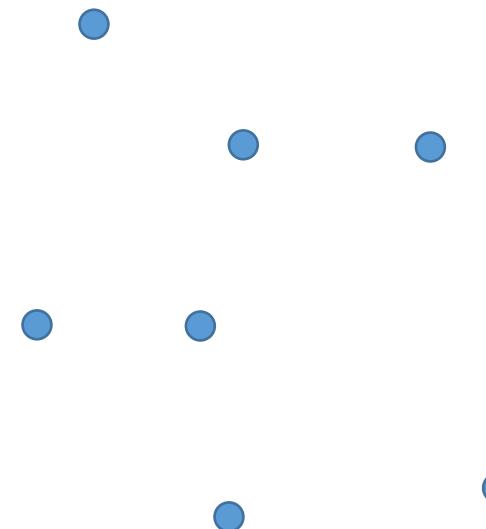


GRAHAM'S ALGORITHM EXAMPLE



GRAHAM SCAN EXAMPLE

- PLACE L AND THE 2 POINTS AFTER L ONTO THE STACK
- REPEAT UNTIL L
 - Place the 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.



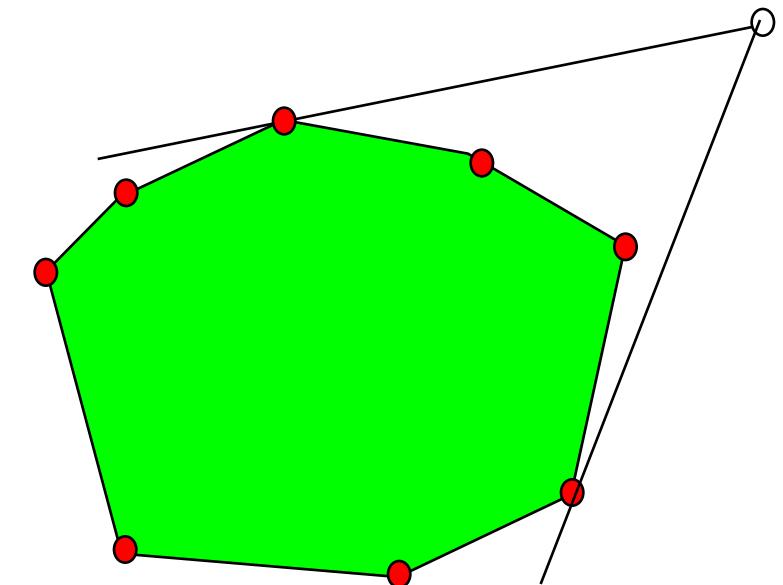
COMPLEXITY OF GRAHAM'S ALGORITHM

- 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: $O(N \log N)$



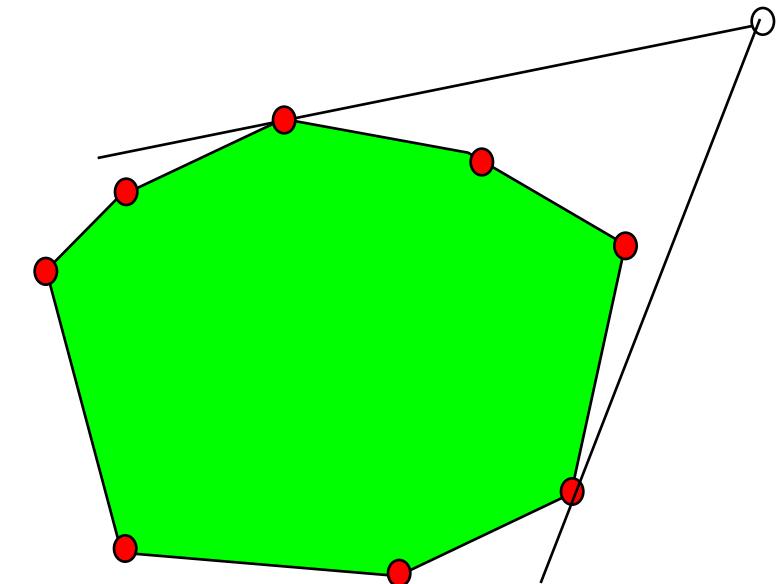
2D INCREMENTAL

- 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 TANGENTS?

