Computational Geometry

2. Convex Hull

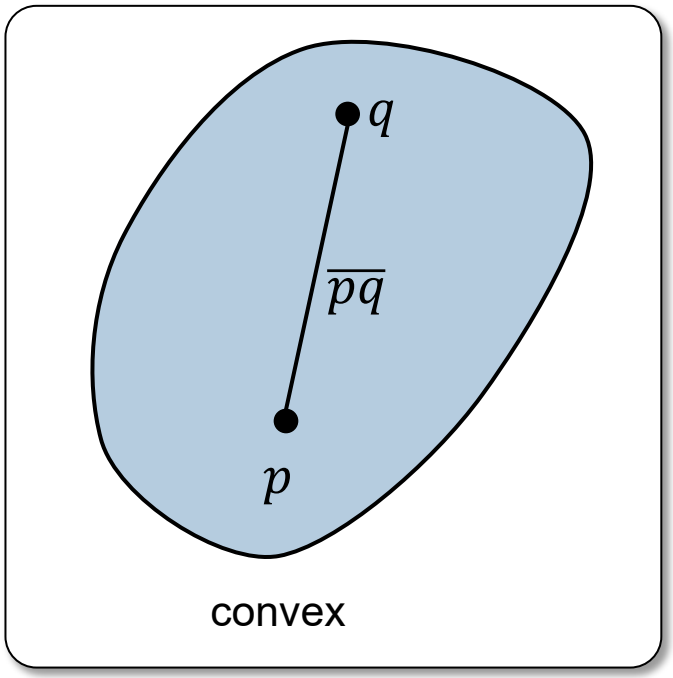
Contents

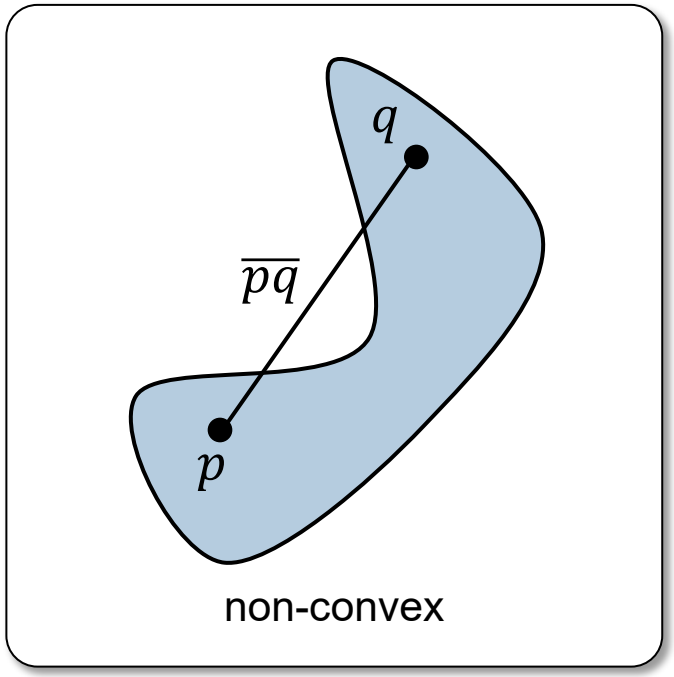
1. Definitions
2. A simple approach
3. A lower complexity bound
4. Graham's scan
5. Jarvis' march
6. Chan's algorithm

2.1 Definitions

Definition (Convex Set)

A set $S \subseteq \mathbb{R}^d$ is called **convex**, iff for all points $p, q \in S$ the connecting line segment $\overline{pq} = \{x : x = \lambda p + (1-\lambda)q, \lambda \in [0,1]\}$ is contained in S, i.e., $\overline{pq} \subseteq S$.



