Computational Geometry
Chapter 2: Convex Hull

Prof. Dr. Sándor Fekete

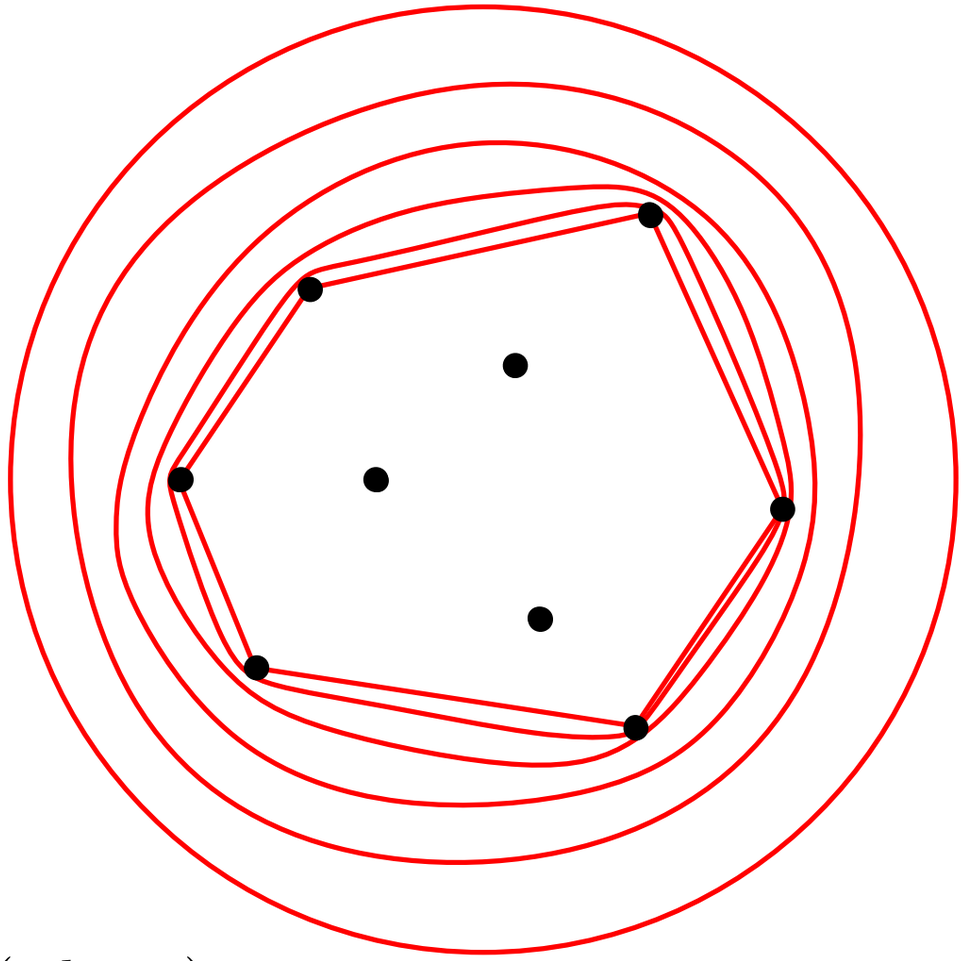
Algorithms Division
Department of Computer Science
TU Braunschweig

Technische Universität Braunschweig

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Task:

- Given: Set of n points in $\mathbb{R}^d$
- Wanted: Smallest enclosing convex object

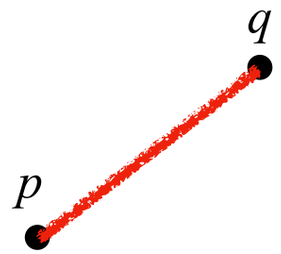


Intuition in $\mathbb{R}^2$:

- Draw points on a wooden board.
- Put in nails at points.
- Let a rubber band snap to the nails.

Theorem 2.18: Computing the convex hull takes $\Omega(n \log n)$.

Theorem 2.35: Computing the h vertices of the convex hull can be done in $O(n \log h)$.



Definition 2.1

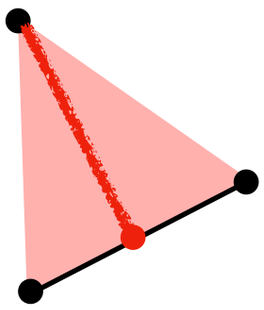
For $p, q \in \mathbb{R}^d : \overline{pq} := \{x \in \mathbb{R}^d \mid \exists \alpha, \beta \in \mathbb{R}, \alpha, \beta \geq 0, \alpha + \beta = 1, x = \alpha p + \beta q\}$

$p + \lambda(q - p), 0 \leq \lambda \leq 1$
$= (1 - \lambda)p + \lambda q$

Definition 2.2

For $\{p_0, \ldots, p_{n-1}\} \subset \mathbb{R}^d$ point $x \in \mathbb{R}^d$ is a convex combination of $\{p_0, \ldots, p_{n-1}\}$, if

$\exists \alpha_0, \ldots, \alpha_{n-1} \in [0, 1]$ with 1. $\sum_{i=0}^{n} \alpha_i p_i = x$



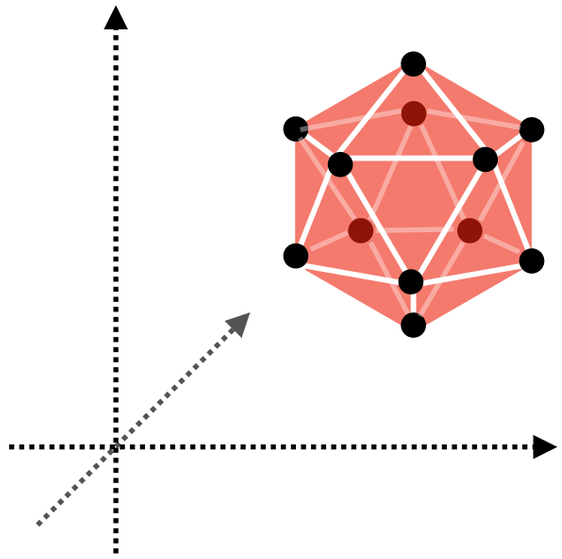
2. $\sum_{i=0}^{n} \alpha_i = 1$

- $\overline{pq} = \{x \mid x$ convex combination of $\{p, q\}\}$
- $\triangle(p, q, r) = \{x \mid x$ convex combination of $\{p, q, r\}\}$

Definition 2.3

Convex hull $conv(\mathcal{P})$ of $\mathcal{P} := \{p_0, \ldots, p_{n-1}\}$:

$$conv(\mathcal{P}) := \{x \in \mathbb{R}^d \mid x \quad \text{convex combination of} \quad \mathcal{P}\}$$

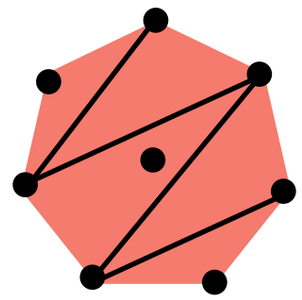


Theorem 2.4 (Carathéodory)

$conv(\mathcal{P}) =$ Union of all convex combinations with at most $(d+1)$ points in $\mathcal{P}$

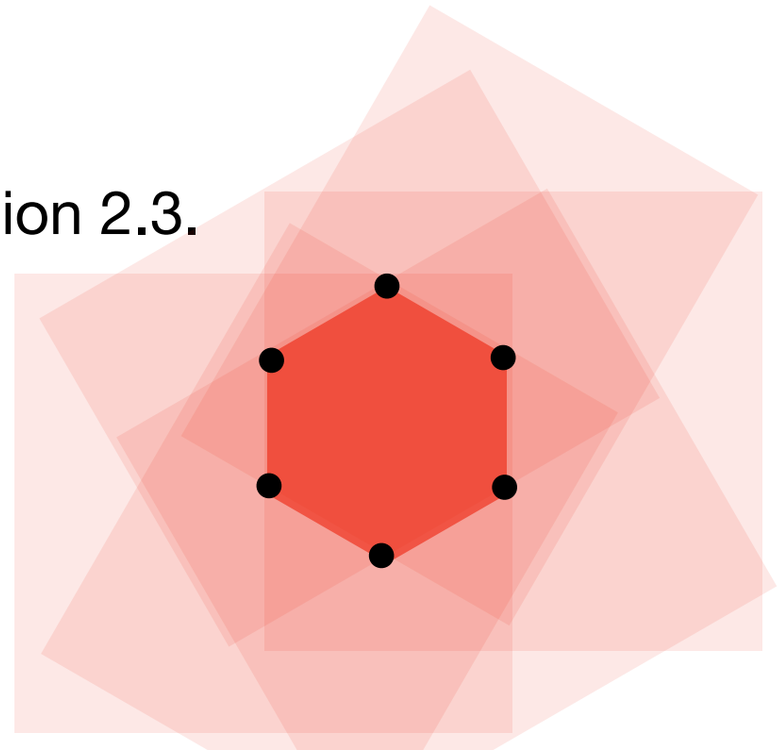
Corollary 2.5

$\mathcal{P} \subset \mathbb{R}^2 \Rightarrow conv(\mathcal{P})$ union of all $\triangle(p,q,r)$ with $p,q,r \in \mathcal{P}$.



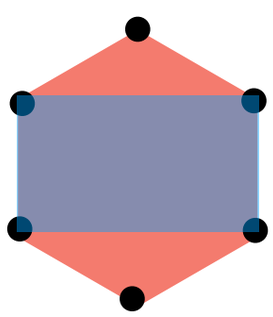
Lemma 2.6

The following definitions are equivalent to Definition 2.3.

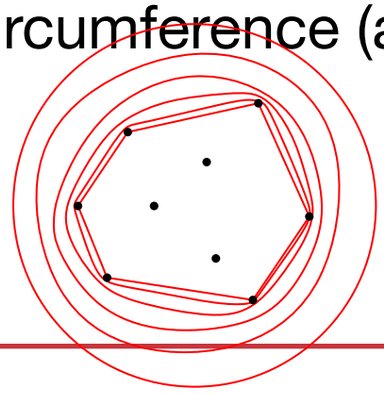


1. $conv(\mathcal{P}) := \bigcap_{P \supset \mathcal{P}, P \text{ convex}} P$

2. For $d = 2$: $\nexists$ convex polygon $P : \mathcal{P} \subseteq P \subsetneq \text{conv}(\mathcal{P})$



3. For $d = 2$: $conv(\mathcal{P}) :=$ conv. polygon P with minimal circumference (area) with $\mathcal{P} \subset P$

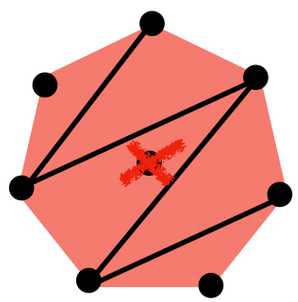


- From now on $\mathcal{P} \subset \mathbb{R}^2$
- First Approach: Find vertices of $conv(\mathcal{P})$ by elimination
- Negation of Corollary 2.5:

x not vertex of $conv(\mathcal{P})$

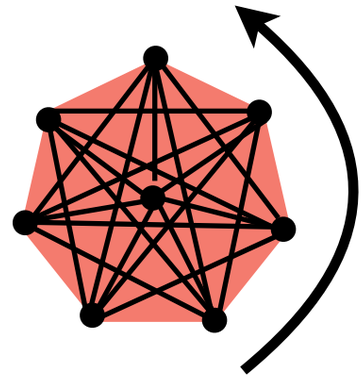
$\Updownarrow$

$\exists p_i, p_j, p_k \in \mathcal{P} : x$ = non-trivial convex combination of p_i, p_j, p_k.



Algorithm 2.7

1: for (all triples (p_i, p_j, p_k) of points in $\mathcal{P}$) do
2: for (all points in $\in \mathcal{P}$) do
3: if (p lies in the inside of $\Delta(p_i, p_j, p_k)$
or on a boundary edge of $\Delta(p_i, p_j, p_k)$) then
4: mark p as an interior point.
5: $\mathcal{P}' := \{p \in \mathcal{P} \mid$ is unmarked $\}$;

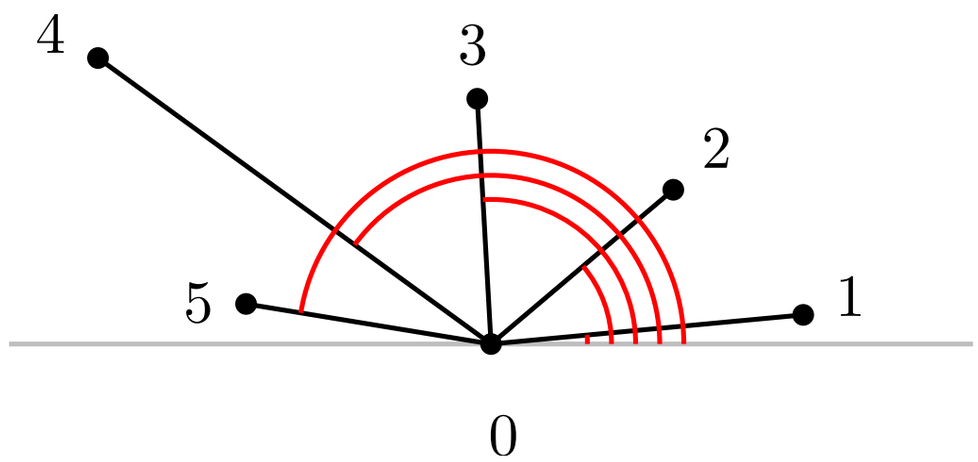


Analysis of Algorithm 2.7:

- $\binom{n}{3} \in \Theta(n^3)$ triangles
- Per triangle: $\Theta(n)$ further points
- Sort $\mathcal{O}(n)$ vertices
- Total runtime: $\mathcal{O}(n^4 + n \log n) = \mathcal{O}(n^4)$

Sorting criterion:

- CCW on $conv(\mathcal{P})$.
- Polar angle wrt y-minimal point in $\mathcal{P}$.
- Issue: Do we need trigonometry?



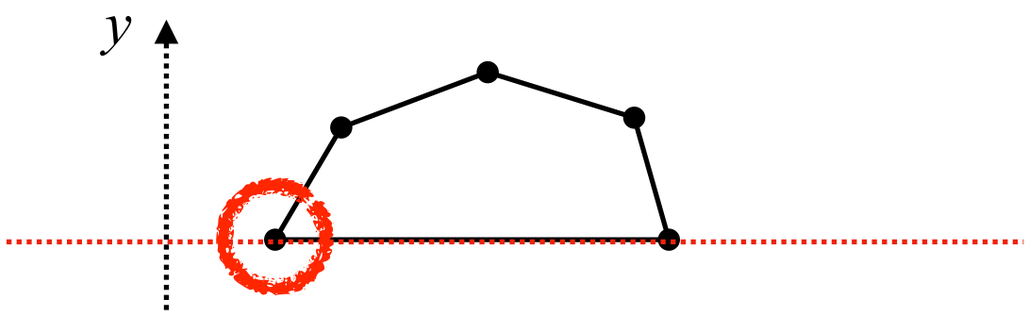
Lexicographic order

- Choose the „y-minimal“ point by lexicographic order:

$$(p.x, p.y) \leq_y (q.x, q.y) :\Leftrightarrow ((p.y < q.y) \vee ((p.y = q.y) \wedge (p.x \leq q.x)))$$

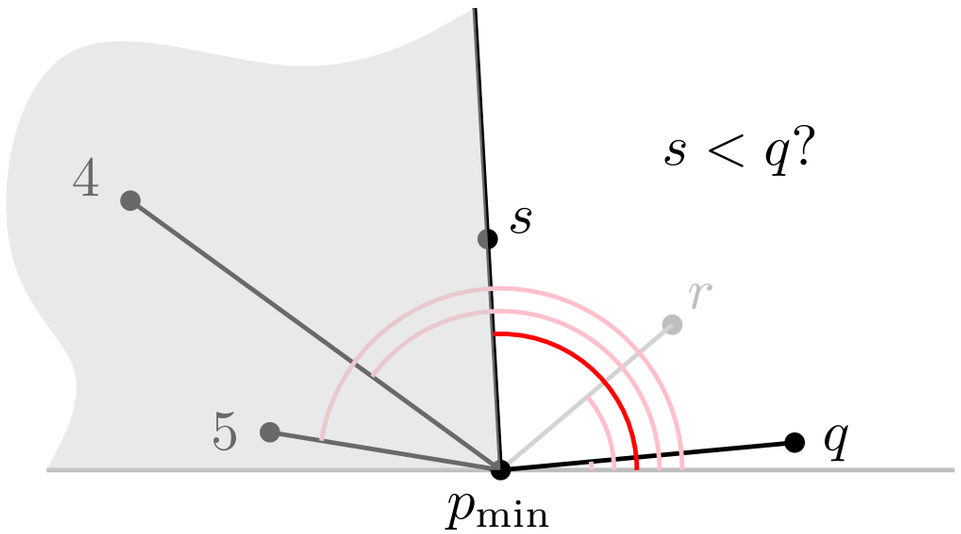
- Analogously: by x-coordinate

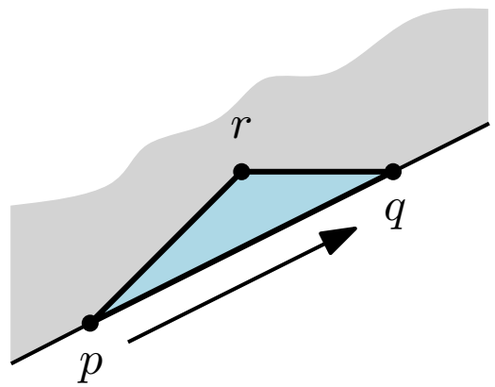
$$(p.x, p.y) \leq_x (q.x, q.y) :\Leftrightarrow ((p.x < q.x) \vee ((p.x = q.x) \wedge (p.y \leq q.y)))$$



Observation:

- Goal: CCW order
- Sort by polar angle.
- Sufficient: pairwise comparion of points s, q
- Check relative position of q wrt $\overline{p_{\min} s}$





Consequence:

- Predicate: $a \leq b :\Leftrightarrow ((a = p_{\min}) \vee (a = b) \vee (\textsc{LeftTurn}(p_{\min}, a, b) = \textsc{true}))$

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Review: Sorting

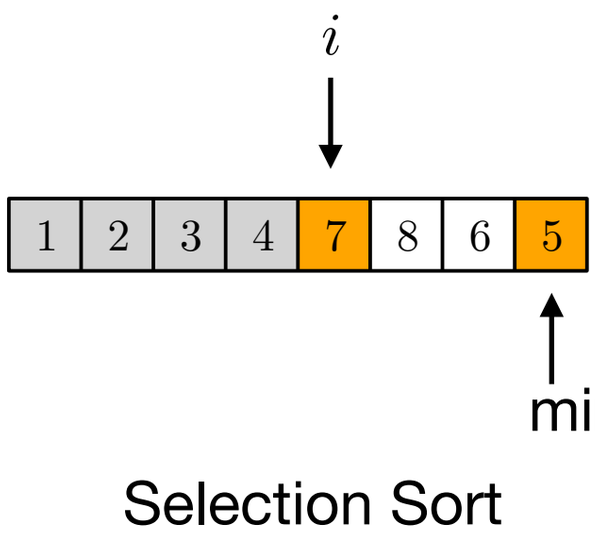
- Algorithms and Data Structures 1
- Various algorithmic paradigms

Sorting algorithms:

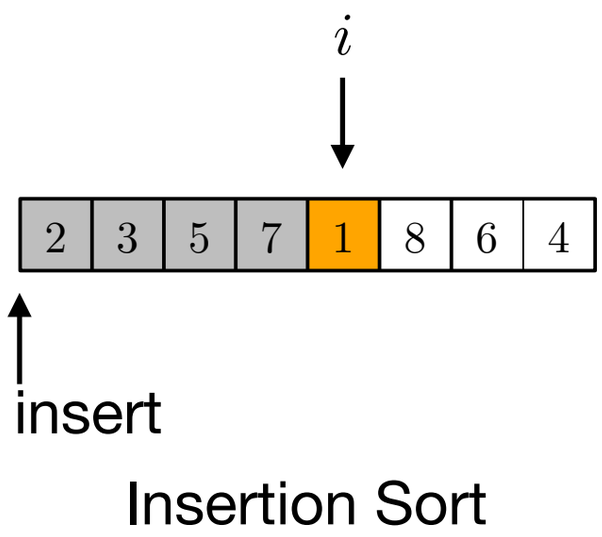
- Incremental methods:
  - Bubble Sort, Selection Sort, Insertion Sort
- Divide-and-conquer methods:
  - Quicksort, Mergesort
- Methods based on data structures:
  - Heapsort, sorting by AVL tree
- Other methods:
  - Bucket Sort, Shellsort, …