- 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

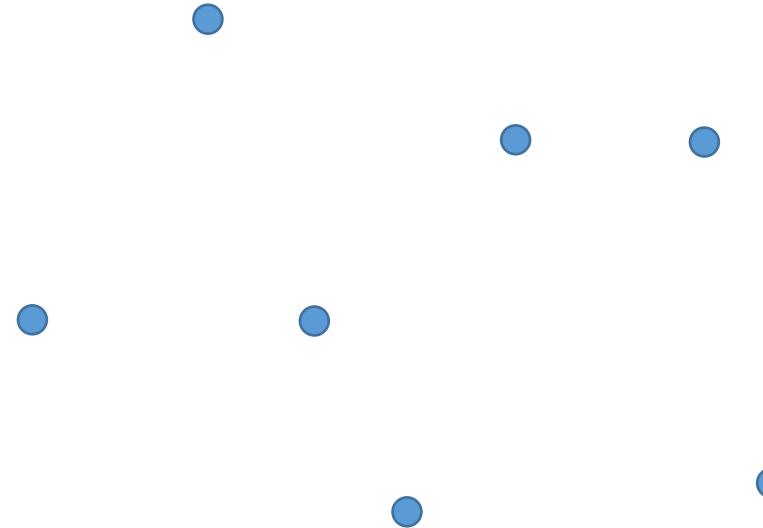


2D INCREMENTAL OPTIMAL

- 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 $f(n)$

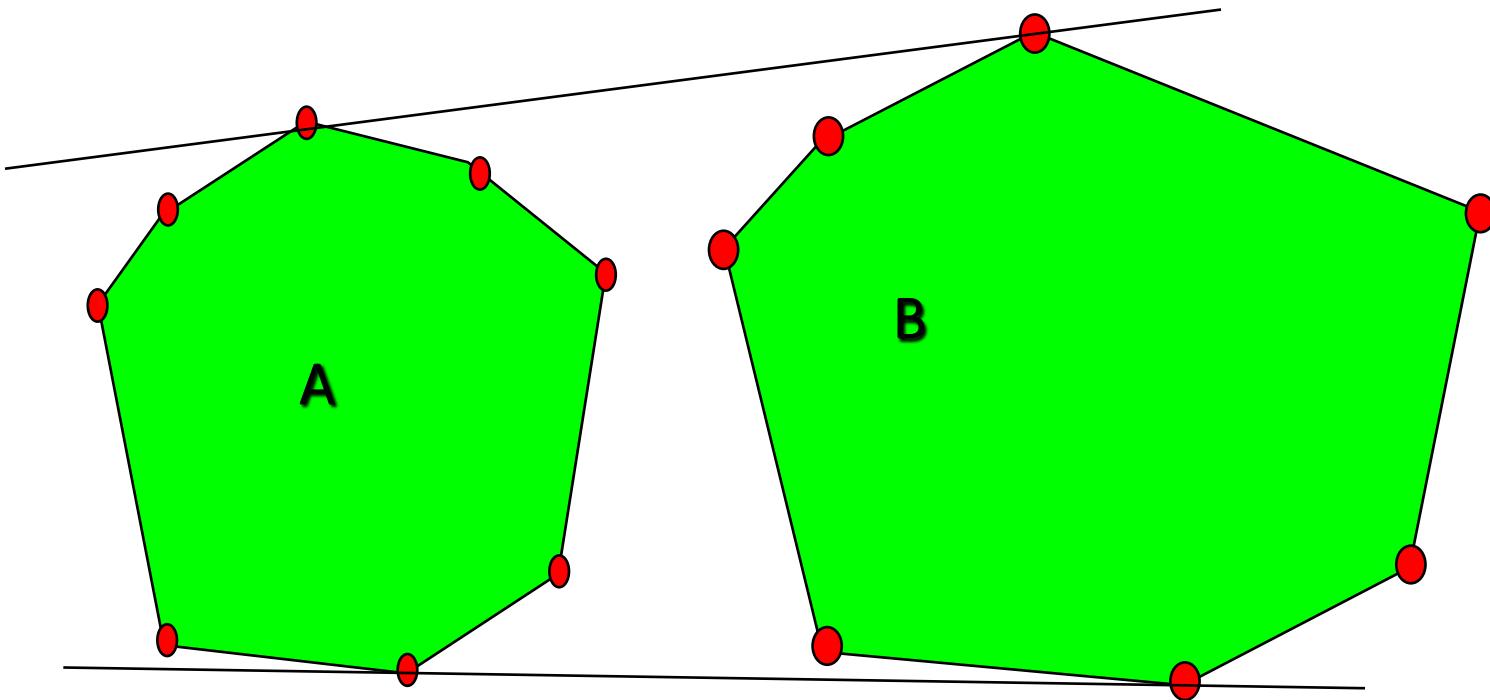


DIVIDE AND CONQUER

- 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?