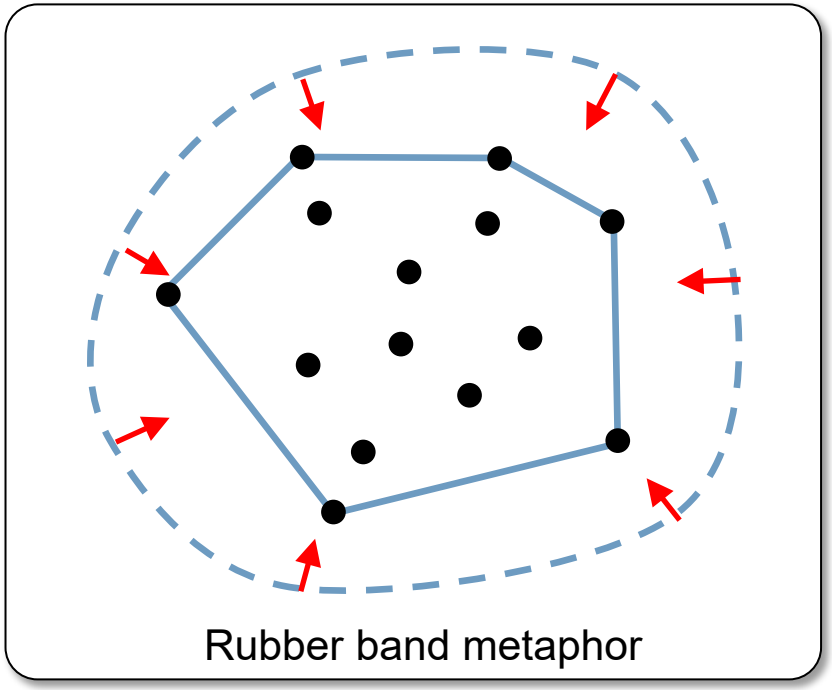


2.1 Definitions

Definition (Convex Hull)

The **convex hull** $\mathcal{CH}(S)$ of a set $S \subseteq \mathbb{R}^d$ is the smallest convex set containing S,

- i.e. it is the intersection of all convex sets containing S.



Rubber band metaphor

2.1 Definitions

- For $d = 2$ there is an alternative (informal) definition:

  The convex hull $\mathcal{CH}(P)$ of a finite set P of n points is the unique, convex polygon with corners from P containing all points of P.

- This definition is used to develop an algorithm to compute the convex hull of a set of 2d-points.

Remark: Convex hull algorithms are to Computational Geometry what sorting is to discrete algorithms.

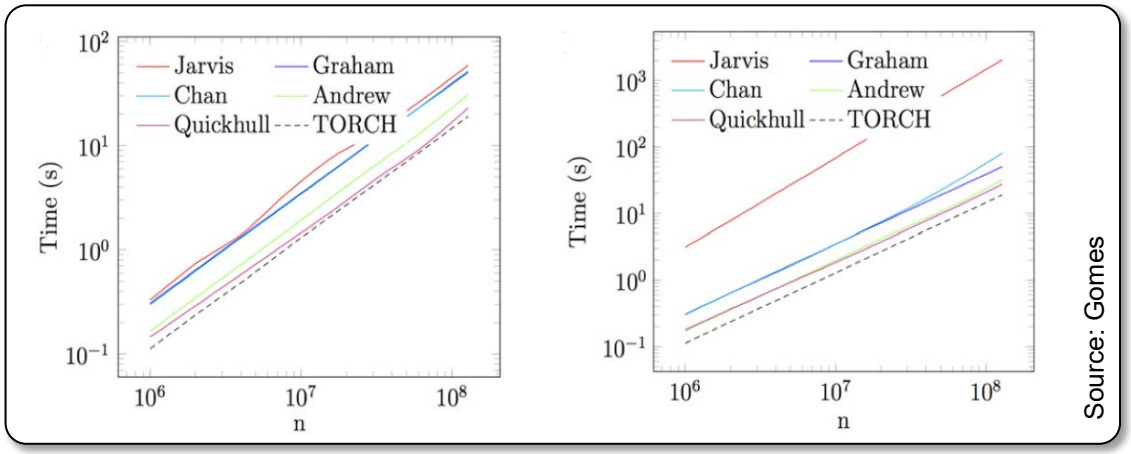
Remark: There are many algorithms to compute the convex hull of a point set P.