Difference: Selection Sort <-> Insertion Sort

- Selection Sort: Search in *unsorted* part of array
- Insertion Sort: Search in *sorted* part of array



Selection Sort

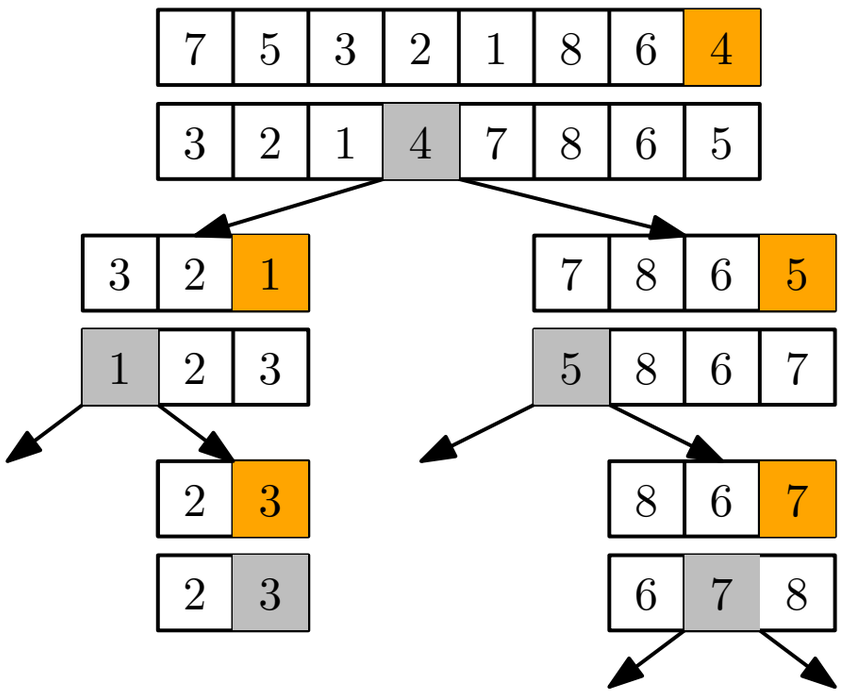


Insertion Sort

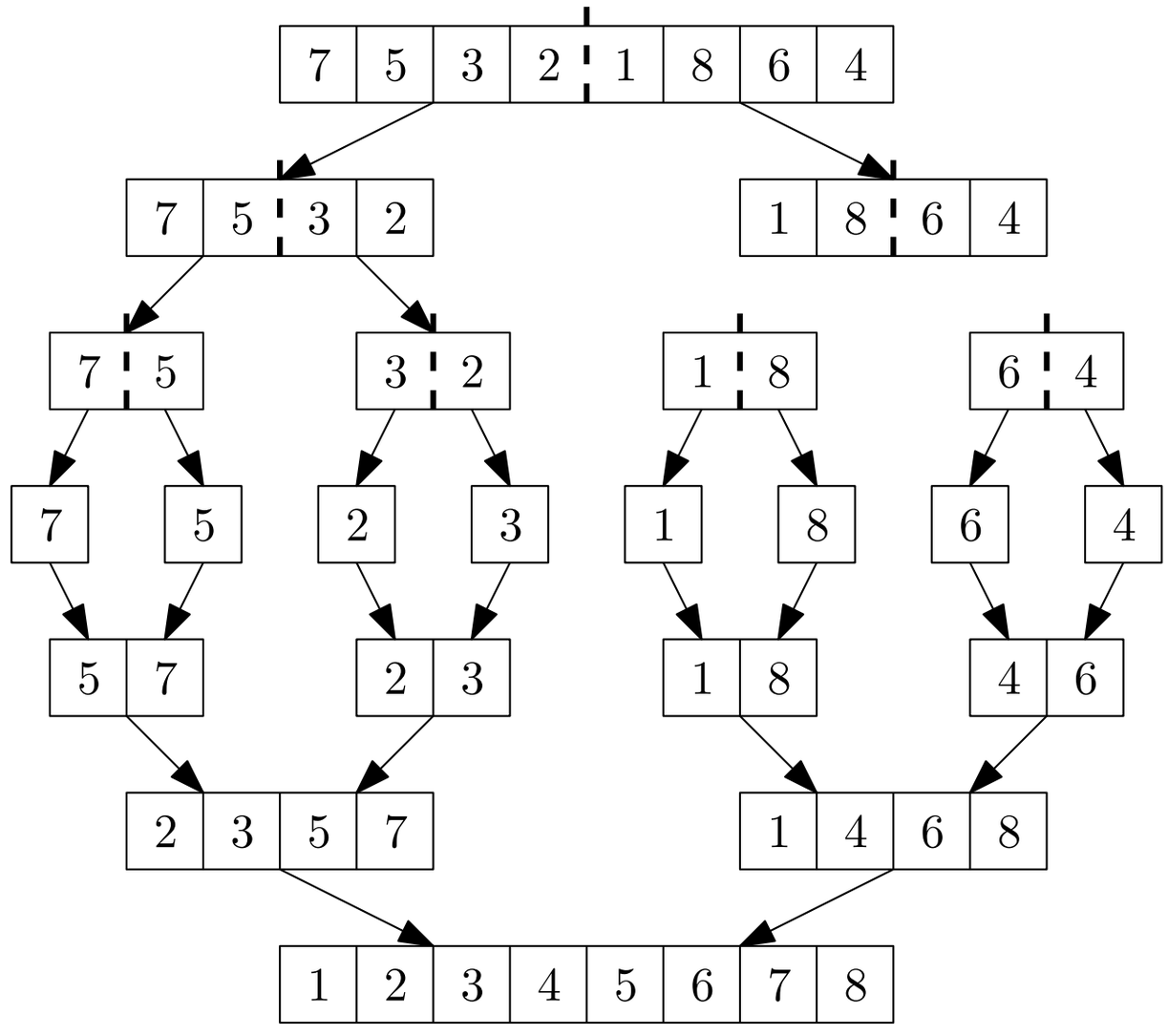
Approach:

- Selection Sort: Find next element for extending the order
- Insertion Sort: Insert next element, such that sequence remains sorted

- Split (Quicksort) or combine (Mergesort)

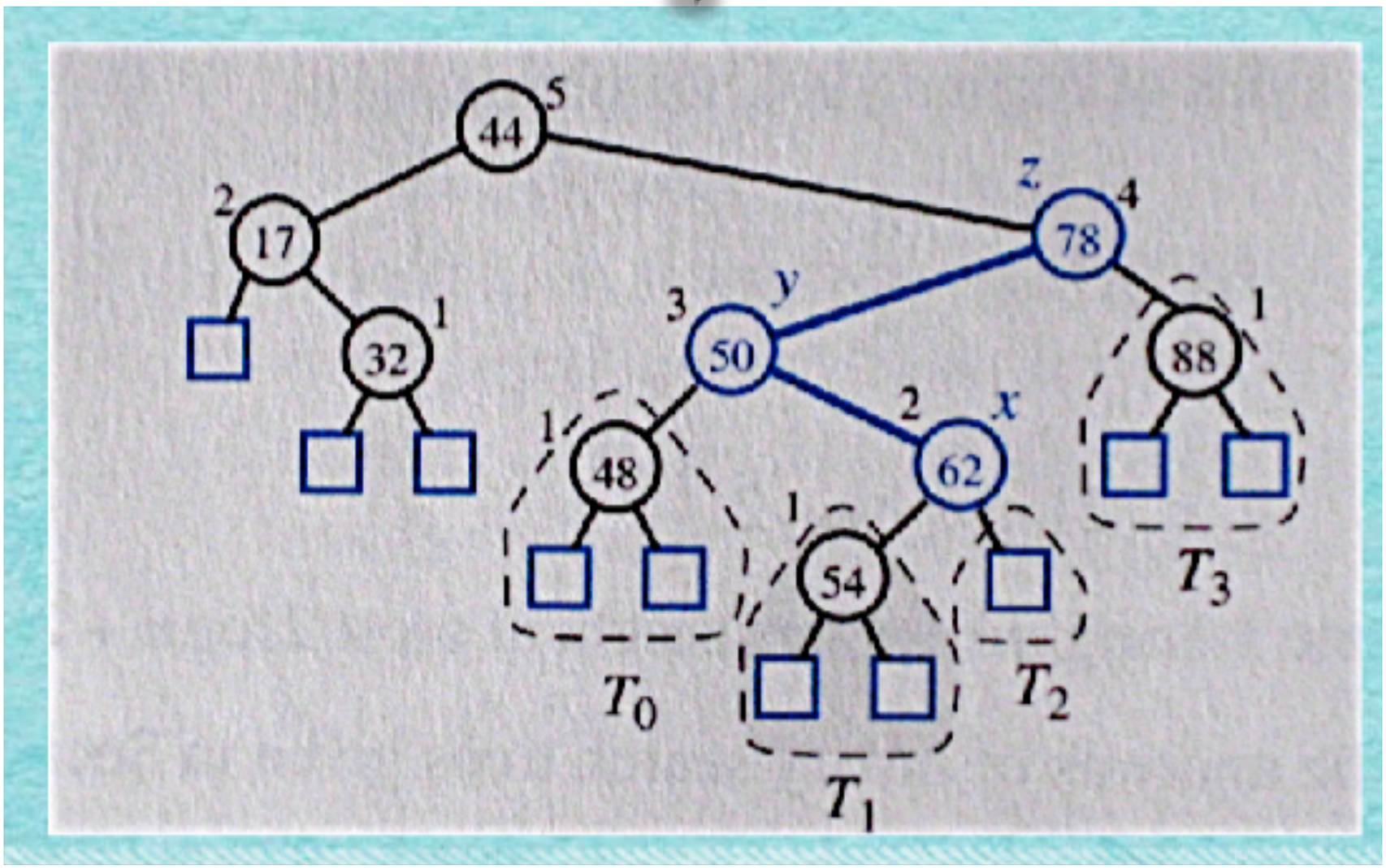


Quicksort



Mergesort

„Algorithms and Data Structures“: AVL-tree

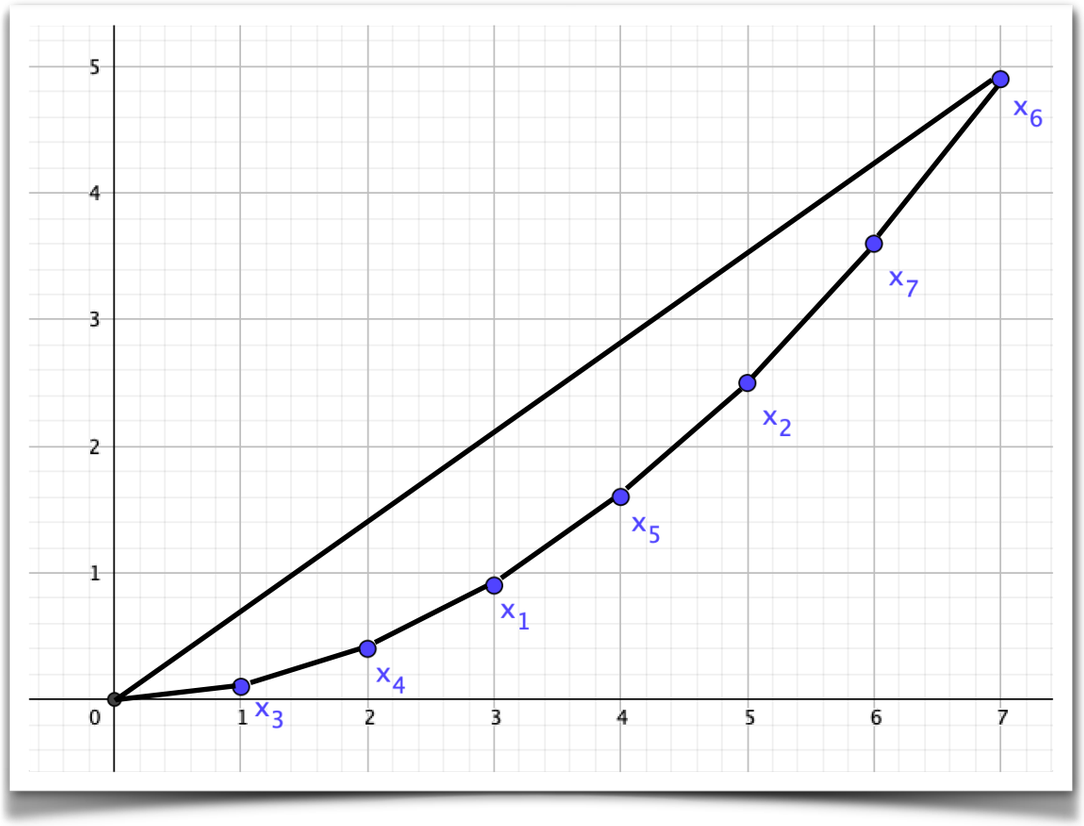


Theorem 1.18: Computing the convex hull takes $\Omega(n \log n)$ in certain models of computation.

Proof: Recall that comparison-based sorting takes $\Omega(n \log n)$.

Consider a set of n numbers, $x_1, \ldots, x_n$.

Map them to the points $(x_1, x_1^2) \ldots, (x_n, x_n^2)$.



3, 5, 1, 2, 4, 7, 6

The convex hull yields the sorted order of numbers.

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

INFORMATION PROCESSING LETTERS 2 (1973) 18--21. NORTH-HOLLAND PUBLISHING COMPANY

ON THE IDENTIFICATION OF THE CONVEX HULL OF A FINITE SET OF POINTS IN THE PLANE

R.A. JARVIS

The Australian National University, Department of Statistics, Box 4, Canberra, A.C.T. 2600, Australia

Received 6 December 1972

convex hull algorithm

1. Introduction

This paper presents an extremely simple algorithm for identifying the convex hull of a finite set of points in the plane in essentially, at most $n(m+1)$ operations for n points in the set and $m \leqslant n$ points on the convex hull. In most cases far less than $n(m+1)$ operations are necessary because of a powerful point deletion mechanism that can easily be included. The operations are themselves trivial (computationally inexpensive) and consist of angle comparisons only. Even these angle comparisons need not be actually carried out if an improvement suggested in a later section is implemented. Although Graham's algorithm [1] requires no more than $(n \log n)/\log 2 + Cn$ operations*, the operations are themselves more complex than those of the method presented here; in particular, Graham's method would not be as efficient for low m.

2. Geometric interpretation

The underlying method of the algorithm can be described simply: find an origin point outside the point set and swing a radius arm in an arbitrary direction until a point of the set is met; this point becomes

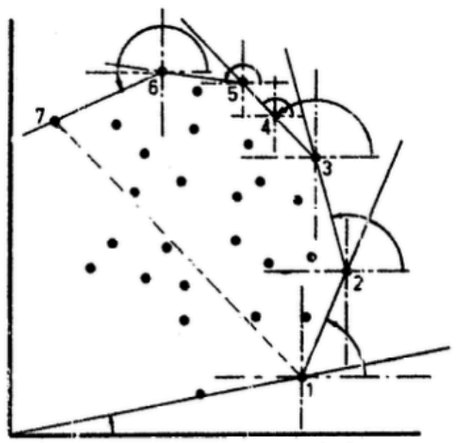


Fig. 1. Geometric interpretation of the algorithm.

the first point on the hull. Make this the new origin point and swing a radius arm from this point in the same direction as before till the next hull point is found. Repeat until the points are enclosed by the convex hull. Delete points from further consideration if

(i) they have already been identified as being on the convex hull,

(ii) they lie in the area enclosed by a line from the first to the last convex hull point found and the lines joining the convex hull points in the sequence found.

Fig. 1 illustrates this geometric interpretation.

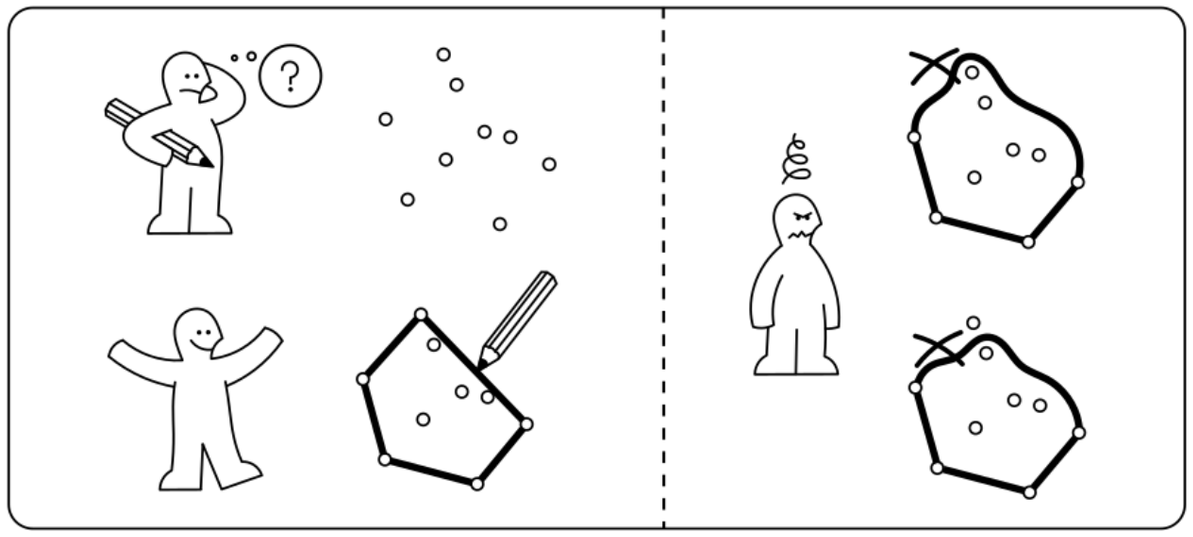
* To quote Graham, "C is a small positive constant which depends on what is meant by an 'operation'". In fact, C is distributed over the five basic steps of Graham's algorithm and his paper should be consulted for detailed interpretation.

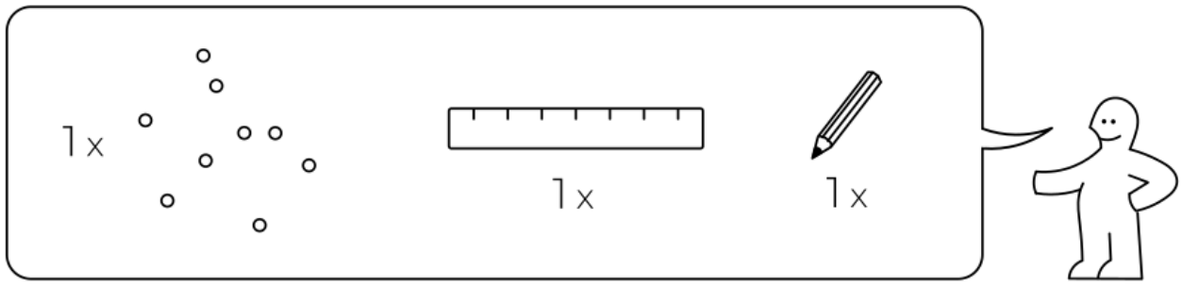

IDEA
A series of nonverbal
algorithm assembly instructions.