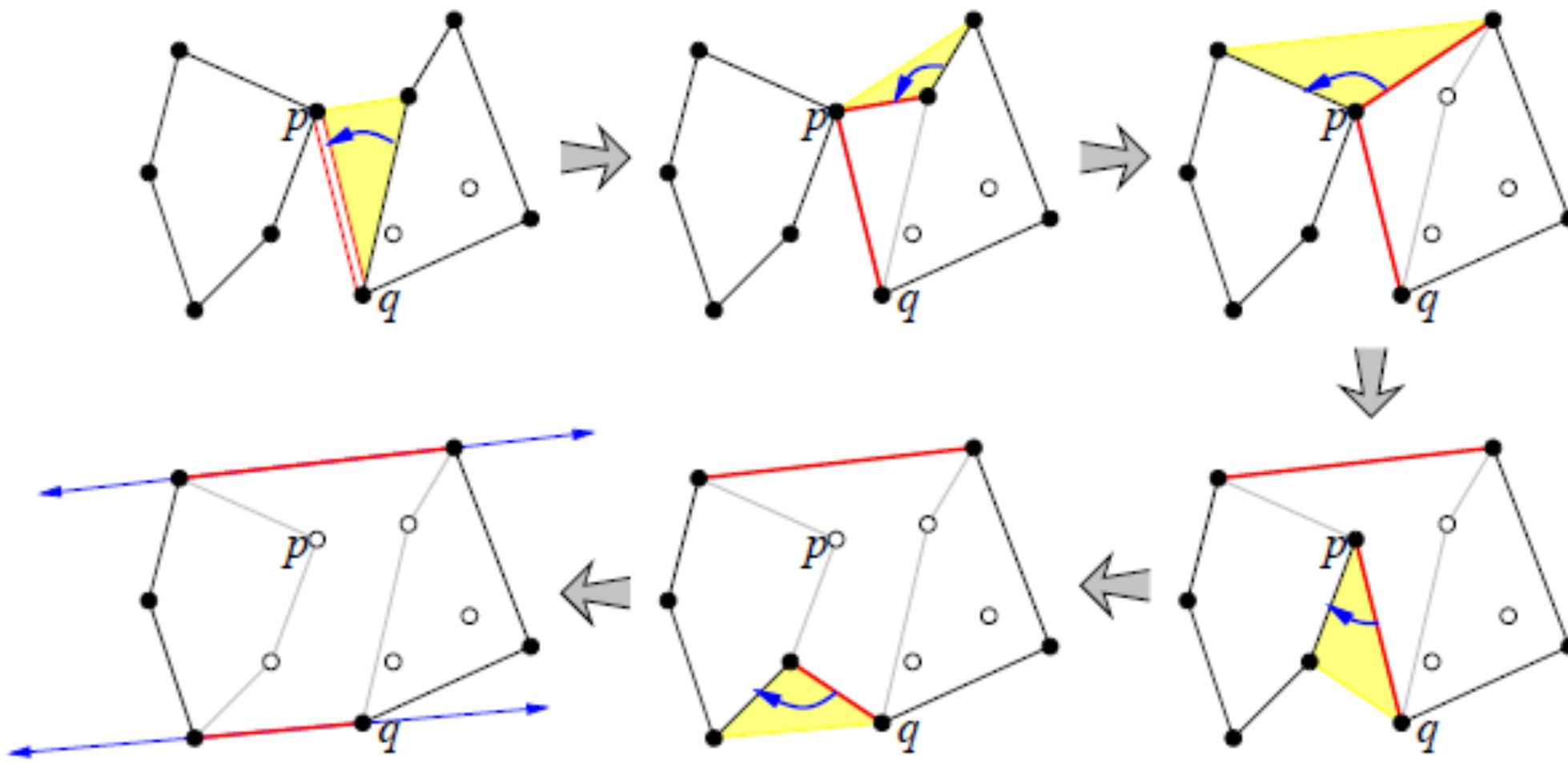


DIVIDE AND CONQUER (MERGING)

- 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



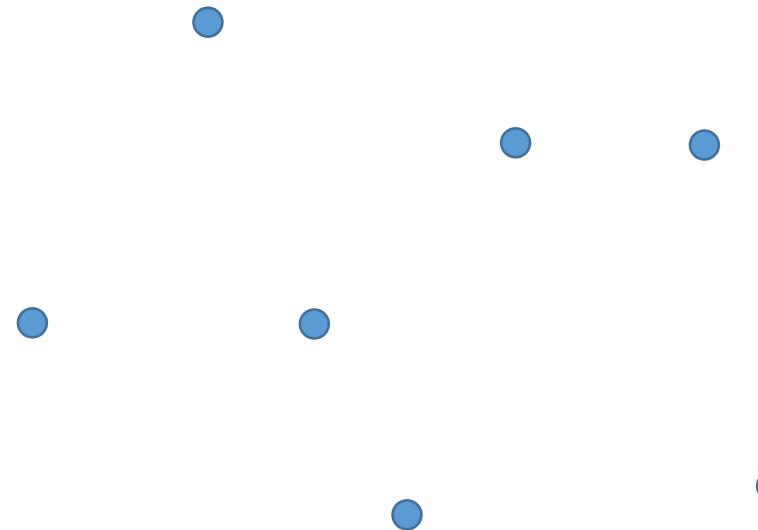
DIVIDE AND CONQUER (MERGE EXAMPLE)



Merging the left and right subhulls.



DIVIDE AND CONQUER EXAMPLE



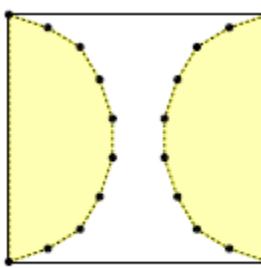
ANALYSIS OF DIVIDE AND CONQUER

- 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 EVERYTHING



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



ANALYSIS OF DIVIDE AND CONQUER

- MERGE SORT STYLE IMPLEMENTATION
 - Initial \times sorting takes $O(N \log N)$ 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 $O\left(\binom{n}{h} h \log h\right) = O(n \log h)$ TIME.