2.1 Definitions

Remark

There are many algorithms to compute the convex hull of a 2d point set P:

1. **Incremental:** Naïve, Kallay: $O(n^3)$, $O(n \log n)$.
2. **Recursive:** Divide-and-conquer, QuickHull: $O(n \log n)$.
3. **Sweeping:** Graham, Andrew, TORCH: $O(n \log n)$.
4. **Output-size sensitive:** Jarvis, Chan: $O(n\,k)$, $O(n \log k)$, if $\mathcal{CH}(P)$ is a k-gon.



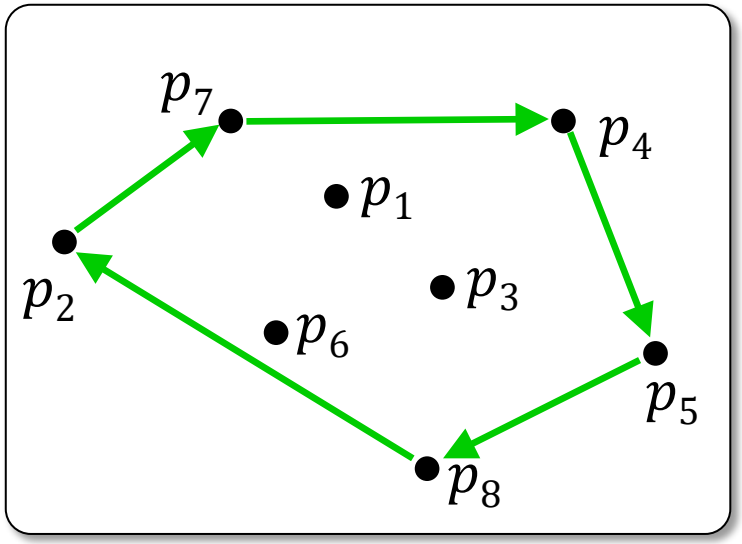
2.2 Simple approach

Input: Set $P = \{p_1, p_2, \ldots, p_n\}$ of n mutually distinct points in the plane.

Output: Sorted list of points from P (clockwise), representing the corners of the convex hull $\mathcal{CH}(P)$.

Example:

- Input: $(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$
- Output: $(p_4, p_5, p_8, p_2, p_7)$



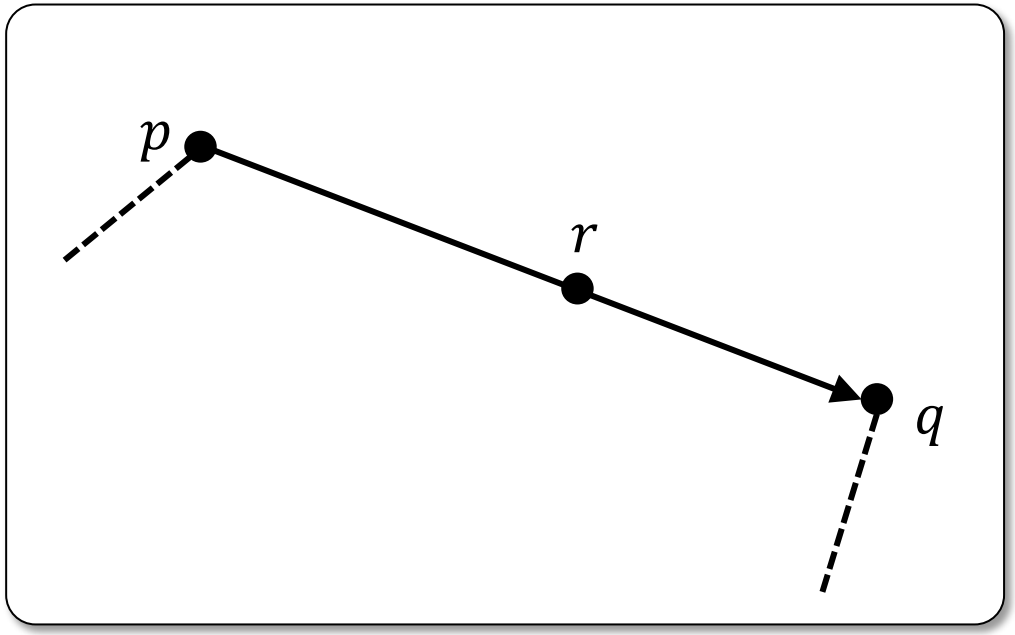
- **Idea**: A directed edge $\overline{pq}$ is part of the convex hull, if no other point of P lies left of it.