GIFT WRÄPPING

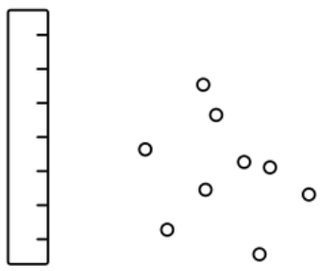
idea-instructions.com/gift-wrapping/
Based on a guest contribution by Christoph Hansknecht – v1.1, CC by-nc-sa 4.0

IDEA

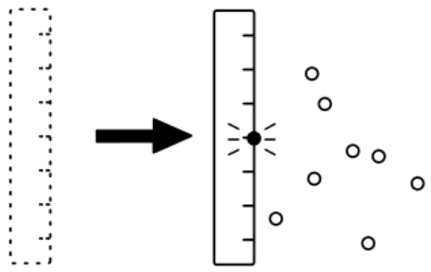




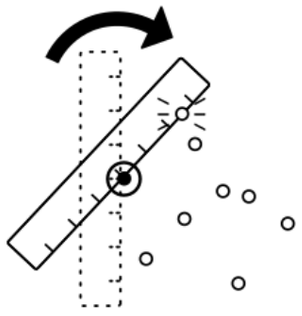
1a



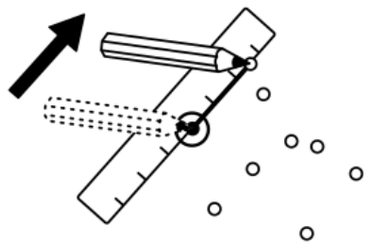
1b



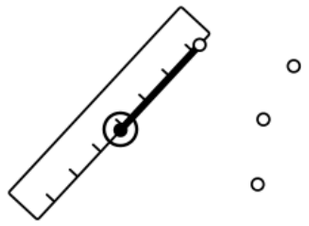
1c



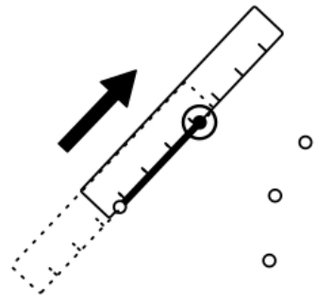
1d



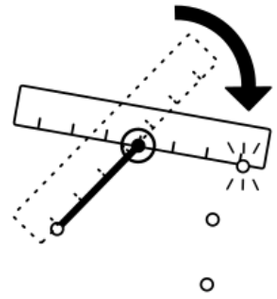
2a



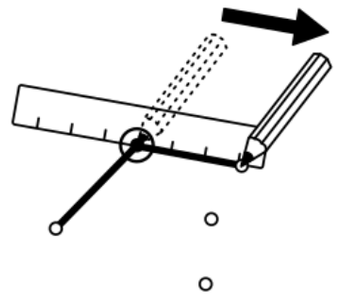
2b



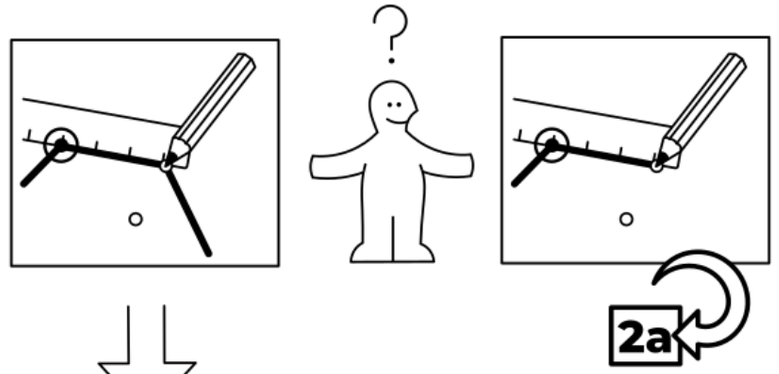
2c



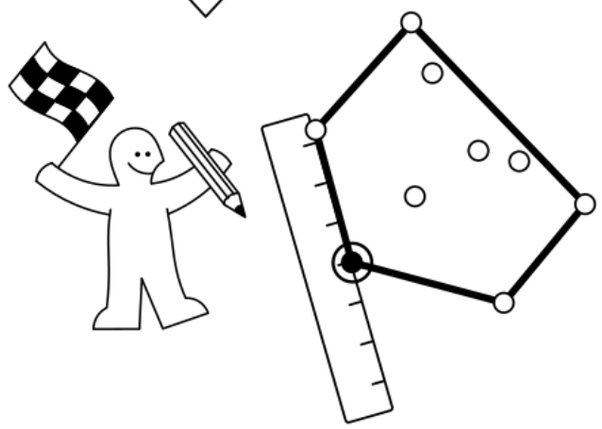
2d



3

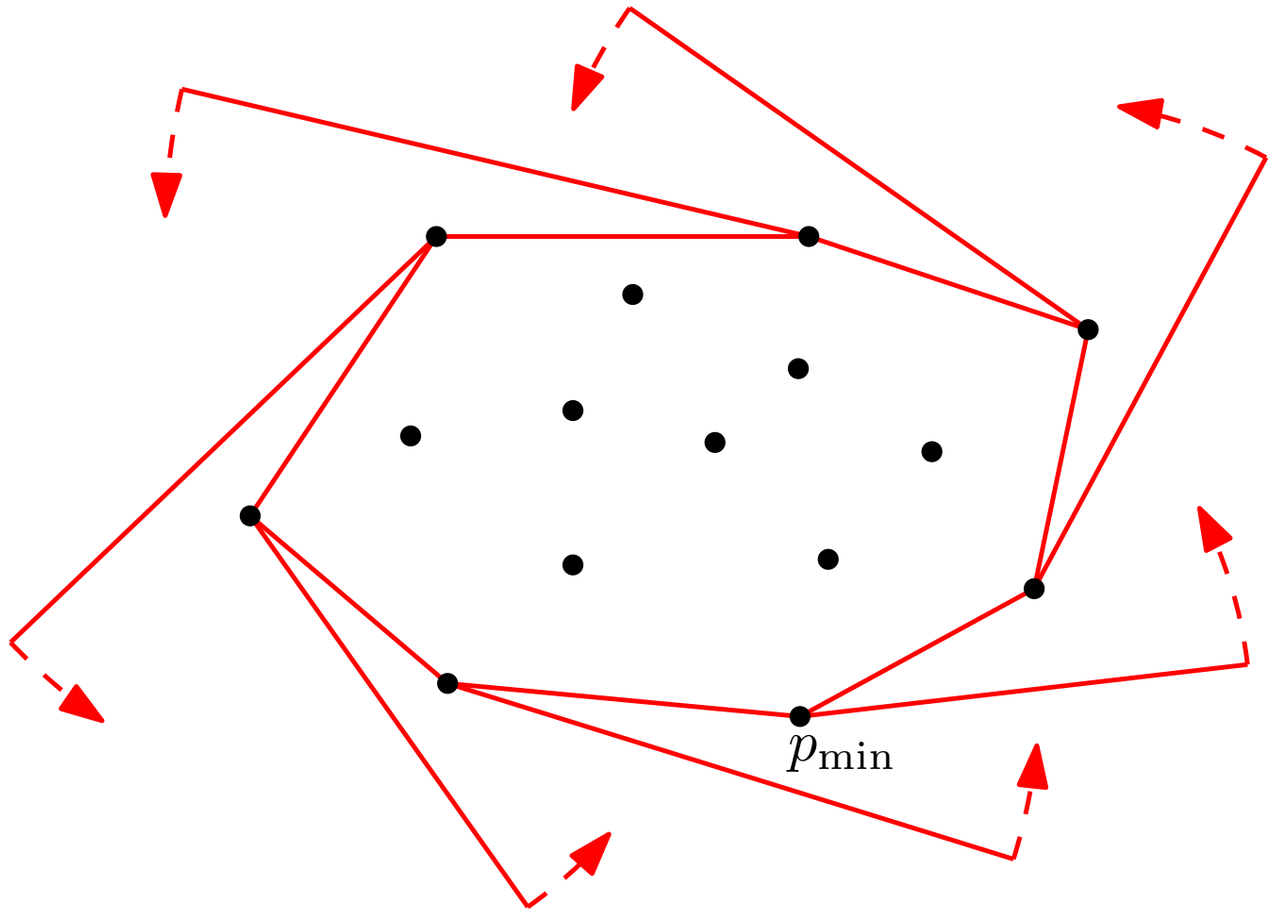


4



Basic idea:

- Iteratively find next edge on boundary of $conv(\mathcal{P})$
- Analogy: Selection Sort.
  - Find next element for continuing sorted order
- Start: minimal point $p_{\min}$ wrt $\leq_y$



Intuition:

- „Gift wrapping“

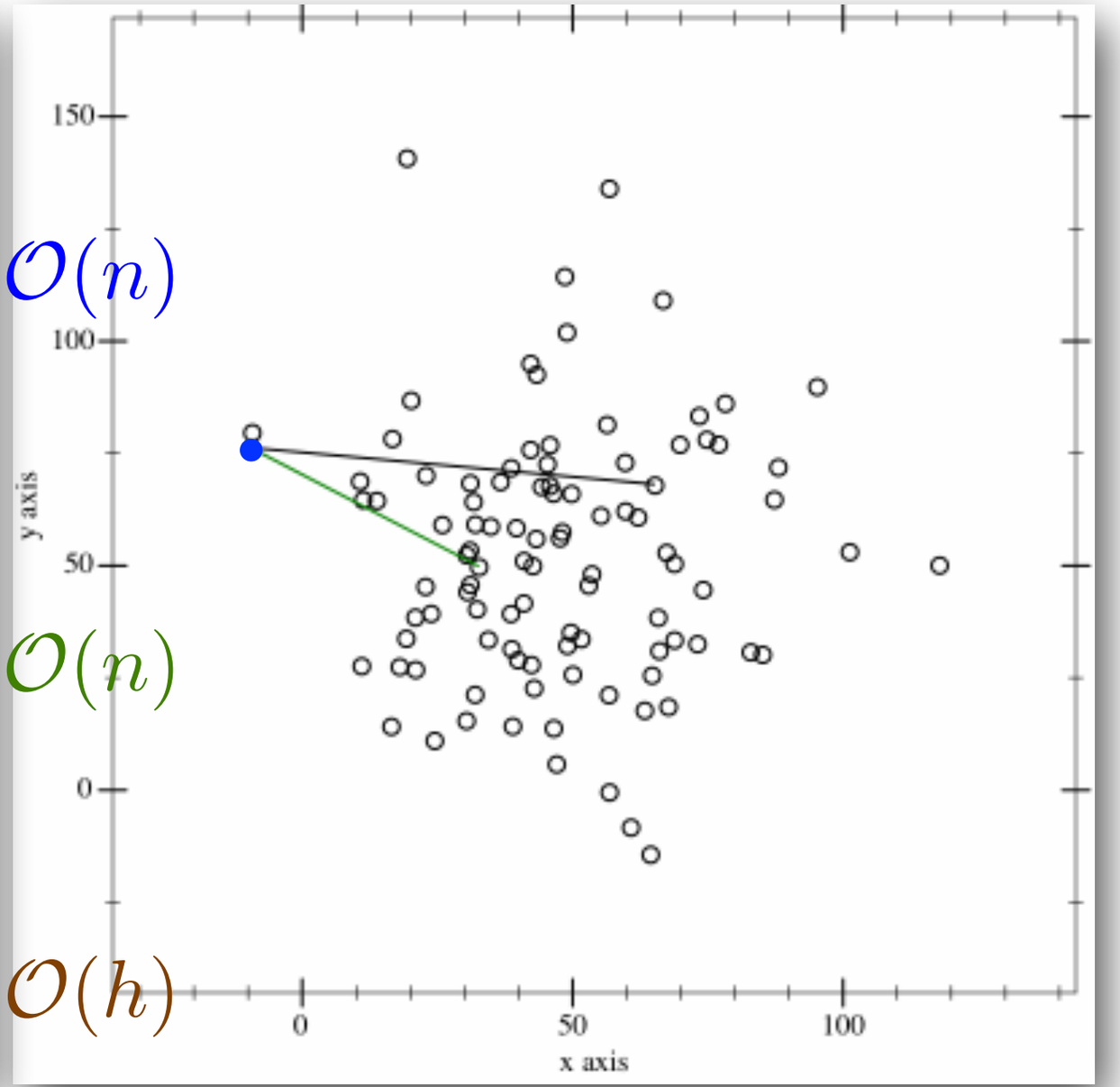
Algorithm 2.9: Compute $conv(\mathcal{P})$ with Jarvis' March.

```
algorithm jarvis(S) is
    // S is the set of points
    // P will be the set of points which form the
convex hull. Final set size is i.
    pointOnHull = leftmost point in S // which is
guaranteed to be part of the CH(S)
    i := 0
    repeat
        P[i] := pointOnHull
        endpoint := S[0]      // initial endpoint
for a candidate edge on the hull
        for j from 0 to |S| do
            // endpoint == pointOnHull is a rare
case and can happen only when j == 1 and a better
endpoint has not yet been set for the loop
            if (endpoint == pointOnHull) or (S[j]
is on left of line from P[i] to endpoint) then
                endpoint := S[j]   // found
greater left turn, update endpoint
        i := i + 1
        pointOnHull = endpoint
    until endpoint = P[0]      // wrapped around
to first hull point
```

$\mathcal{O}(n)$

$\mathcal{O}(n)$

$\mathcal{O}(h)$



From Wikipedia, the free encyclopedia

By Maonus, CC BY-SA 4.0

Theorem 2.10

Jarvis' March computes the h vertices of $conv(\mathcal{P})$ in $\mathcal{O}(hn)$.

Algorithm 2.9: Compute $conv(\mathcal{P})$ with Jarvis' March.

```
algorithm jarvis(S) is
    // S is the set of points
    // P will be the set of points which form the
convex hull. Final set size is i.
    pointOnHull = leftmost point in S // which is
guaranteed to be part of the CH(S)
    i := 0
    repeat
        P[i] := pointOnHull
        endpoint := S[0]      // initial endpoint
for a candidate edge on the hull
        for j from 0 to |S| do
            // endpoint == pointOnHull is a rare
case and can happen only when j == 1 and a better
endpoint has not yet been set for the loop
            if (endpoint == pointOnHull) or (S[j]
is on left of line from P[i] to endpoint) then
                endpoint := S[j]   // found
greater left turn, update endpoint
        i := i + 1
        pointOnHull = endpoint
    until endpoint = P[0]      // wrapped around
to first hull point
```

$\mathcal{O}(n)$

$\mathcal{O}(n)$

$\mathcal{O}(h)$

output-sensitive

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Quickhull [Eddy, 1977]

A New Convex Hull Algorithm for Planar Sets

WILLIAM F. EDDY
Carnegie-Mellon University

A new algorithm, CONVEX, that determines which points of a planar set are vertices of the convex hull of the set is presented. It is shown that CONVEX operates in a fashion similar to the sorting algorithm QUICKERSORT. Evidence is given which indicates that in some situations CONVEX is preferable to earlier algorithms. A Fortran implementation, intended to minimize execution time, is presented and an alternative, which minimizes storage requirements, is discussed.

Key Words and Phrases: convex hull, QUICKERSORT, partitioning, sorting
CR Categories: 5.30, 5.31
The Algorithm: CONVEX, A New Convex Hull Algorithm for Planar Sets. *ACM Trans. Math. Software 3*, 4 (Dec. 1977), 411–412.

INTRODUCTION

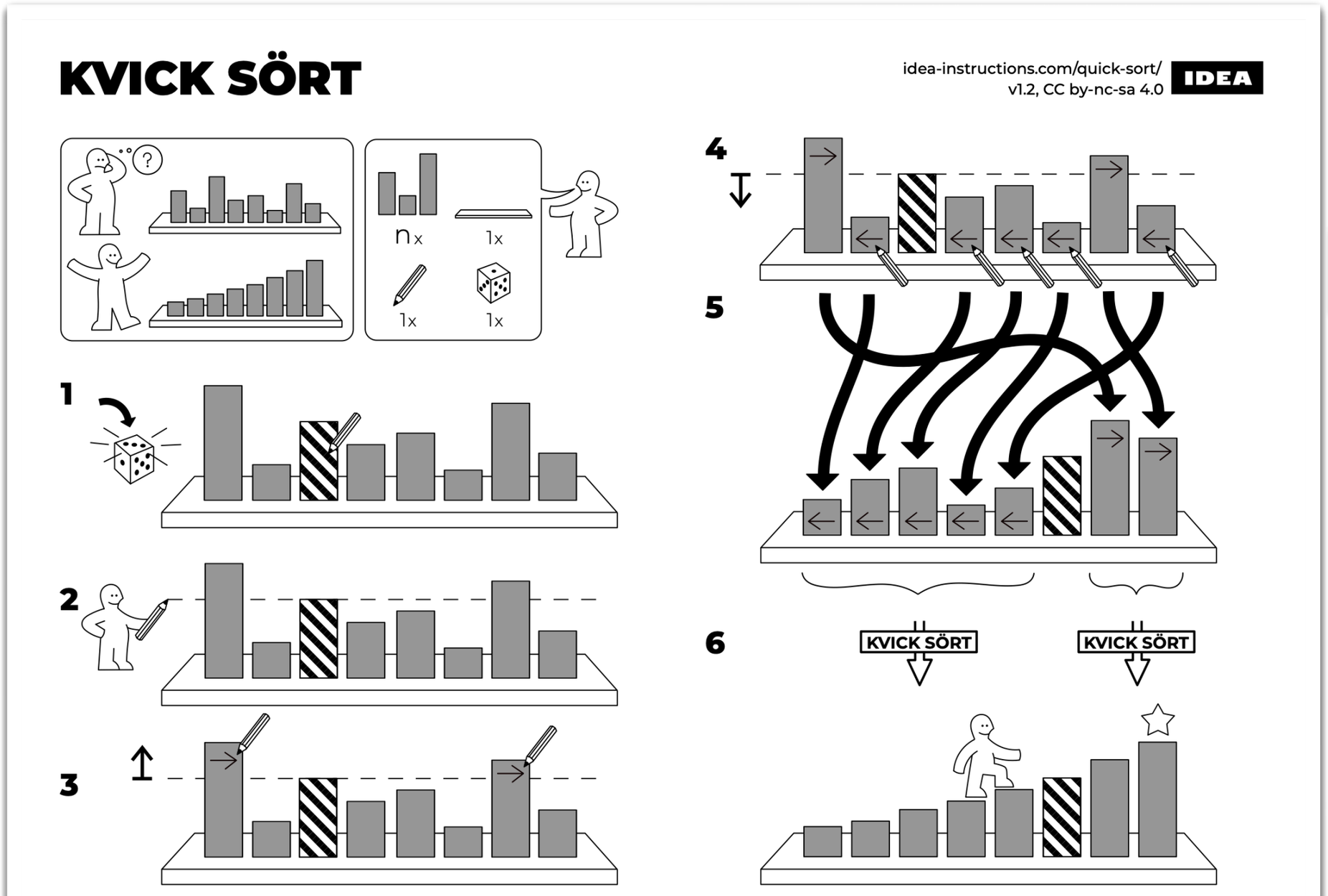
The convex hull of a planar set is the minimum-area convex polygon containing the planar set. A convex polygon is clearly determined by its vertices. Graham [1] suggests an algorithm for determining which points of a planar set are vertices of its convex hull. Because his algorithm requires sorting the points, if there are N points then at least $O(N \log N)$ operations are needed to determine the vertices. Recently, Preparata and Hong [3, 4] have shown that there exist sets of points for which every algorithm requires at least $O(N \log N)$ operations to determine the vertices of the convex hull. Jarvis [2] gives an algorithm which requires $O(N \cdot C)$ operations, where C is the number of vertices. For some configurations of the points in the plane (those with small values of C) the algorithm given by Jarvis will be faster than the algorithm of Graham; for other configurations it may be slower. An adaptive algorithm, CONVEX, is presented here which never requires more than $O(N \cdot C)$ operations to determine the vertices of the convex hull and may require substantially fewer. However, CONVEX may require more operations than Graham's algorithm for some configurations of points. Evidence is presented which suggests that in applications CONVEX is preferable to the "sorting" algorithms [1, 3, 4] and to Jarvis's algorithm [2].

METHOD

Operationally, this algorithm is analogous to the sorting algorithm QUICKERSORT [5]. At each step QUICKERSORT partitions the input array with respect to a

This research was supported in part by the National Science Foundation under Grant DCR75-08374.
Author's address: Department of Statistics, Carnegie-Mellon University, Schenley Park, Pittsburgh, PA 15213.

ACM Transactions on Mathematical Software, Vol. 3, No. 4, December 1977, Pages 398–403.