2.2 Simple approach

```
 1 SortedList SlowConvexHull(PointSet P)
 2 {
 3    PointSet E = ∅;
 4    for all (oriented pairs (p,q)∈P² with p≠q){
 5       valid = true;
 6       for all (points r∈P\{p,q}){
 7          if (r lies left of the oriented line pq)
 8             valid = false;
 9       }
10       if (valid) add the oriented line pq to E;
11    }
12    SortedList L = ConstructSortedList(E);
13    return L;
14 }
```

2.2 Simple approach

- This algorithm is **(almost)** correct.
  - Neglect rounding errors from floating point arithmetic!
  - Lines 3-11 yield an unsorted list of all edges of the convex hull.
  - But, there might be degenerate cases!



2.2 Simple approach

- Run time of the algorithm SlowConvexHull:
  - Line 4: $n^2 - n = n(n-1)$ pairs of points.
  - Line 6: $n - 2$ points are tested.
  - Line 7: The test to decide if a point lies left/right of a line is a primitive operation, so it is computable in constant time.
  - ▶ Lines 3-11 yield a run time of $O(n^3)$.
  - Line 12: The brute force approach has run time $O(n^2)$.
- ▶ Total run time: $O(n^3) + O(n^2) = O(n^3)$.
- This can be improved?!

2.3 Lower Bound

- The minimal run time to compute the convex hull of n points in 2d is $\Omega(n \log n)$.

Proposition 1

The sorting problem can be transformed in linear time to the convex hull problem.

Thus, the convex hull of n points in the plane can be computed in $\Omega(n \log n)$.

- **Proof:** See white board.

2.3 Lower Bound

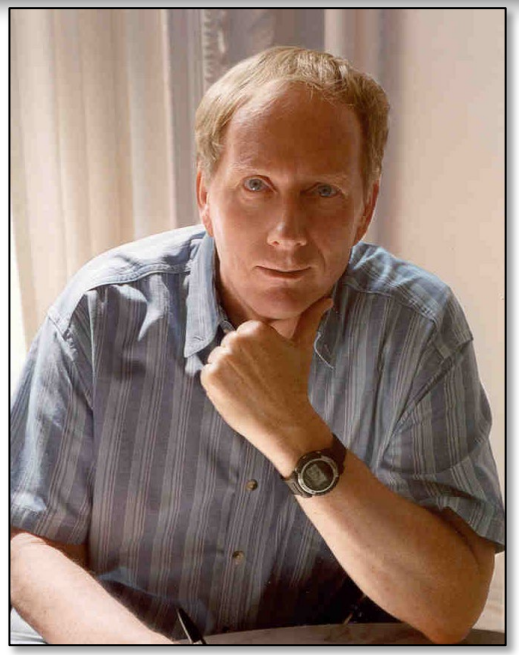
Remark:

The complexity of the convex hull problem in 2d is $O(n \log n)$.

Remark: Four approaches:

1. Naïve approach: $O(n^3)$.
2. Divide-and-conquer approach: $O(n \cdot \log n)$.
3. Scan line approach, Graham's scan: $O(n \cdot \log n)$.
4. Output-size sensitive approach (if the convex hull is a k-gon), Jarvis' march: $O(k \cdot n)$.
5. Output-size sensitive, Chan $O(n \cdot \log k)$

2.4 Graham´s Scan



- Graham's Scan was developed in 1972 by Ronald L. Graham.
- Improved by M.A. Andrew in 1979.

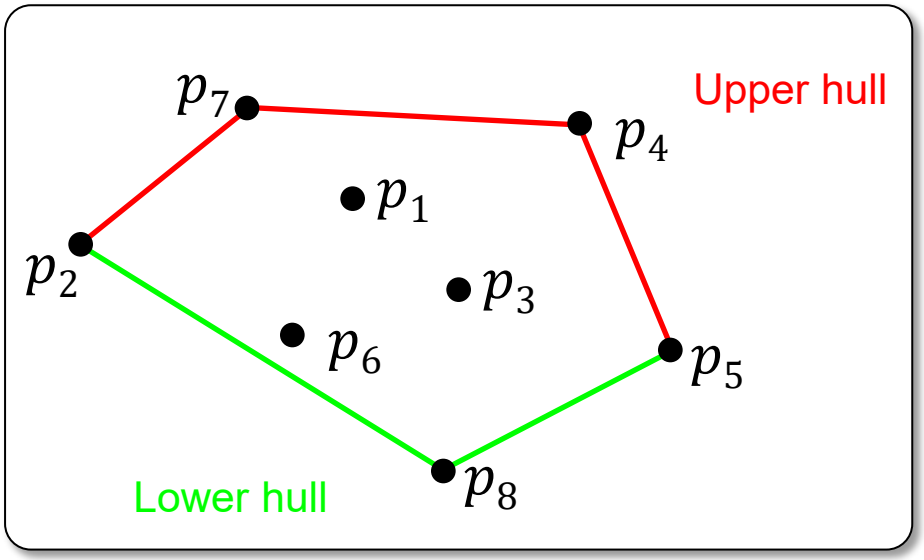
- Incremental algorithm
  - Points are analyzed one by one and added successively to the solution.
  - This is done left to right.
  - Therefore, the points are sorted by their x-coordinate.
  - **Problem:** The knots of the convex hull should also be added to the convex hull from left to right.

2.4 Graham´s Scan

- **Solution**

  Compute first the vertices of the *upper convex hull* and then the vertices of the *lower convex hull*.

  - The points to split the convex hull in the upper and lower convex hull are determined by the minimal and maximal x-coordinate



2.4.1 Algorithm

```
 1  Polygon GrahamScan(Points P) {
 2     Sort points by x-coordinate (p1,...,pn);
 3     Lupper.append(p1); Lupper.append(p2);
 4     for (i=3; i<=n; i++) {
 5        Lupper.append(pi);
 6        while (Lupper containes more than two points and
 7               the last three points a,b,c make a left turn)
 8           Lupper.delete(b);
 9     }
10     Llower.append(pn); Llower.append(pn-1);
11     for (i=n-2; i>=1; i--) {
12        Llower.append(pi);
13        while (Llower containes more than two points and
14               the last three points a,b,c make a left turn)
15           Llower.delete(b);
16     }
17     Llower.delete(first point); Llower.delete(last point);
18     return Lupper.append(Llower);
19  }
```