KVICK SÖRT
idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0
IDEA
n x
1x
1x
1x
1
2
3
4
5
6
KVICK SÖRT
KVICK SÖRT


2 1 3 4 7 5 6 8

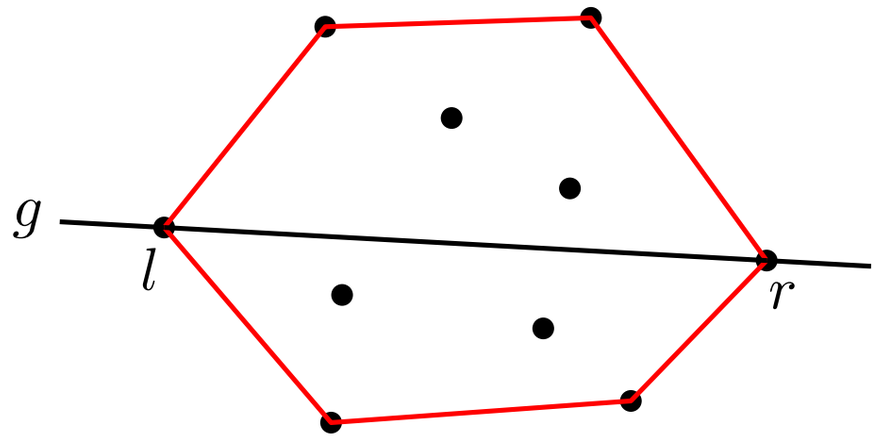
Basic idee:

- Use pivot element for subdivision into independent subsets
- Analogy: Quicksort
  - Pivot element $m \in A$: $A \to A_{<m} \circ A_{=m} \circ A_{>m}$
  - Concatenate subsequences

Transfer to $\mathbb{R}^2$:

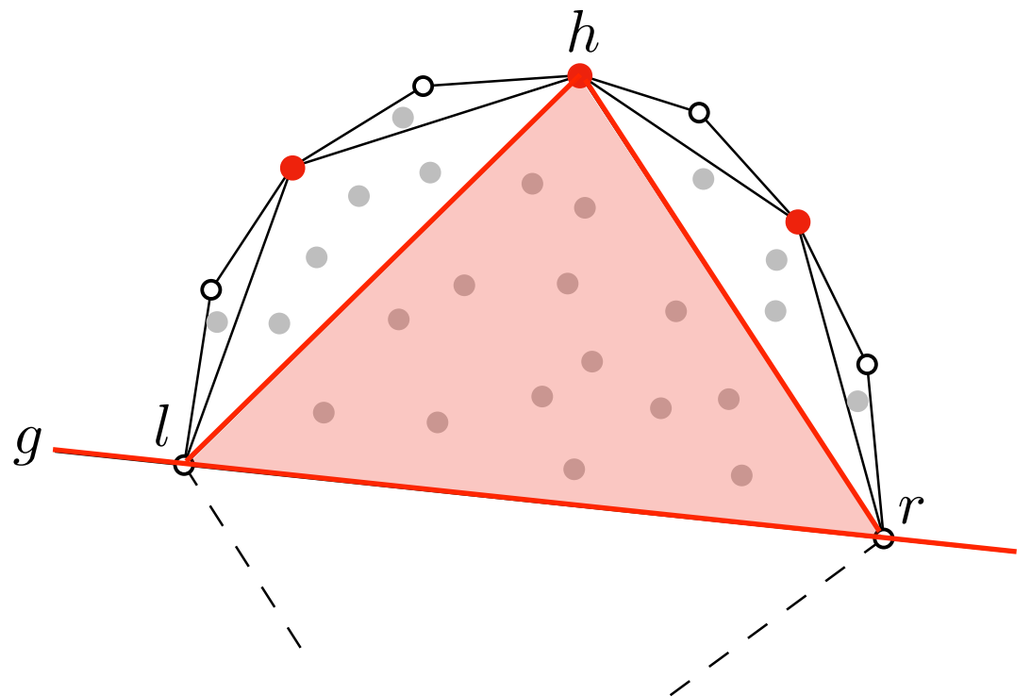
- Separation of $\mathcal{P}$ by line g
- $g :=$ line through extreme points l, r
- Concatenation of recursively computed hull

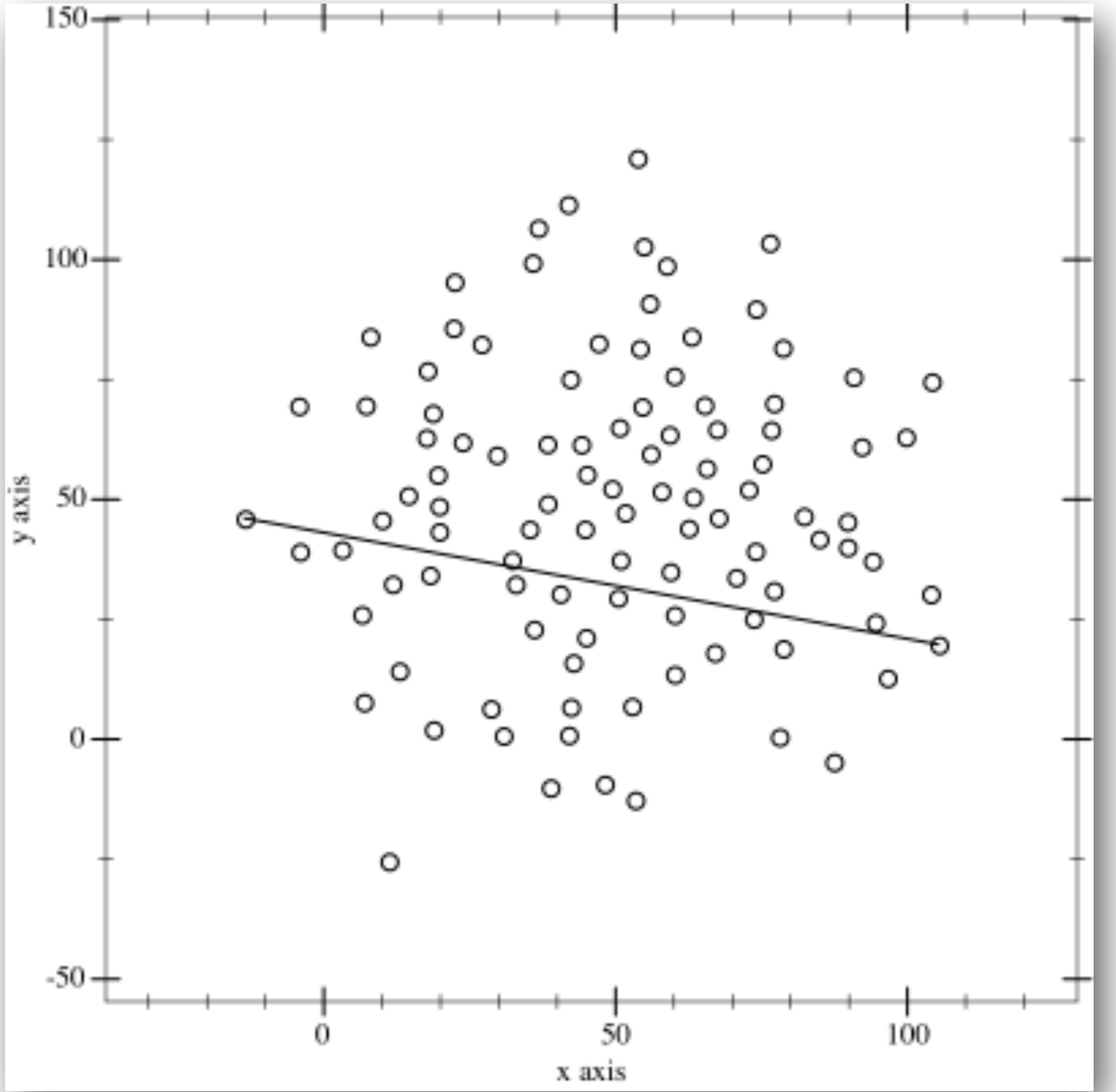
  $\to conv(\mathcal{P})$



Choosing the pivot element:

- Points above g
- Auxiliary point h: maximal distance to g.
- New pivot elements: $\overline{lh}, \overline{hr}$
- Delete $\mathcal{P} \cap \triangle(l, r, h)$.
- Two recursions:
  Above $\overline{lh}$ and below $\overline{rh}$
- „Exhaustion from inside“
- Analogously below g





By Maonus, CC BY-SA 4.0

Algorithm 2.11: Compute $conv(\mathcal{P})$ with Quickhull.

```
Input = a set S of n points
Assume that there are at least 2 points in the input set S of points

function QuickHull(S) is
    // Find convex hull from the set S of n points
    Convex Hull := {}
    Find left and right most points, say A & B, and add A & B to convex hull
    Segment AB divides the remaining (n - 2) points into 2 groups S1 and S2
        where S1 are points in S that are on the right side of the oriented line from A to B,
        and S2 are points in S that are on the right side of the oriented line from B to A
    FindHull(S1, A, B)
    FindHull(S2, B, A)
    Output := Convex Hull
end function

function FindHull(Sk, P, Q) is
    // Find points on convex hull from the set Sk of points
    // that are on the right side of the oriented line from P to Q
    if Sk has no point then
        return
    From the given set of points in Sk, find farthest point, say C, from segment PQ
    Add point C to convex hull at the location between P and Q
    Three points P, Q, and C partition the remaining points of Sk into 3 subsets: S0, S1, and
S2
        where S0 are points inside triangle PCQ, S1 are points on the right side of the
oriented
        line from P to C, and S2 are points on the right side of the oriented line from C to
Q.
    FindHull(S1, P, C)
    FindHull(S2, C, Q)
end function
```

From Wikipedia, the free encyclopedia

Theorem 2.12
Quickhull computes $conv(\mathcal{P})$ in $\mathcal{O}(n^2)$ worst-case and in $\mathcal{O}(n \log n)$ best-case runtime.

Excercises:

- Details of implementation
- Termination
- Runtime

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Divide-and-Conquer [Preparata and Hong, 1977]

4. Bobrow, D.G., Burchfiel, J.D., Murphy, D.L., and Tomlinson, R.S. TENEX, a paged time sharing system for the PDP-10. *Comm. ACM 15*, 3 (March 1972), 135–143.
5. Bobrow, D.G., and Murphy, D.L. The structure of a LISP system using two level storage. *Comm. ACM 10*, 3 (March 1967). 155–159.
6. Cheney, C.J. A nonrecursive list compacting algorithm. *Comm. ACM 13*, 11 (Nov. 1970), 677–678.
7. Deutsch, L.P. An interactive program verifier. Ph.D. Th., Comptr. Sci. Dep., U. of California, Berkeley, Calif., May 1973.
8. Deutsch, L.P. A LISP machine with very compact programs. Third Int. Joint Conf. on Artificial Intelligence, Stanford, Calif., 1973, pp. 697–703.
9. Fenichel, R.R., and Yochelson, J.C. A LISP garbage-collector for virtual-memory computer systems. *Comm. ACM 12*, 11 (Nov. 1969), 611–612.
10. Hansen, W.J. Compact list representation: Definition, garbage collection, and system implementation. *Comm. ACM 12*, 9 (Sept. 1969), 499–507.
11. Hehner, E.C.R. Matching program and data representations to a computing environment. Ph.D. Th., Comptr. Systems Res. Group, U. of Toronto, Toronto, Canada, Nov. 1974.
12. McCarthy, J., et al. *LISP 1.5 Programmer's Manual*. M.I.T. Press, Cambridge, Mass., 1962.
13. Minsky, M.L. A LISP garbage collector algorithm using serial secondary storage. Artificial Intelligence Proj. Memo 58 (rev.), Project MAC, M.I.T., Cambridge, Mass., Dec. 1963.
14. Quam, L.H. Stanford LISP 1.6 manual. Artificial Intelligence Proj., Stanford University, Stanford, Calif., Sept. 1969.
15. Reboh, R., and Sacerdoti, E. A preliminary QLISP manual. Tech. Note 81, Stanford Res. Inst. AI Center, Menlo Park, Calif., Aug. 1973.
16. Sacerdoti, E. The nonlinear nature of plans. Fourth Int. Joint Conf. on Artificial Intelligence, Tbilisi, Georgia, U.S.S.R., 1975, pp. 206–214.
17. Shannon, C.E. A mathematic theory of communication. *Bell System Tech. J. 27* (July 1948), 379–423.
18. Smith, D.H., Masinter, L.M., and Sridharan, N.S. Heuristic DENDRAL: Analysis of molecular structure. In *Computer Representation and Manipulation of Chemical Information*, W.T. Wipke, S. Heller, R. Feldman, and E. Hyde, Eds., Wiley, New York, 1974.
19. Teitelman, W. INTERLISP Reference manual. Xerox Palo Alto Res. Center, Palo Alto, Calif., 1974.
20. Van der Poel, W.L. A Manual of HISP for the PDP-9. Technical U., Delft, Netherlands.
21. Weissman, C. *LISP 1.5 Primer*. Dickenson Pub. Co., Belmont, Calif., 1967.
22. Wilner, W.T. Design of the B1700. Proc. AFIPS 1972 FJCC, Vol. 41, AFIPS Press, Montvale, N.J., pp. 579–586.
23. Zipf, G.K. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, Mass., 1949.

Convex Hulls of Finite Sets of Points in Two and Three Dimensions

F. P. Preparata and S. J. Hong
University of Illinois at Urbana-Champaign

The convex hulls of sets of n points in two and three *dimensions can be determined with O(n log n)* operations. The presented algorithms use the "divide and conquer" technique and recursively apply a merge procedure for two nonintersecting convex hulls. Since any convex hull algorithm requires at least O(n log n) operations, the time complexity of the proposed algorithms is optimal within a multiplicative constant.

Key Words and Phrases: computational complexity, convex hull, optimal algorithms, planar set of points, spatial set of points
CR Categories: 4.49, 5.25, 5.32

1. Introduction

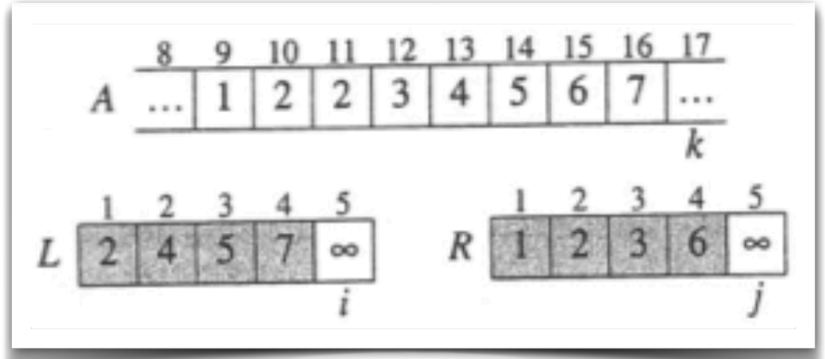
The determination of the convex hull of a finite set of points is relevant to several problems in computer graphics, design automation, pattern recognition and operations research: references [3, 4, 10]—just to cite a few—discuss some interesting applications in these areas, which require convex hull computation.

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB-07-72-C-0259.

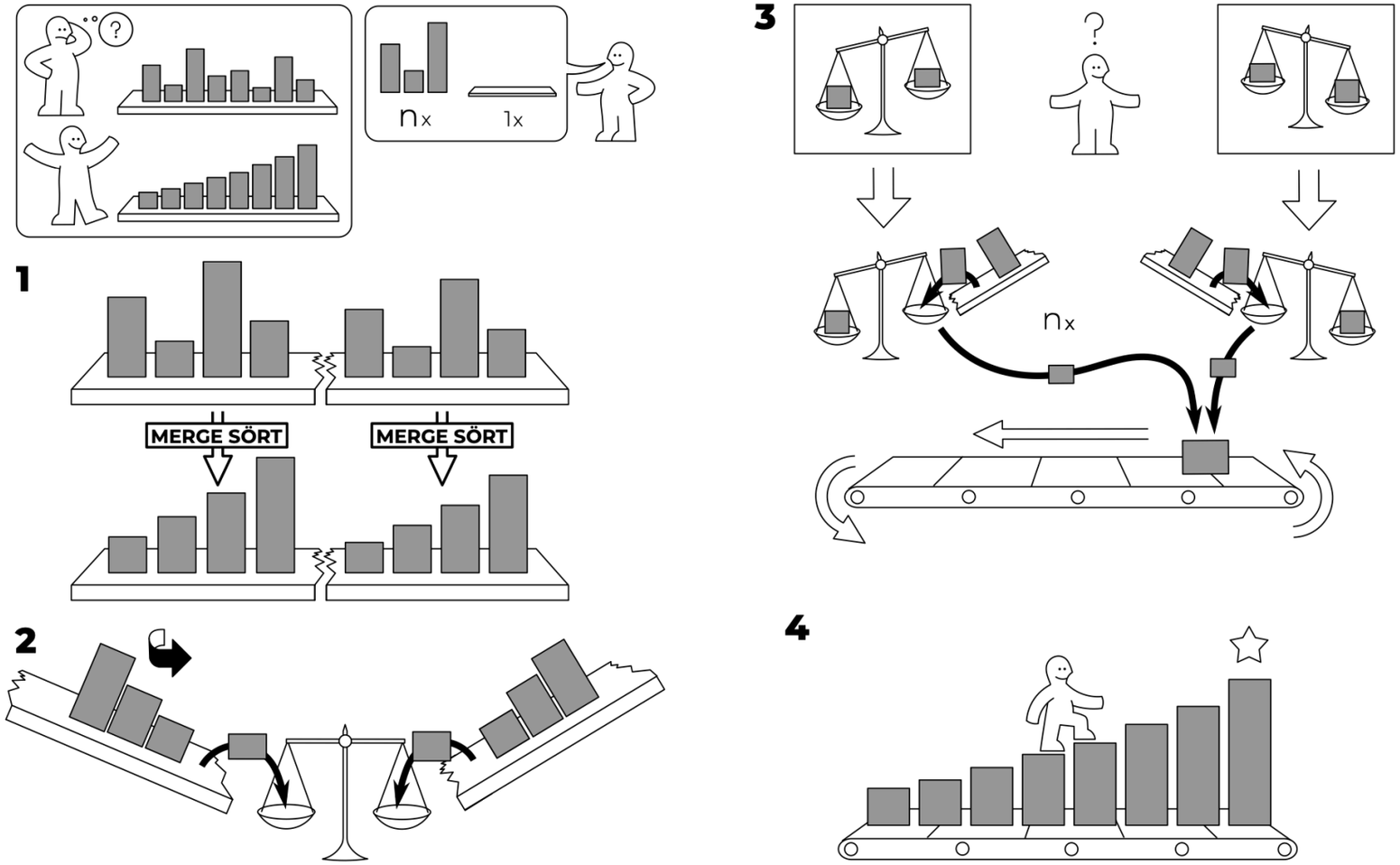
Authors' addresses: F.P. Preparata, Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801; S.J. Hong, IBM Systems Product Division, Poughkeepsie, NY 12602. This work was completed while the second author was on leave at the University of Illinois.

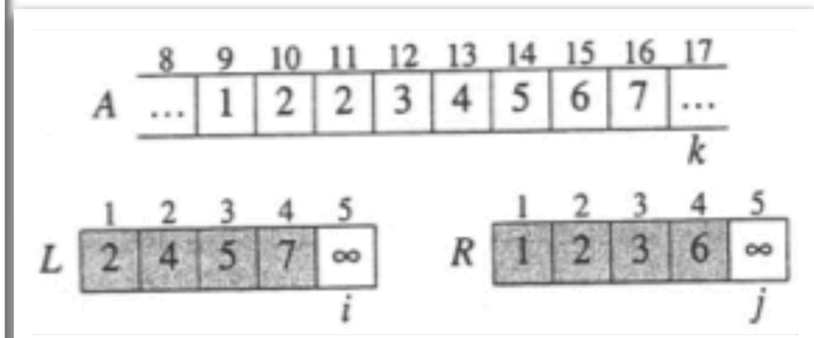
Basic idea:

- Balanced subdivision, solve recursively
- Analogy: Mergesort
  - Split array in a balanced fashion into two arrays
  - Sort recursively
  - Combine



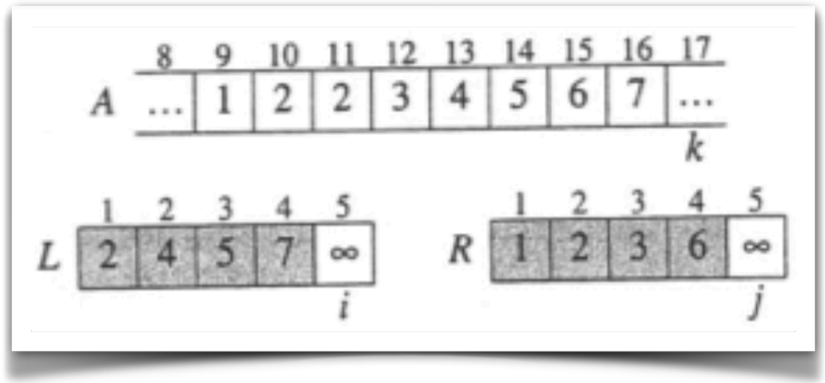
Divide-and-Conquer [Preparata and Hong, 1977]





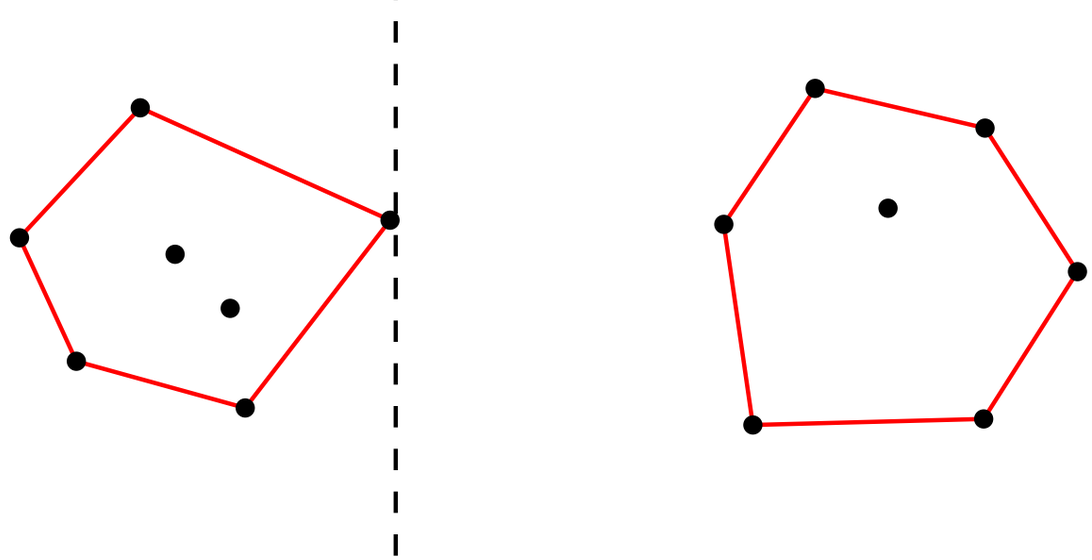
Basic idea:

- Balanced subdivision, solve recursively
- Analogy: Mergesort
  - Split array in a balanced fashion into two arrays
  - Sort recursively
  - Combine



Transfer to $\mathbb{R}^2$:

- Separate by x-median
- Recursively: right and left hull
- Combine left and right hull



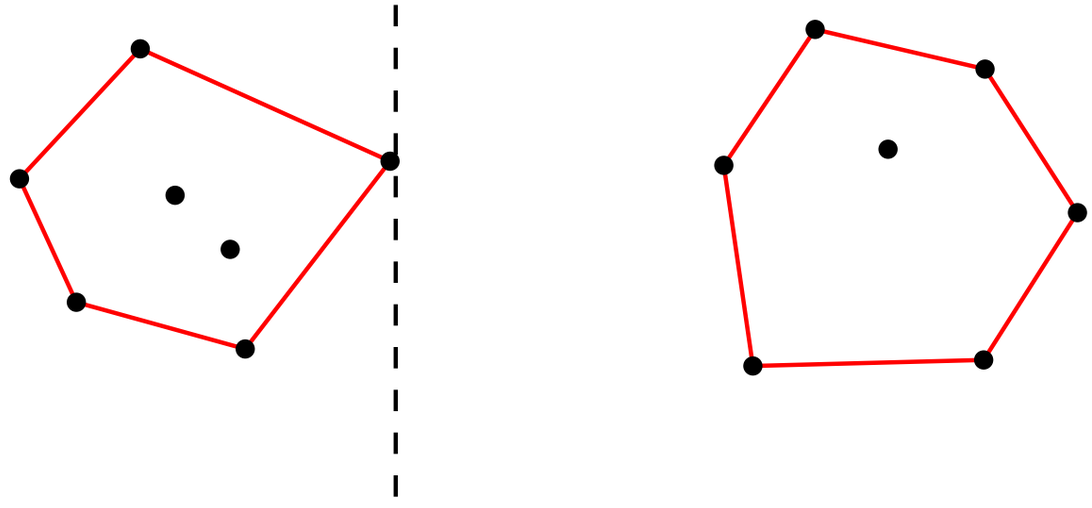
Assumption: General position (1. distinct x-coordinate and
2. no three points collinear)

- Split wrt. x-median $\Rightarrow$ Left and right hull disjoint
- Runtime:

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n) + f_{merge}(n) \text{ with } f(n) \in \Theta(n)$$

$f_{merge}(n)$:= time for merging right and left hull

- Goal: $f_{merge}(n) \in \Theta(n)$
- Then $T(n) \in \mathcal{O}(n \log n)$



Algorithmic approach:

- Find extreme points wrt. $\leq_x$ for both hulls

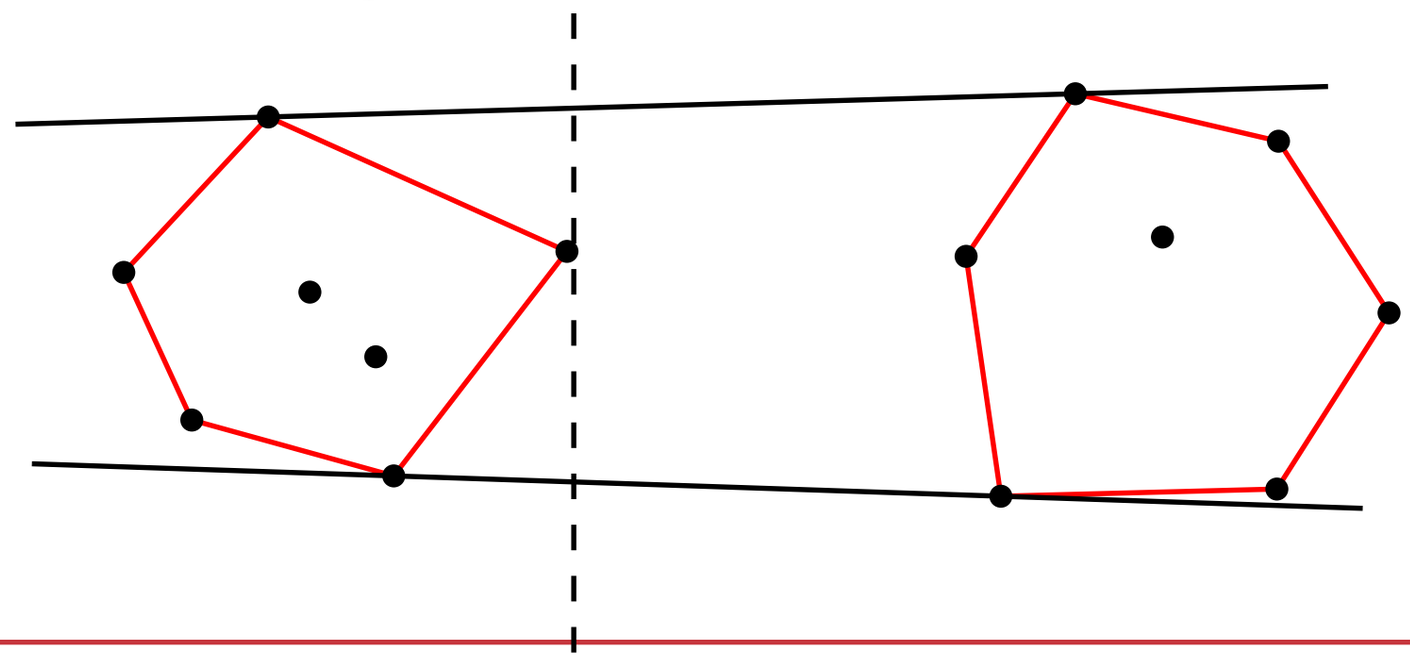
  $v_0, v_k :=$ min./max. x-coordinate in left hull

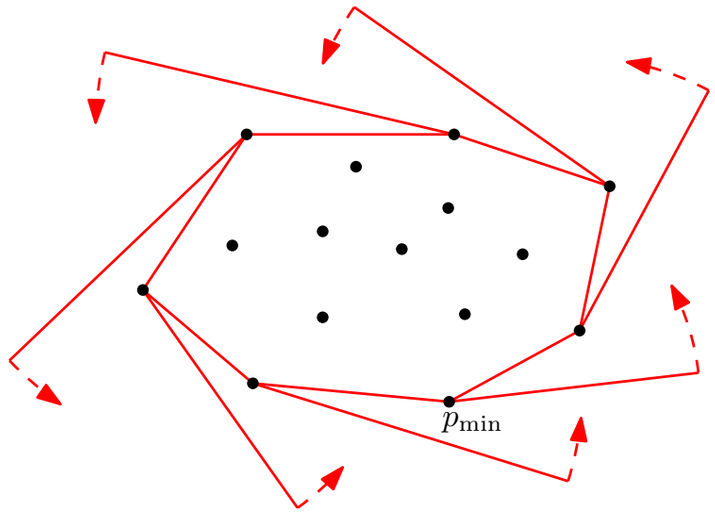
  $w_0, w_l :=$ min./max. x-coordinate in right hull

- Find tangent for lower hulls $(v_0, \ldots, v_k)$ und $(w_0, \ldots, w_l)$.

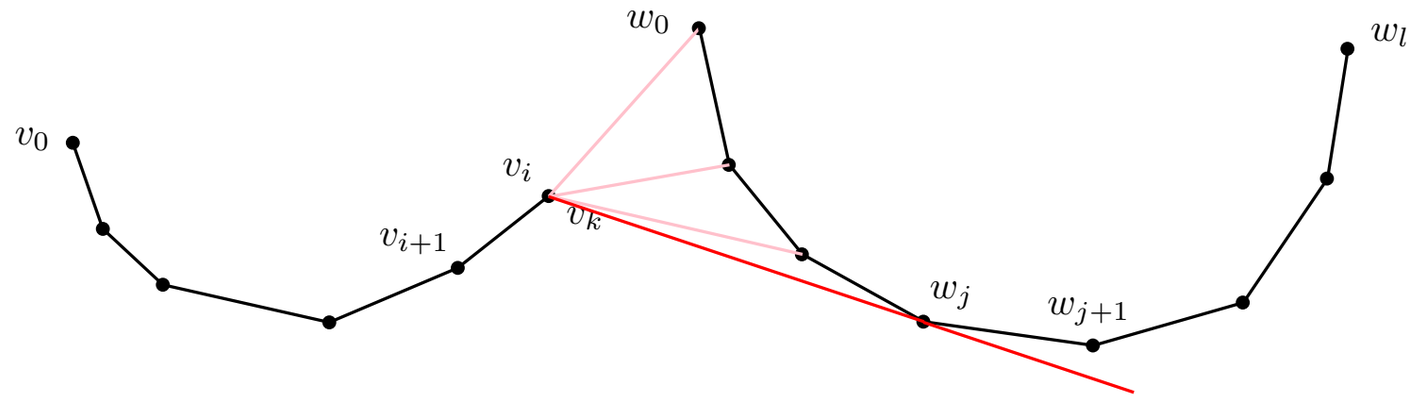
- Find tangent for upper hulls $(v_k, \ldots, v_0)$ und $(w_l, \ldots, w_0)$

- Delete points between tangent points





- Some ideas similar to Gift Wrapping
- Various technical details
- Omitted here
- Lower bound: $\Omega(n \log n)$



Theorem 2.18

The algorithm of Preparata and Hong computes $conv(\mathcal{P})$ in optimal time $\Theta(n \log n)$.

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Graham's Scan [Graham, 1972]

INFORMATION PROCESSING LETTERS 1 (1972) 132–133. NORTH-HOLLAND PUBLISHING COMPANY

AN EFFICIENT ALGORITH FOR DETERMINING THE CONVEX HULL OF A FINITE PLANAR SET

R.L. GRAHAM
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey, USA

Received 28 January 1972

convex hull algorithm

Given a finite set $S = \{s_1, \ldots s_n\}$ in the plane, it is frequently of interest to find the convex hull CH(S) of S. In this note we describe an algorithm which determines CH(S) in no more than $(n \log n)/(\log 2) + cn$ "operations" where c is a small positive constant which depends upon what is meant by an "operation".

The algorithm we give determines which points of S are the extreme points of CH(S). These, of course, define CH(S). The algorithm proceeds in five steps.

Step 1: Find a point P in the plane which is in the interior of CH(S). At worst, this can be done in $c_1 n$ steps by testing 3 element subsets of S for collinearity, discarding middle points of collinear sets and stopping when the first noncollinear set (if there is one), say x, y and z, is found. P can be chosen to be the centroid of the triangle formed by x, y and z.

Step 2: Express each $s_i \in S$ in polar coordinates with origin P and $\theta = 0$ in the direction of an arbitrary fixed half-line L from P. This conversion can be done in $c_2 n$ operations for some fixed constant c_2.

Step 3: Order the elements $\rho_k \exp(i\theta_k)$ of S in terms of increasing θ_k. This is well known to be possible in essentially $(n \log n)/\log 2$ comparisons (cf. [1]). We now have S in the form $S = \{r_1 \exp(i\varphi_1), \ldots, r_n \exp(i\varphi_n)\}$ with $0 \leqslant \varphi_1 \leqslant \ldots \leqslant \varphi_n < 2\pi$ and $r_i \geqslant 0$ (cf. fig. 1). Note that by the choice of P, $\varphi_{k-1} - \varphi_k < \pi$ where the index addition is modulo n.

Step 4: If $\varphi_i = \varphi_{i+1}$ then we may delete the point with the smaller amplitude since it clearly cannot be an extreme point of CH(S). Also any point with $r_i = 0$ can be deleted. We can eliminate all these points in less than n comparisons, and by relabelling the remaining points, we can set $S' = \{r_1 \exp(i\varphi_1), \ldots, r_{n'} \exp(i\varphi_{n'})\}$ where $n' \leqslant n$.

Step 5: Start with three consecutive points in S', say, $r_k \exp(i\varphi_k)$, $r_{k+1} \exp(i\varphi_{k+1})$, $r_{k+2} \exp(i\varphi_{k+2})$ with $\varphi_k < \varphi_{k+1} < \varphi_{k+2}$ (cf. fig. 2). There are two possibilities:

(i) $\alpha + \beta \geqslant \pi$. Then we delete the point $r_{k+1} \exp(i\varphi_{k+1})$ from S' since it cannot be an extreme point of CH(S), and return to the beginning of step 5 with the points $r_k \exp(i\varphi_k)$, $r_{k+1} \exp(i\varphi_{k+1})$, $r_{k+2} \exp(i\varphi_{k+2})$ replaced by $r_{k-1} \exp(i\varphi_{k-1})$, $r_k \exp(i\varphi_k)$, $r_{k+2} \exp(i\varphi_{k+2})$ (where indices are reduced modulo n').

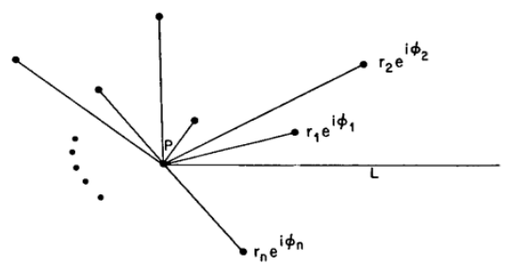


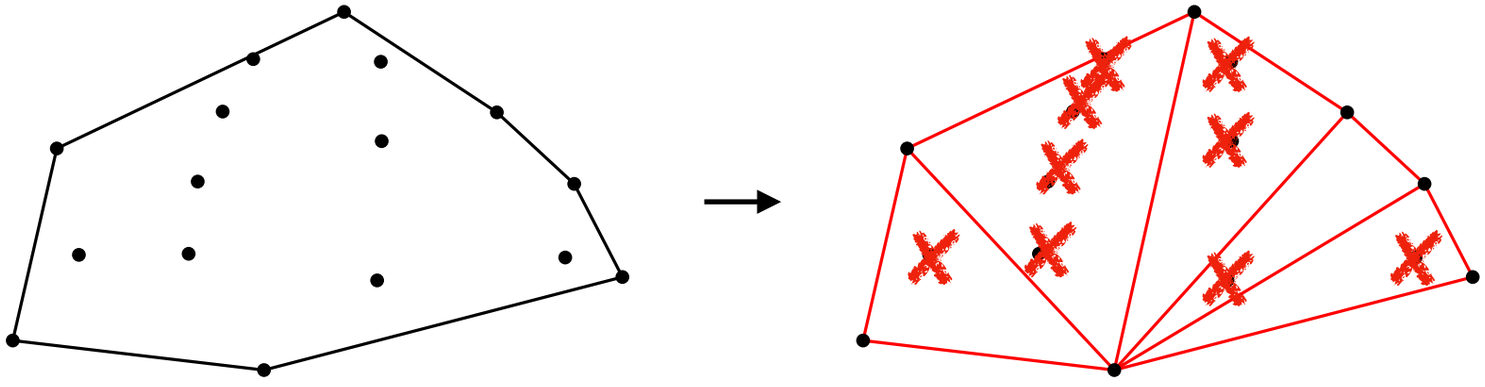
Fig. 1.

Overview:

- First optimal algorithm
- [Graham, 1972] is considered „the first publication in Computational Geometry".

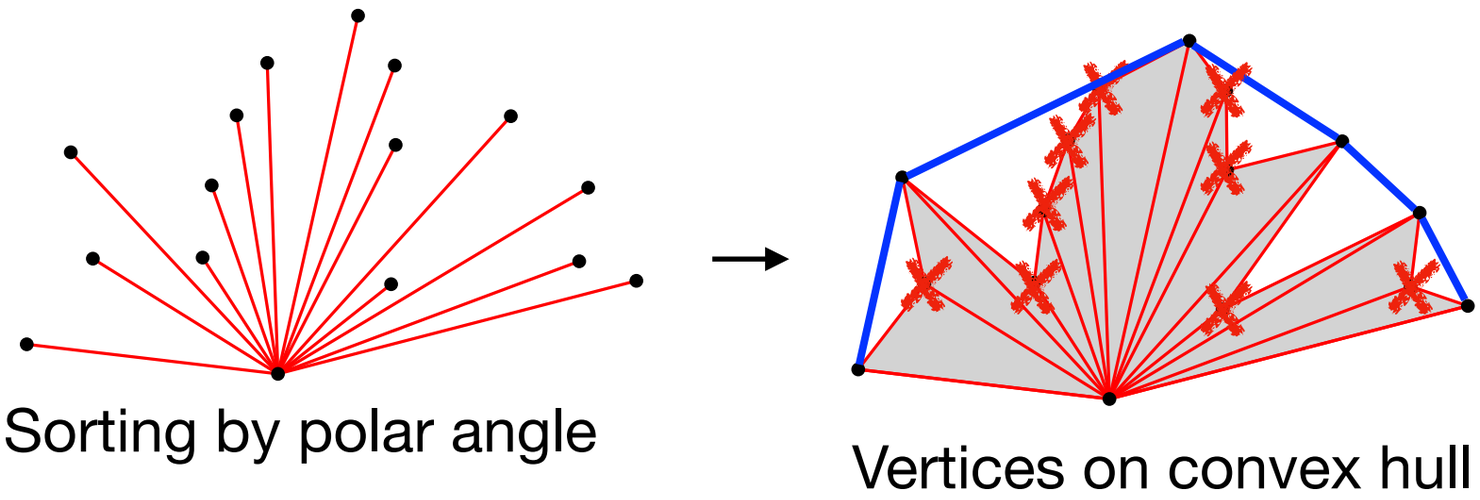
Basic idea:

- Delete points in triangles. [Known]
- Only consider $\mathcal{O}(n)$ triangles. [new]
- Ensure: #"point-in-triangle-tests $\in \mathcal{O}(n)$ [new]



Convex hull

Subdivision into triangles



Sorting by polar angle

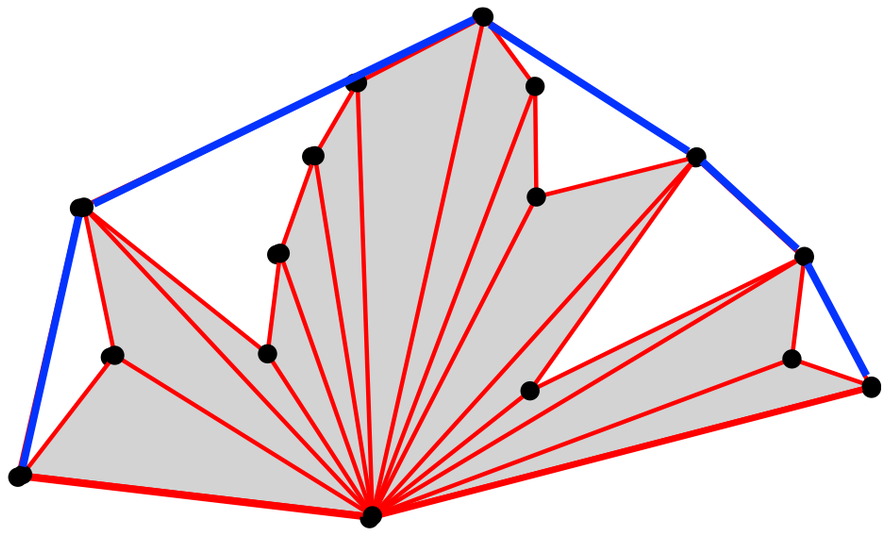
Vertices on convex hull

Goal:

- Finding a sequence of „left“ turns

  ⇳

  Knowing $conv(\mathcal{P})$

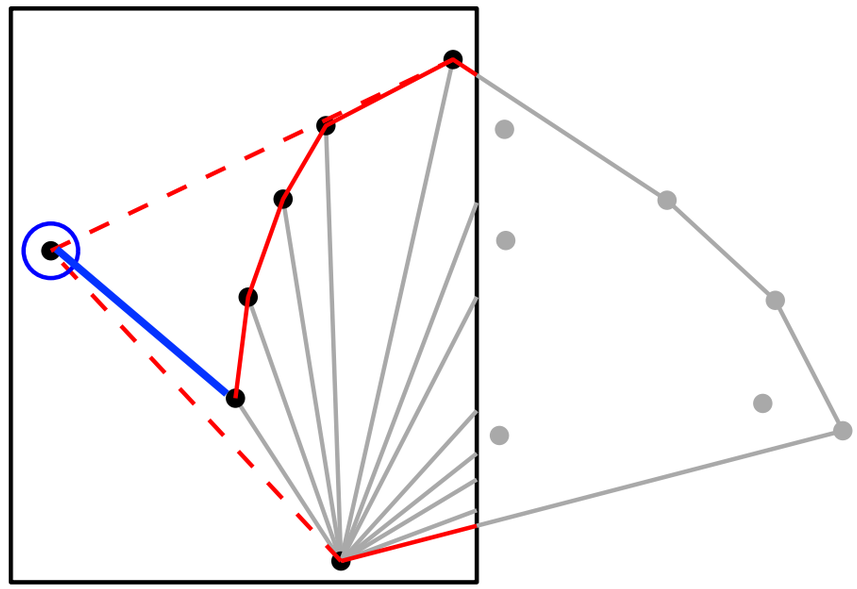


Approach:

- Maintain stack of vertices

- In case of a „right“ turn:

  Pop vertex off stack

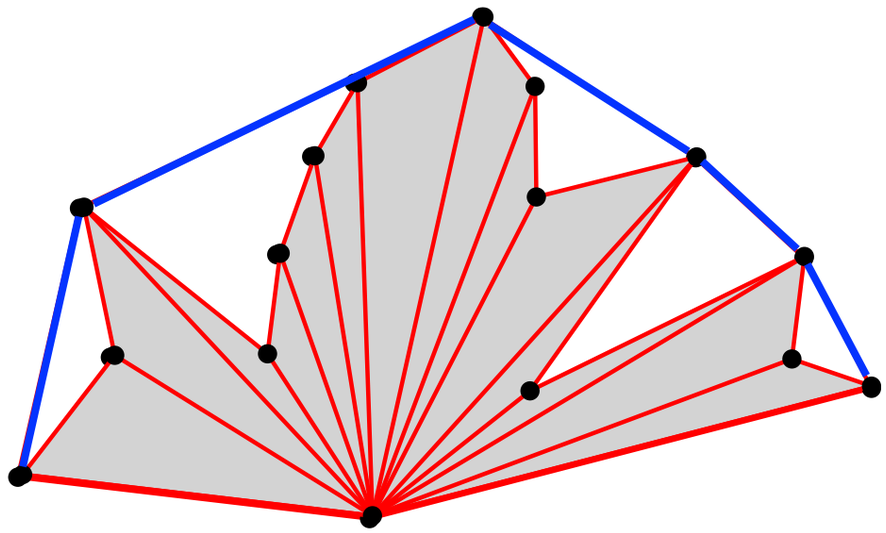


Goal:

- Finding a sequence of „left“ turns

  ⇳

  Knowing $conv(\mathcal{P})$

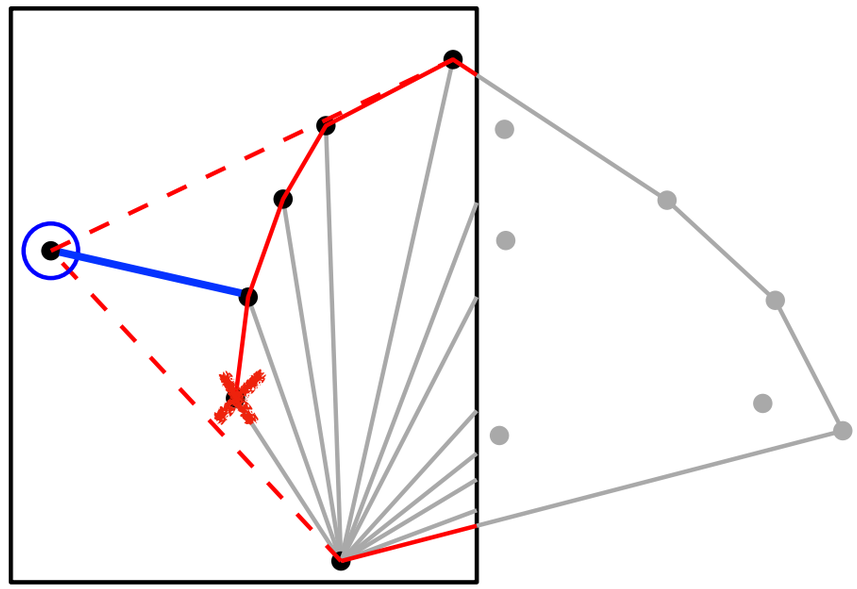


Approach:

- Maintain stack of vertices

- In case of a „right“ turn:

  Pop vertex off stack

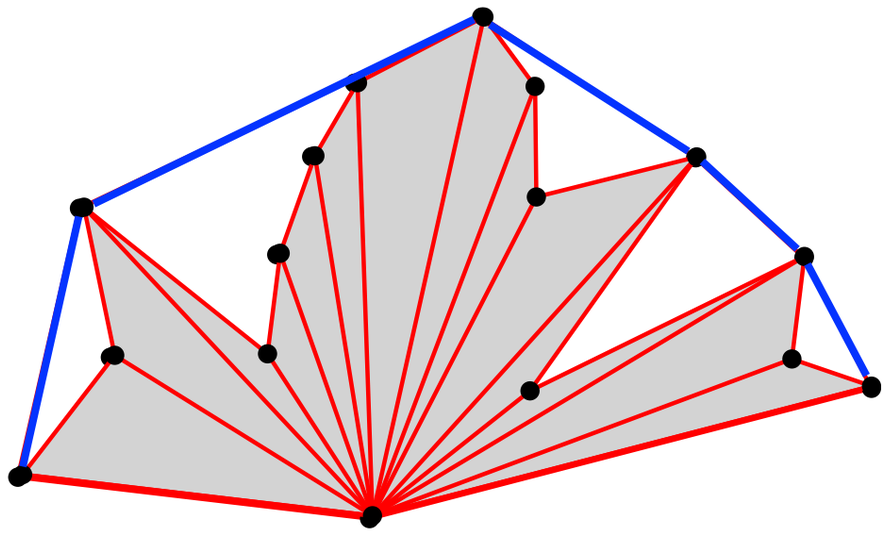


Goal:

- Finding a sequence of „left“ turns

⇳

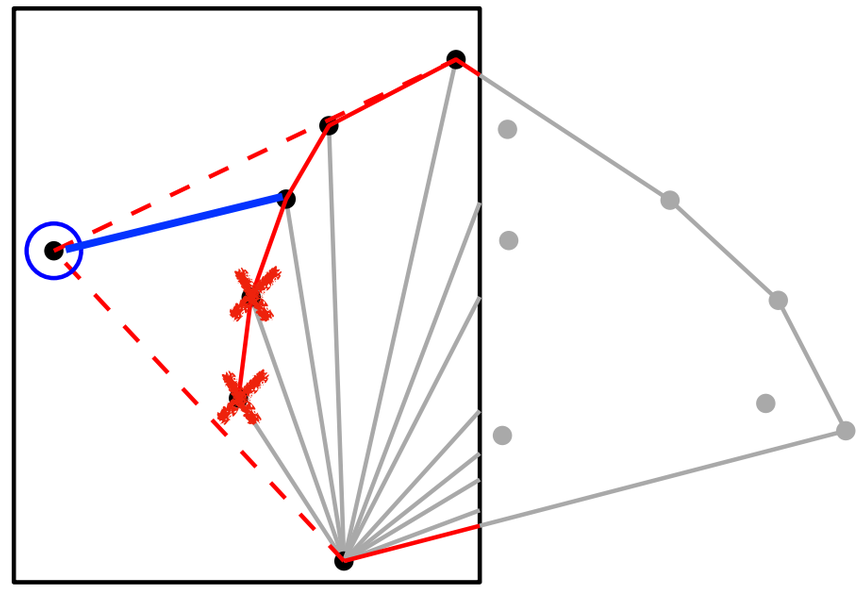
Knowing $conv(\mathcal{P})$



Approach:

- Maintain stack of vertices
- In case of a „right“ turn:

  Pop vertex off stack

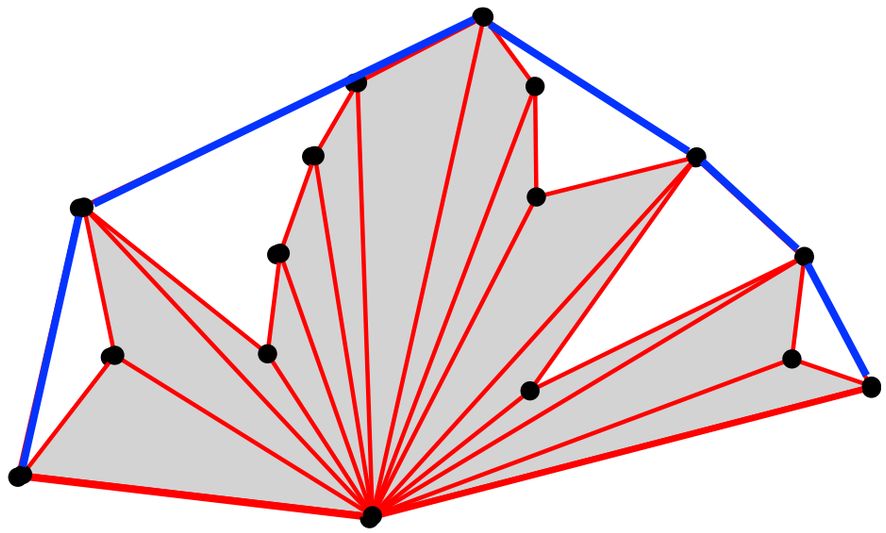


Goal:

- Finding a sequence of „left“ turns

⇳

Knowing $conv(\mathcal{P})$

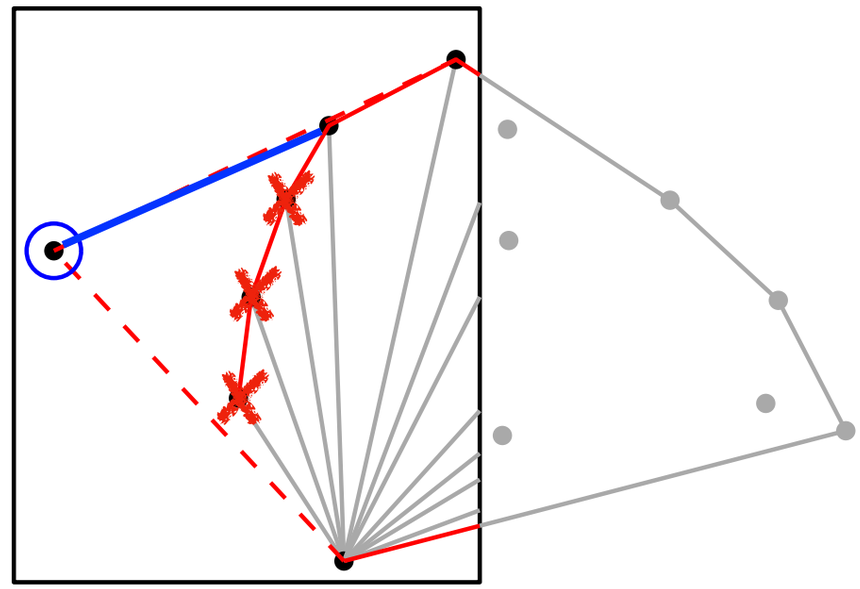


Approach:

- Maintain stack of vertices

- In case of a „right“ turn:

  Pop vertex off stack

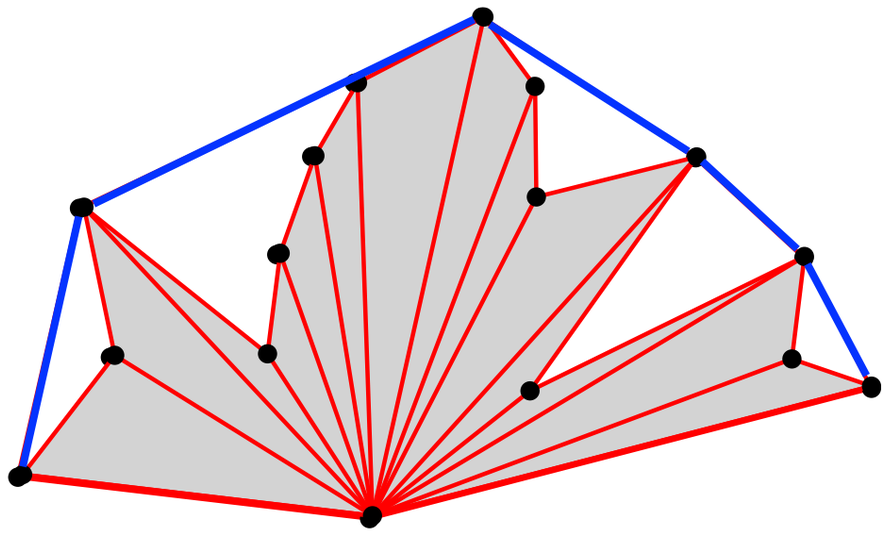


Goal:

- Finding a sequence of „left“ turns

⇳

Knowing $conv(\mathcal{P})$

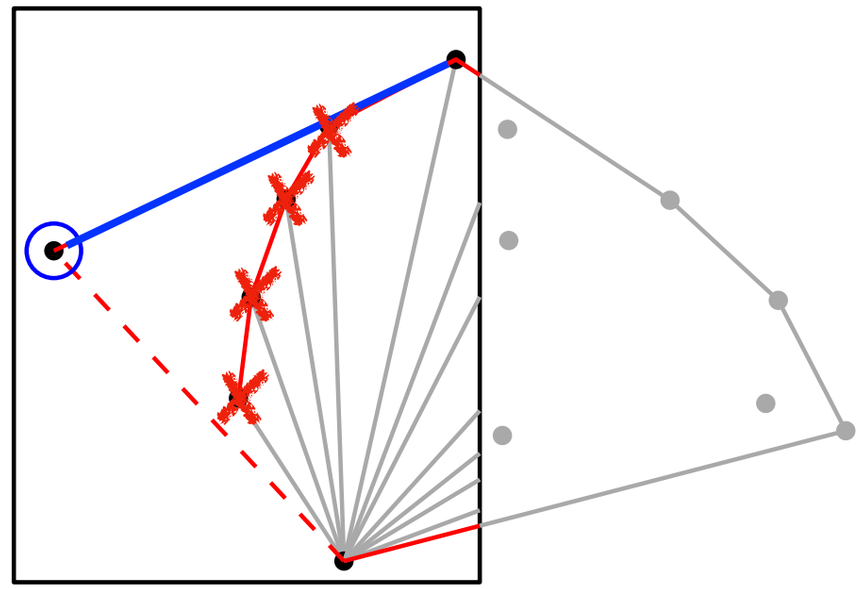


Approach:

- Maintain stack of vertices

- In case of a „right“ turn:

  Pop vertex off stack



Algorithm 2.23: Compute $conv(\mathcal{P})$ with Graham's Scan.

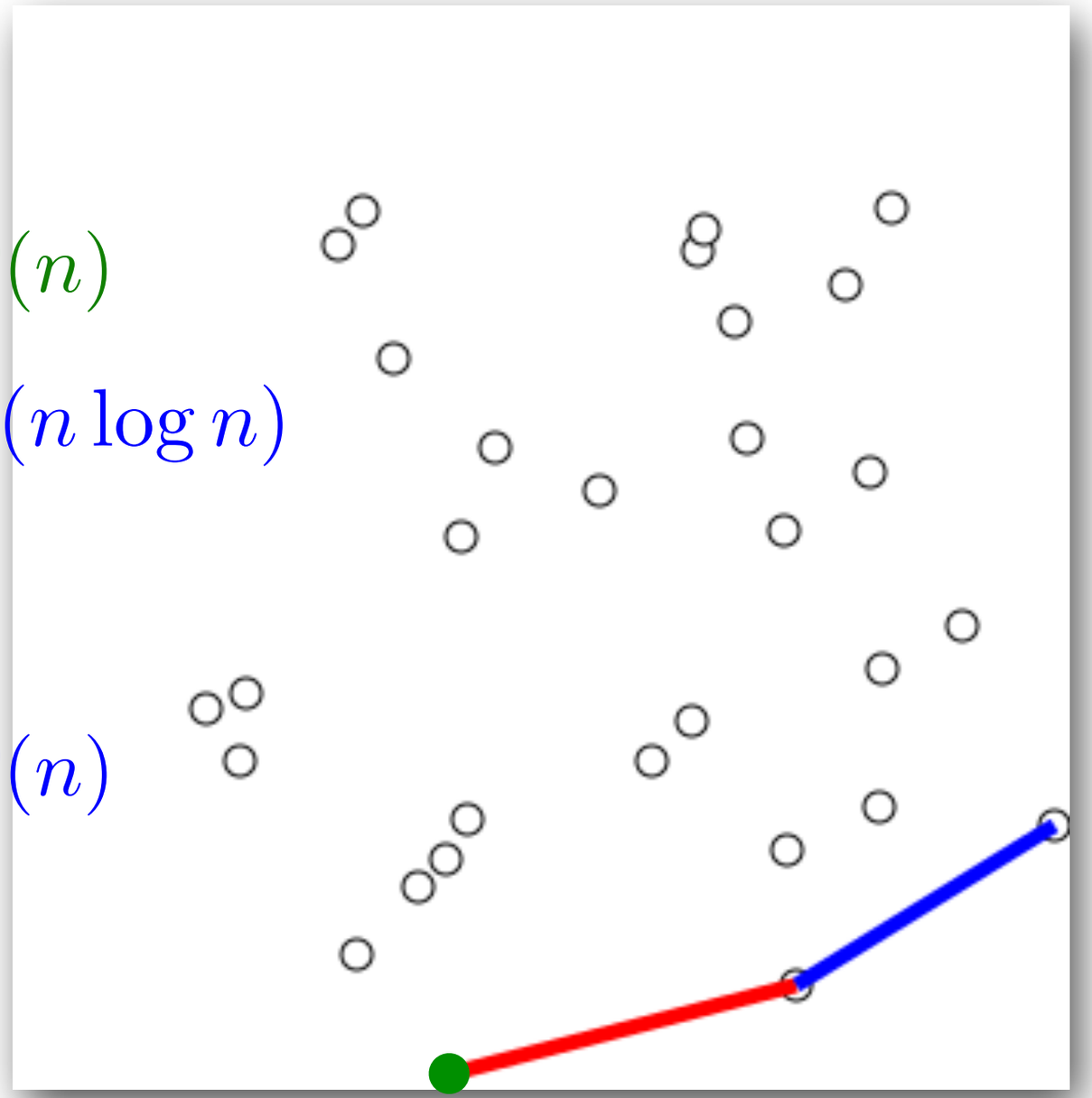
```
let points be the list of points
let stack = empty_stack()

find the lowest y-coordinate and leftmost
point, called P0
sort points by polar angle with P0, if
several points have the same polar angle
then only keep the farthest

for point in points:
    # pop the last point from the stack if
we turn clockwise to reach this point
    while count stack > 1 and
ccw(next_to_top(stack), top(stack), point)
<= 0:
        pop stack
    push point to stack
end
```

$\mathcal{O}(n)$

$\mathcal{O}(n \log n)$

$\mathcal{O}(n)$

From Wikipedia, the free encyclopedia



By Shiyu Ji, CC BY-SA 4.0

Theorem 2.24
Graham's Scan computes $conv(\mathcal{P})$ in $\Theta(n \log n)$.

Remarks:

- Implementation details: [O'Rourke, 1994, Chap. 3.5].
- Graham's Scan is also optimal in the following models:
  - Single-disk I/O-model [Aggarwal und Vitter, 1988].
  - Cache-oblivous model [Frigo et al., 1988].
  - In-place model [Brönnimann et al., 2004].

1. Introduction and Definitions
2. Interlude: Algorithmic Paradigms
3. Jarvis' March
4. Quickhull
5. Divide-and-conquer and incremental construction
6. Graham's Scan
7. Optimal output-sensitive construction

Recall:

- Jarvis' March: output-sensitive: $\mathcal{O}(hn)$.
  - h : Output
  - n : Input
- No contradiction to $\Omega(n \log n)$, as $h \in \Theta(n)$ is po

Theorem 2.35 ([Kirkpatrick and Seidel, 1986])
Computing $conv(\mathcal{P})$ has a lower bound of $\Omega(n \log h)$.

Theorem 2.36 ([Kirkpatrick and Seidel, 1986])
Computing $conv(\mathcal{P})$ can be achieved in $\mathcal{O}(n \log h)$.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

021

THE ULTIMATE PLANAR CONVEX HULL ALGORITHM?*

DAVID G. KIRKPATRICK† AND RAIMUND SEIDEL‡

Abstract. We present a new planar convex hull algorithm with worst case time complexity $O(n \log H)$ where n is the size of the input set and H is the size of the output set, i.e. the number of vertices found to be on the hull. We also show that this algorithm is asymptotically worst case optimal on a rather realistic model of computation even if the complexity of the problem is measured in terms of input as well as output size. The algorithm relies on a variation of the divide-and-conquer paradigm which we call the "marriage-before-conquest" principle and which appears to be interesting in its own right.

Key words. computational geometry, convex hull, divide-and-conquer, lower bounds

AMS(MOS) subject classifications. 68P10, 52-04, 52A10

1. Introduction. The *convex hull* of a finite point set S in the plane is the smallest convex polygon containing the set. The vertices (corners) of this polygon must be points of S. Thus in order to compute the convex hull of a set S it is necessary to find those points of S which are vertices of the hull. For the purposes of constructing upper bounds we define the *convex hull problem*, as the problem of constructing the ordered sequence of points of S which constitute the sequences of vertices around the hull.

The convex hull problem was one of the first problems in the field of computational geometry to have been studied from the point of view of computational complexity. In fact, efficient algorithmic solutions were proposed even before the term "computational geometry" was coined. This, along with its very extensive analysis in recent years, reflects both the theoretical and practical importance of the problem.

Of the convex hull algorithms proposed so far several have $O(n \log n)$ worst case time bounds [4], [8], [14], [15], [17], where n is the size of the input point set. Shamos [17] even argued that the $O(n \log n)$ time bound is worst case optimal. He observed that a set S of n real numbers could be sorted by finding the convex hull of the planar set $S' = \{(x, x^2) | x \in S\}$. But sorting, of course, has an $\Omega(n \log n)$ lower bound on a wide range of computational models. Yao [19] and on weaker computational models Avis [2], van Emde Boas [7], and Preparata and Hong [15] proved the $\Omega(n \log n)$ bound for a less demanding version of the convex hull problem: just the vertices of the convex hull are to be identified, irrespective of their sequence.

In contrast to the results above, it is interesting to observe that algorithms exist which solve the planar convex hull problem in $O(nH)$ time, where H is the number of vertices found to be on the hull [6], [9]. For small H, these algorithms seem to be superior to the $O(n \log n)$ methods. (This, of course, does not contradict the previously cited lower bound results, as H could be as large as n). It is notable, however, that all of the lower bound arguments mentioned above are insensitive to H in that they assume that some fixed fraction of the data points are vertices of the convex hull.

In this paper we present a convex hull algorithm with worst case time complexity $O(n \log H)$. Thus its running time is not only sensitive to both n and H, but it is also worst case optimal in the traditional sense when the running time is measured as a function of n only. However, we also show that our algorithm is asymptotically worst case optimal even if the complexity of the problem is measured as a function of both n and H.

* Received by the editors November 15, 1983, and in revised form August 15, 1984. This research was supported by the Natural Sciences and Engineering Research Council of Canada, grant A3583.
† Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1W5 Canada.
‡ Department of Computer Science, Cornell University, Ithaca, New York 14853.

Chan's Algorithm [Chan, 1996]

Discrete Comput Geom 16:361–368 (1996)

Discrete & Computational Geometry

Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions*

T. M. Chan

Department of Computer Science, University of British Columbia,
Vancouver, British Columbia, Canada V6T 1Z4

Abstract. We present simple output-sensitive algorithms that construct the convex hull of a set of n points in two or three dimensions in worst-case optimal $O(n \log h)$ time and $O(n)$ space, where h denotes the number of vertices of the convex hull.

1. Introduction

Given a set P of n points in the Euclidean plane E^2 or Euclidean space E^3, we consider the problem of computing the *convex hull* of P, conv(P), which is defined as the smallest convex set containing P. The convex hull problem has received considerable attention in computational geometry [11], [21], [23], [25]. In E^2 an algorithm known as *Graham's scan* [15] achieves $O(n \log n)$ running time, and in E^3 an algorithm by Preparata and Hong [24] has the same complexity. These algorithms are optimal in the worst case, but if h, the number of hull vertices, is small, then it is possible to obtain better time bounds. For example, in E^2, a simple algorithm called *Jarvis's march* [19] can construct the convex hull in $O(nh)$ time. This bound was later improved to $O(n \log h)$ by an algorithm due to Kirkpatrick and Seidel [20], who also provided a matching lower bound; a simplification of their algorithm has been recently reported by Chan *et al.* [2]. In E^3 an $O(nh)$-time algorithm can be obtained using the *gift-wrapping method*, an extension of Jarvis's march originated by Chand and Kapur [3]. A faster but more involved algorithm in E^3 was discovered by Edelsbrunner and Shi [13], having a running time of $O(n \log^2 h)$. Finally, by derandomizing an algorithm of Clarkson and Shor [8], Chazelle and Matoušek [7] succeeded in attaining optimal $O(n \log h)$ time in E^3. These

* This research was supported by a Killam Predoctoral Fellowship and an NSERC Postgraduate Scholarship.

Full Details

2. An Output-Sensitive Algorithm in Two Dimensions

Let $P \subset E^2$ be a set of $n \geq 3$ points. For simplicity, we assume that the points of P are in general position, i.e., no three points are collinear; see Section 4 for how to deal with degenerate point sets.

Recall that *Jarvis's march* [19], [23], [25] computes the h vertices of the convex hull one at a time, in counterclockwise (ccw) order, by a sequence of h *wrapping steps*: if p_{k-1} and p_k are the previous two vertices computed, then the next vertex p_{k+1} is set to be the point $p \in P$ that maximizes the angle $\angle p_{k-1} p_k p$ with $p \neq p_k$. One wrapping step can obviously be done in $O(n)$ time by scanning all n points; with an appropriate initialization the method constructs the entire convex hull in $O(nh)$ time.

We observe that a wrapping step can be done faster if we preprocess the points. Choose a parameter m between 1 and n and partition P into $\lceil n/m \rceil$ groups each of size at most m. Compute the convex hull of each group in $O(m \log m)$ time by, say, Graham's scan [15]. This gives us $\lceil n/m \rceil$ possibly overlapping convex polygons each with at most m vertices, after a preprocessing time of $O((n/m)(m \log m)) = O(n \log m)$. Now, a wrapping step can be done by scanning all $\lceil n/m \rceil$ polygons and computing tangents or supporting lines of the polygons through the current vertex p_k, as shown in Fig. 1. Since tangent finding takes logarithmic time for a convex polygon by binary or Fibonacci search [5], [25] (the dual problem is to intersect a convex polygon with a ray), the time required for a wrapping step is then $O((n/m) \log m)$. As h wrapping steps are needed to compute the hull, the total time of the algorithm becomes $O(n \log m + h((n/m) \log m)) = O(n(1+h/m) \log m)$.

The following is a pseudocode of the algorithm just described. The procedure always runs within $O(n(1 + H/m) \log m)$ time and successfully returns the list of vertices of $\text{conv}(P)$ in ccw order when $H \geq h$.

Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute $\text{conv}(P_i)$ by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P

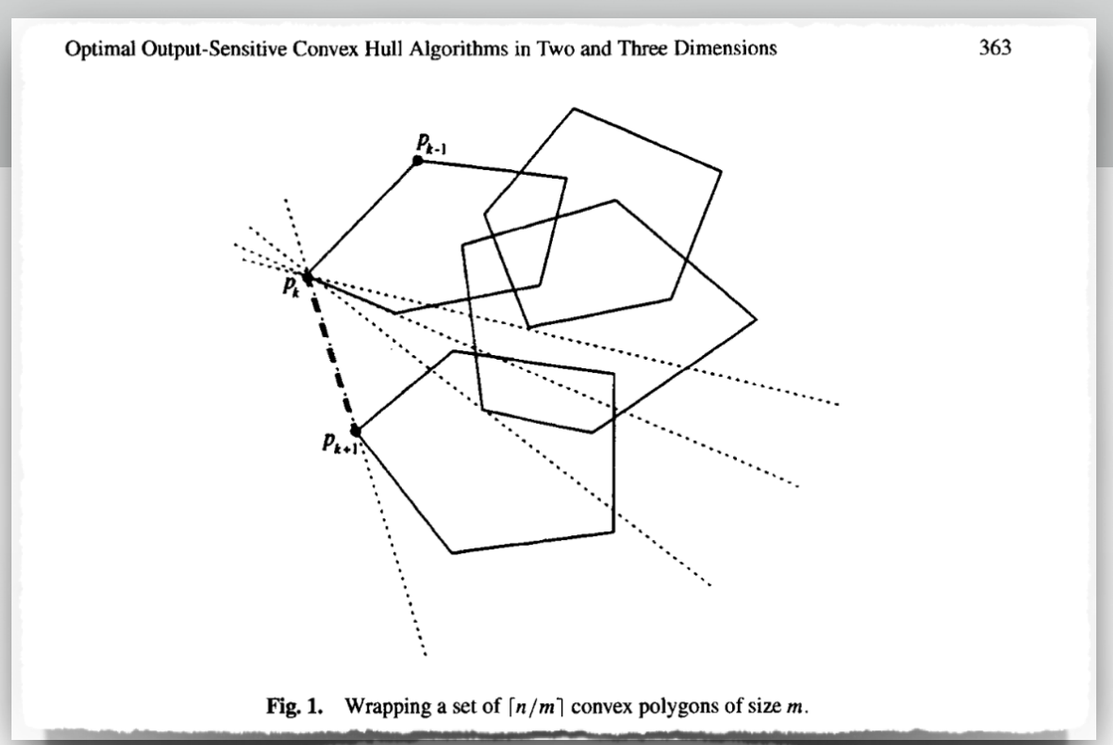


Fig. 1. Wrapping a set of $\lceil n/m \rceil$ convex polygons of size m.

6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of $\text{conv}(P_i)$
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

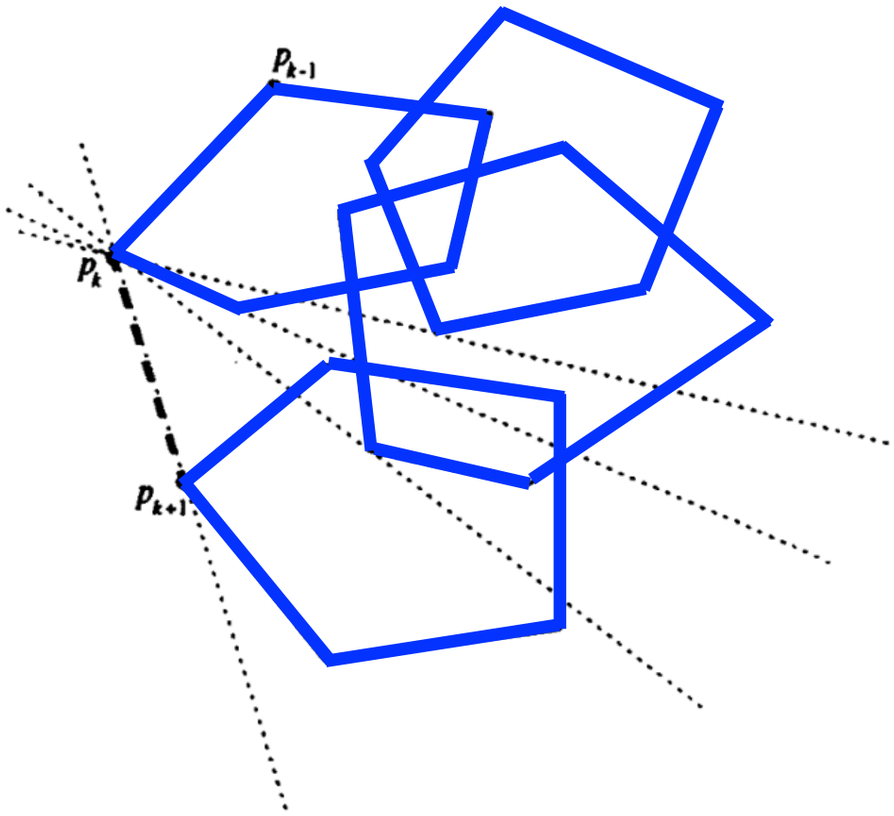
By choosing $m = H$, the complexity of the algorithm is then $O(n(1+H/m) \log m) = O(n \log H)$. Since the value of h is not known in advance, we use a sequence of H's to "guess" its value as shown below (the same strategy is used in Chazelle and Matoušek's algorithm):

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. if $L \neq$ *incomplete* then return L

The procedure stops with the list of hull vertices as soon as the value of H in the for-loop reaches or exceeds h. The number of iterations in the loop is $\lceil \log \log h \rceil$ (using base-2 logarithms), and the tth iteration takes $O(n \log H) = O(n2^t)$ time. Therefore, the total running time of the algorithm is $O(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t) = O(n2^{\lceil \log \log h \rceil + 1}) = O(n \log h)$. The storage requirement is clearly linear.

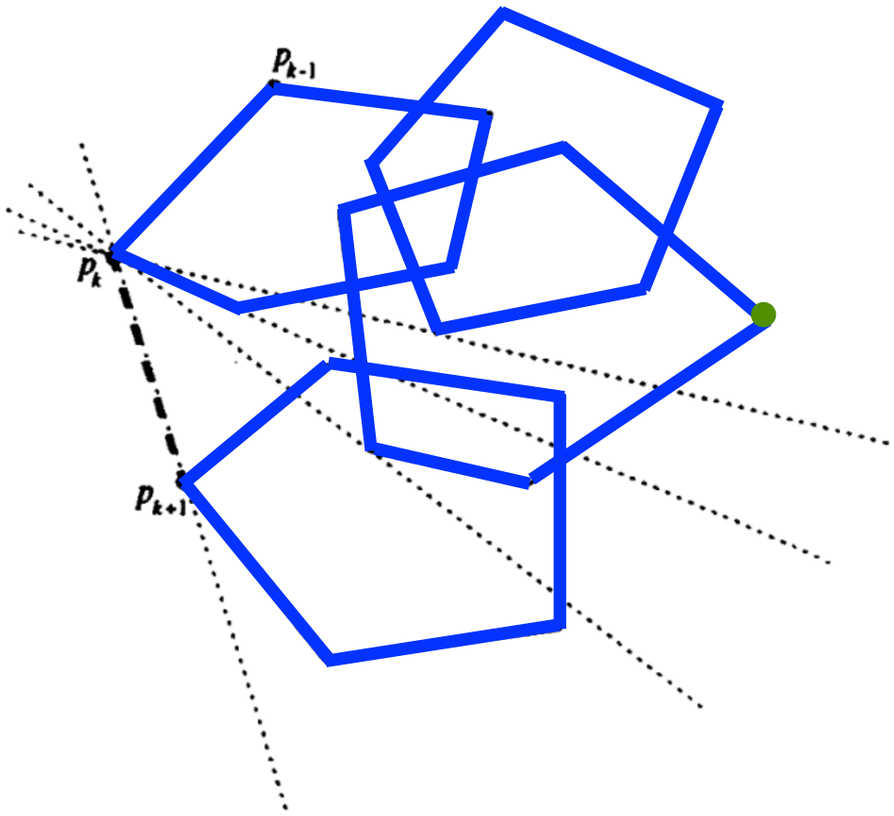
Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

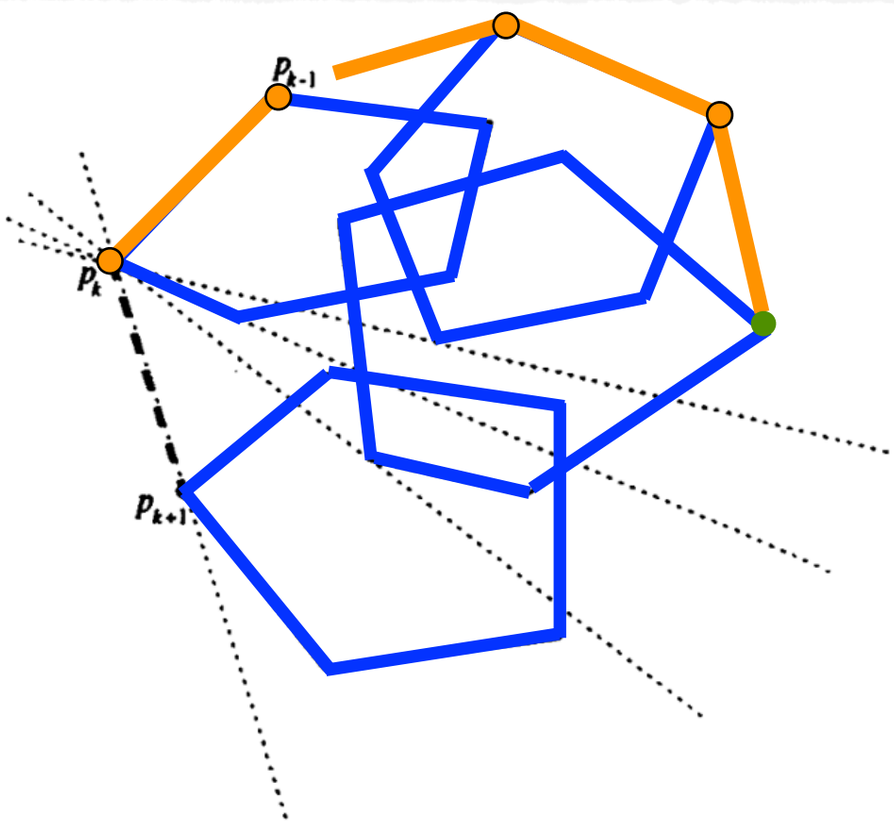
Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

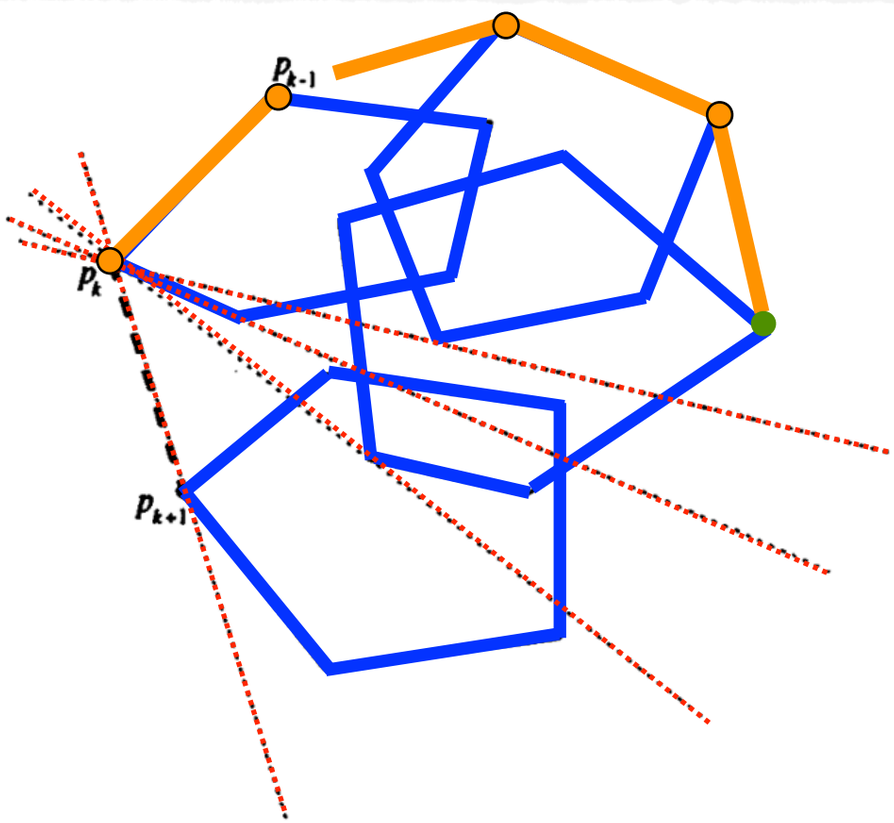
Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

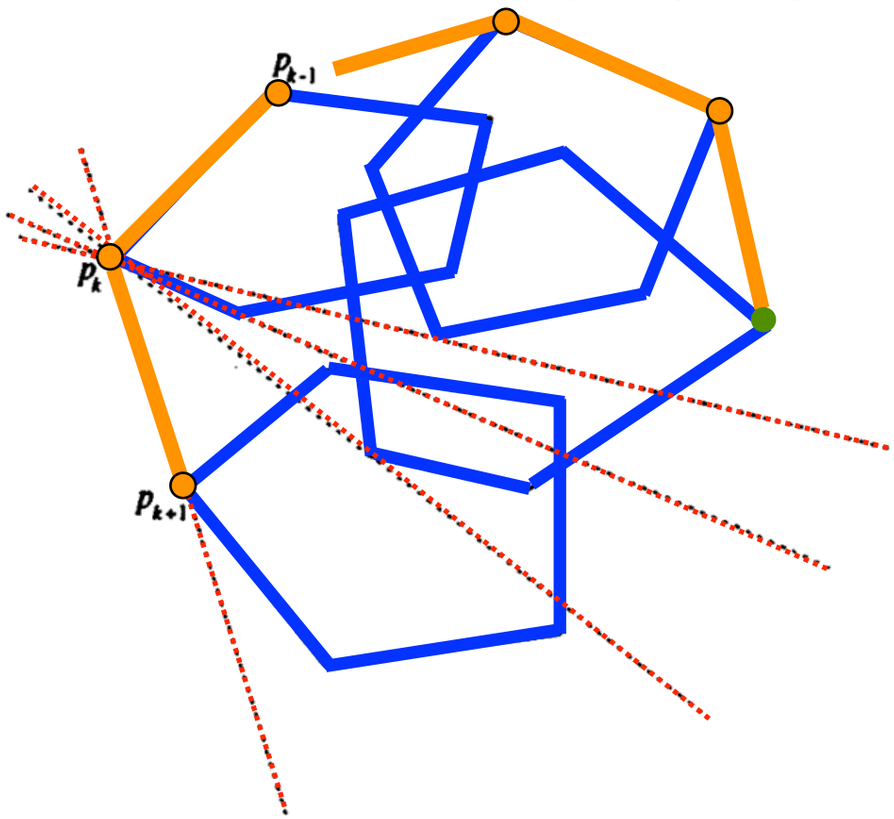
Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

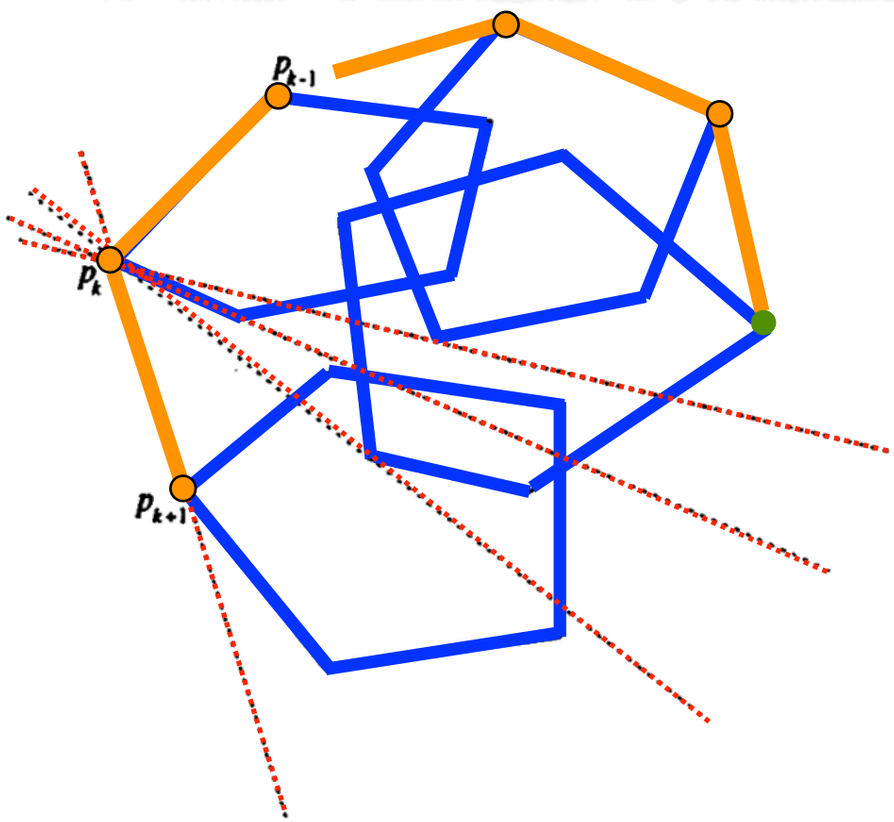
By choosing $m = H$, the complexity of the algorithm is then $O(n(1+H/m)\log m) = O(n \log H)$. Since the value of h is not known in advance, we use a sequence of H's to "guess" its value as shown below (the same strategy is used in Chazelle and Matoušek's algorithm):

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

The procedure stops with the list of hull vertices as soon as the value of H in the for-loop reaches or exceeds h. The number of iterations in the loop is $\lceil \log \log h \rceil$ (using base-2 logarithms), and the tth iteration takes $O(n \log H) = O(n2^t)$ time. Therefore, the total running time of the algorithm is $O(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t) = O(n2^{\lceil \log \log h \rceil + 1}) = O(n \log h)$. The storage requirement is clearly linear.

Algorithm



Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

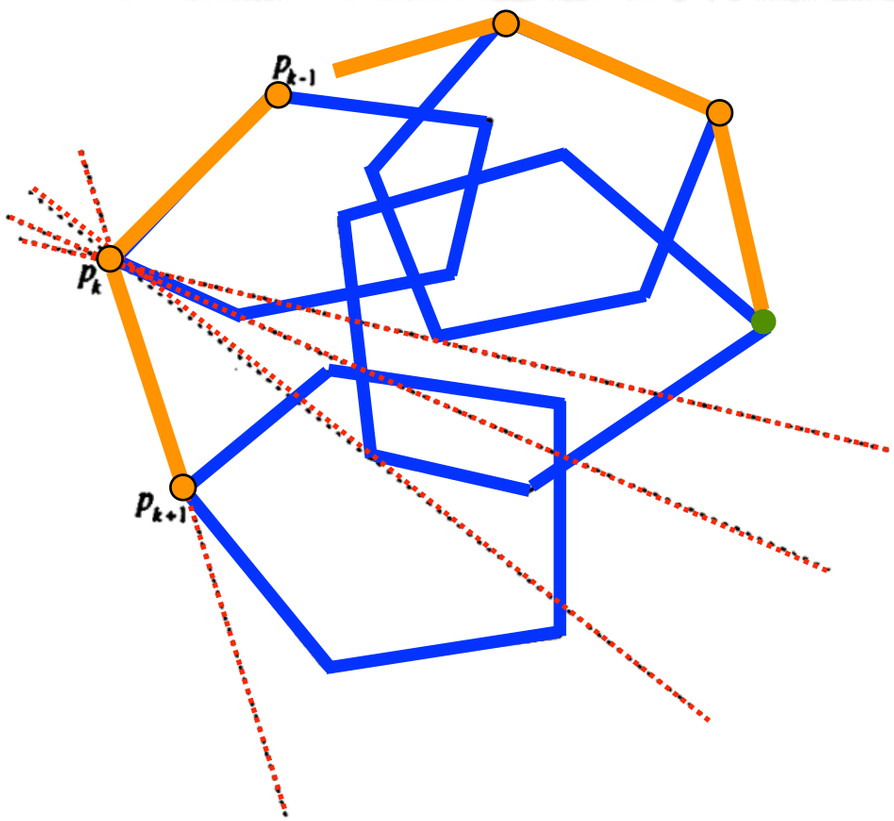
1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

Issues:

1. What is the runtime?
2. How do we find the tangent points?
3. How do we set the parameters m and H?
4. What is „*incomplete*“?
5. What is the *overall* runtime?

Algorithm

$\mathcal{O}((n/m)m \log m)$ $\mathcal{O}(H(n/m)T)$



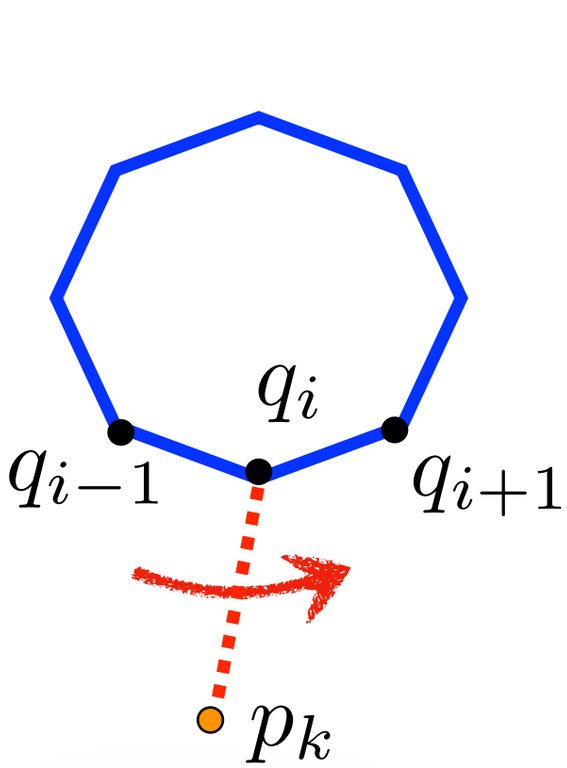
Algorithm Hull2D(P, m, H), where $P \subset E^2$, $3 \leq m \leq n$, and $H \geq 1$

1. partition P into subsets $P_1, \ldots, P_{\lceil n/m \rceil}$ each of size at most m
2. for $i = 1, \ldots, \lceil n/m \rceil$ do
3. compute conv(P_i) by Graham's scan and store its vertices in an array in ccw order
4. $p_0 \leftarrow (0, -\infty)$
5. $p_1 \leftarrow$ the rightmost point of P
6. for $k = 1, \ldots, H$ do
7. for $i = 1, \ldots, \lceil n/m \rceil$ do
8. compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of conv(P_i)
9. $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil n/m \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
10. if $p_{k+1} = p_1$ then return the list $\langle p_1, \ldots, p_k \rangle$
11. return *incomplete*

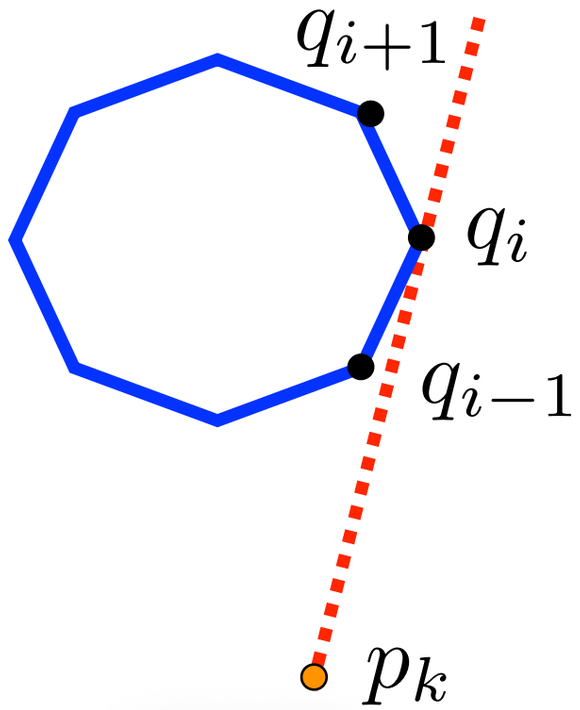
Issues:

1. What is the runtime?
2. How do we find the tangent points?
3. How do we set the parameters m and H?
4. What is „*incomplete*“?
5. What is the *overall* runtime?

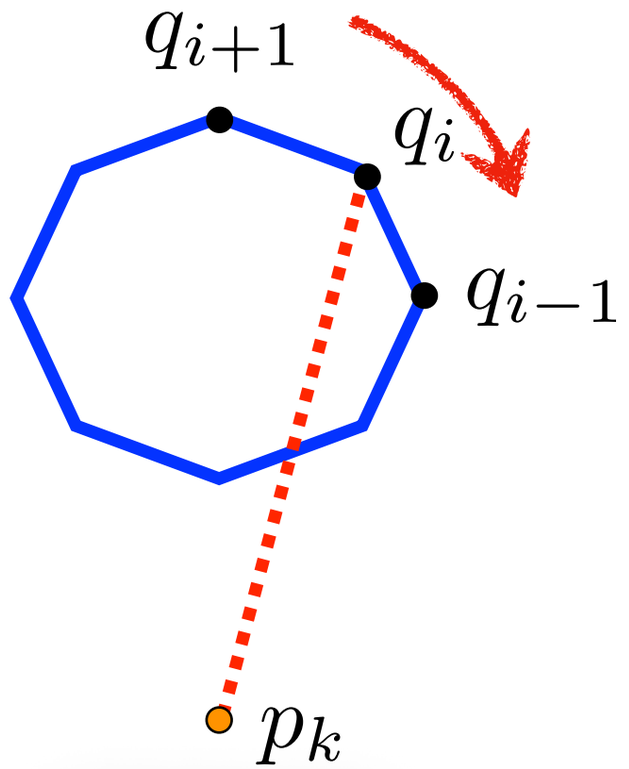
- Check relative position:



Increase i



Found i



Decrease i

- Binary search! $\rightarrow$ $T = (\log m)$

Choose $m=H$:

$$\mathcal{O}((n/m)m\log m) \quad +\mathcal{O}(H(n/m)\log m)$$

$$= \mathcal{O}((n\log m)$$

How do we find m ?

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

tth iteration takes $O(n \log H) = O(n2^t)$ time

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

tth iteration takes $O(n \log H) = O(n2^t)$ time

$O(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t)$

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

tth iteration takes $O(n \log H) = O(n2^t)$ time

$$O\left(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t\right) = O(n2^{\lceil \log \log h \rceil + 1})$$

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $\quad L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. $\quad$ if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

tth iteration takes $O(n \log H) = O(n2^t)$ time

$$O\left(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t\right) = O(n2^{\lceil \log \log h \rceil + 1}) = O(n \log h)$$

Algorithm Hull2D(P), where $P \subset E^2$

1. for $t = 1, 2, \ldots$ do
2. $L \leftarrow$ Hull2D(P, m, H), where $m = H = \min\{2^{2^t}, n\}$
3. if $L \neq$ *incomplete* then return L

The number of iterations in the loop is $\lceil \log \log h \rceil$

tth iteration takes $O(n \log H) = O(n2^t)$ time

$$O\left(\sum_{t=1}^{\lceil \log \log h \rceil} n2^t\right) = O(n2^{\lceil \log \log h \rceil + 1}) = O(n \log h)$$

Theorem 2.38 ([Chan, 1996])
Computing $conv(\mathcal{P})$ can be achieved in $\mathcal{O}(n \log h)$.

Refinements

4. Refinements

In this section we suggest ideas on possible improvements that may speed up our algorithms in practice; we also discuss how degenerate cases can be handled.

Idea 1. First, points found to be in the interior of conv(P_i) in line 3 of Hull2D(P, m, H) or Hull3D(P, m, H) can be eliminated from further consideration. This may potentially save work during future iterations of the algorithm, although it does not affect the worst-case complexity.

Idea 2. In Hull2D(P) and Hull3D(P) we choose the group size $m = H$ so as to balance the $O(n \log m)$ preprocessing cost and the $O(H((n/m) \log m))$ cost for the $O(H)$ wrapping steps. Alternatively, we can choose $m = \min\{H \log H, n\}$ (or set $H = m/\log m$). This choice of m does not affect the former cost except in the lower-order terms, but it reduces the latter cost from $O(n \log H)$ to $O(n)$ and thus results in a smaller constant factor overall.

Idea 3. With Idea 2, the dominant cost of algorithm Hull2D(P, m, H) lies in the preprocessing, i.e., the computation of the convex hulls of the groups in line 3. To reduce this cost, we may consider reusing hulls computed from the previous iteration and merging them as the group size is increased. Suppose m' is the previous group size. Since the convex hull of two convex polygons can be computed in linear time (the dual problem is to intersect two convex polygons), we can compute the convex hull of $\lceil m/m' \rceil$ convex m'-gons in $O(m \log(m/m'))$ time by the standard "mergehull" divide-and-conquer algorithm [25]. Thus, the $\lceil n/m \rceil$ hulls in line 3 can be constructed in $O(n \log(m/m'))$

Refinements

3D

Programming Techniques

G. Manacher S.L. Graham Editors

Convex Hulls of Finite Sets of Points in Two and Three Dimensions

F. P. Preparata and S. J. Hong
University of Illinois at Urbana-Champaign

The convex hulls of sets of n points in two and three dimensions can be determined with $O(n \log n)$ operations. The presented algorithms use the "divide and conquer" technique and recursively apply a merge

Refinements

3D

Programming Techniques
G. Manacher S.L. Graham
Editors

Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions*

T. M. Chan

F. P. Preparata and S. J. Hong

We can use the same grouping idea from the previous section to improve the time complexity to optimal $O(n \log h)$ while maintaining linear space. The calls to Graham's scan (line 3 of Hull2D(P, m, H)) are now replaced by calls to Preparata and Hong's three-dimensional convex hull algorithm [24], which has the same complexity. To make line 8

The convex hulls of sets of n points in two and three dimensions can be determined with $O(n \log n)$ operations. The presented algorithms use the "divide and conquer" technique and recursively apply a merge

References

[Aggarwal and Vittler, 1988] Aggarwal, Alok und Jeffrey Scott Vitter: The input/output complexity of sorting and related problems. Communicatios of the ACM 31(9):1116-1127, September 1988.

[Brönnimann et al**., 2004]** Brönnimann, Hervé, John Iacono, Jyrki Katajainen, Pat Morin, Jason Morrison und Gottfried T. Toussaint: Space-efficient planar convex hull algorithms. Theoretical Computer Science 321(1):25-40, Juni 2004.

[Chan, 1996] Chan, Timothy: Moon Yew: Optimal output sensitive convex hull algorithms in two and three dimensions. Computational Geometry: Theory and Applications 16(14): 361-368, April 1996.

[Chand and Kapur, 1970] Chand, Donald R. and Sham S. Kapur: An algorithm for convex polytopes. Journal of the ACM 17(1):78-86, Januar 1970.

[Clarkson and Shor, 1988] Clarkson, Kenneth L. und Peter W. Shor: Algorithms for diametral pairs and convex hull that are optimal, randomized, and incremental. In: Proceedings of the Fourth Annual Symposium on Computational Geometry, S. 12-17, ACM Press, 1988.

[de Berg et al**., 2008]** de Gerb, Mark, Otfried Cheong, Marc J. van Kreveld and Markus Hendrik Overmas: Computational Geometry: Algorithms and Applications. Springer, Berlin, 3. Auflage, 2008.

[Dobkin and Kirkpatrick, 1983] Dobkin, David Paul und David Taler Kirkpatrick: Fast detection of polyhedral intersection. Theoretical Computer Science 27(3):241-253, 1983.
[Eddy, 1977] Eddy, William F.: A new convex hull algorithm for planar point sets. ACM Transactions on Mathematical Software 3(4):398-403, Dezember 1977.
[Edelsbrunner, 1987] Edelsbrunner, Herbert: Algorithms in Combinatorial Geometry, ETACS Monographs on Theoretical Computer Science, Band 10. Springer-Verlag, Berlin, 1987.
[Frigo et al**., 1999]** Frigo, Matteo, Charles E. Leirson, Harald Prokop und Sidhar Ramachandran: Cache-oblivious algorithms. In: Proceedings of the 40th Symposium on the Foundation of Computer Science, S. 285-299, IEEE Computer Press, 1999.
[Graham, 1972] Graham, Ronald Lewis: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters 1(4):132-133, Juni 1972.
[Jarvis, 1973] Jarvis, R.A.: On the identification of the convex hull of a finite point set in the plane. Information Processing Letters 2(1):18-21, März 1973.
[Kallay, 1986] Kallay, Michael: Convex made easy. Information Processing Letters 22(3): 161, März 1986.
[Kirkpatrick und Seidel, 1986] Kirkpatrick, David Galer und Raimund Seidel: The ultimate planar convex hull algorithm? SIAM Journal of Computing 15(1):287-299, Februar 1986.

[Mulmuley, 1994] Mulmuley, Ketan: Computational Geometry: An Introduction Through Randomized Algorithms. Prentice-Hall, Englewood Cliffs, NJ, 1994.

[O'Rourke, 1994] O'Rourke, Joseph: Computational Geometry in C. Cambridge University Press, Cambridge, 2. Auflage, 1994.

[Preparata and Hong, 1977] Preparata, Franco P. und S. J. Hong: Convex hull of finite point sets in two and three dimensions. Communications of the ACM 20(2):87-93, Februar 1977.

[Seidel, 1993] Seidel, Raimund: Backwards analysis of randomized geometric algorithms: In: János Pach, Hg., New Trends in Discrete and Computational Geometry, Algorithms & Combinatorics, Band 10, Kapitel II, S. 37.67, Springer, Berlin, 1993.

Thank you!