2.4.2 Run time

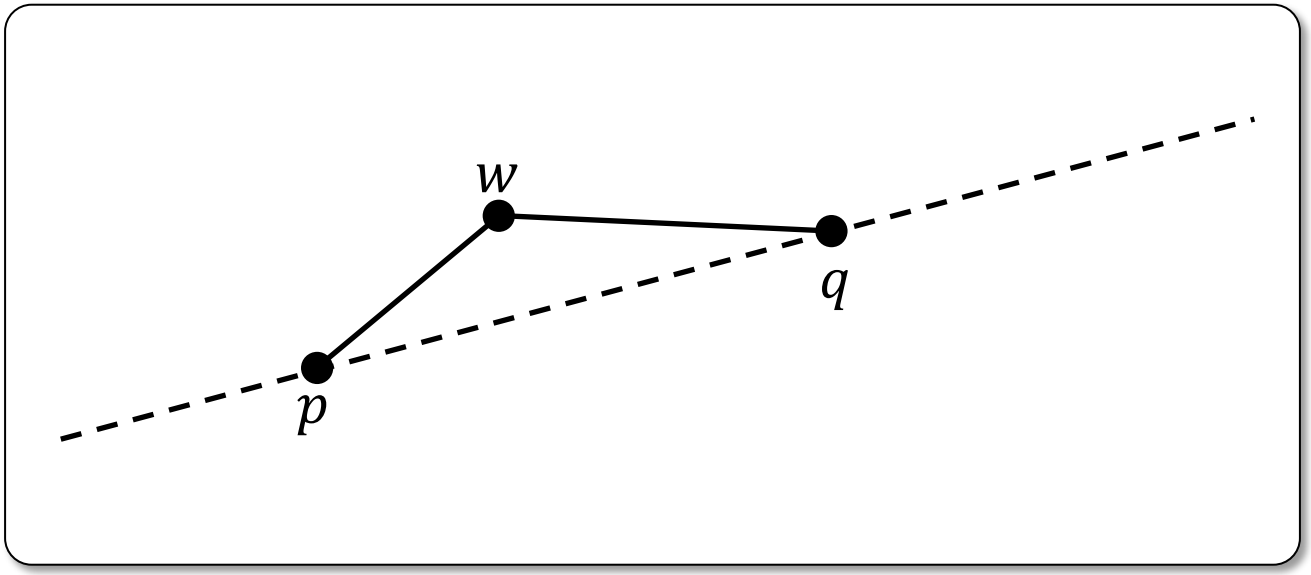
Proposition 2

The convex hull of a set of n points in the plane can be computed in $O(n \log n)$.

- Proof of correctness of the algorithm
  - Induction on the number of processed points.
- Run time
  - Lexicographical sorting: $O(n \log n)$.
  - Upper Hull
    - Loop in line 4: $n - 3$ repetitions.
    - Loop in line 6: At most n points are removed in total.
    - ► Total run time of the upper hull: $O(n)$.
  - Lower hull analogously.
  - ► Total run time: $O(n \log n)$.

2.4.1 Algorithm

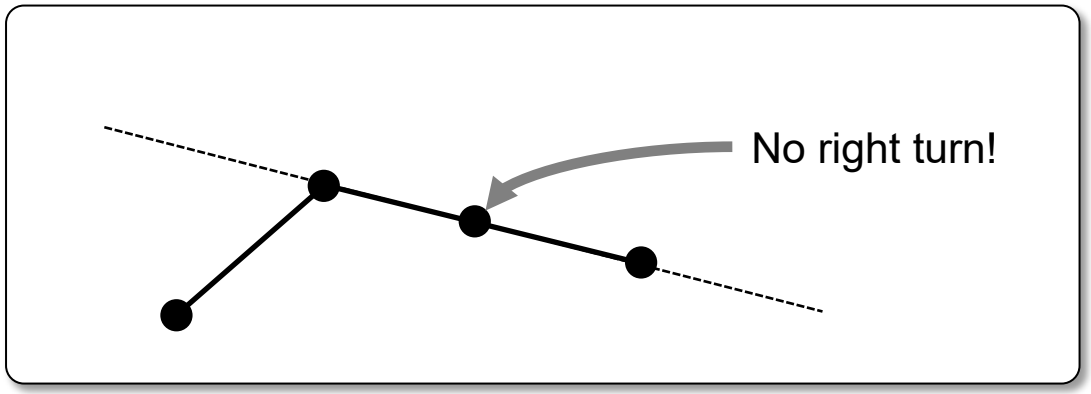
- The test for a right turn is easy to compute:
  - Three points (p, w, q) have a right turn, if w lies left of the line segment $\overline{pq}$.



2.4.1 Algorithm

- The Algorithm is not correct!
  - If several points have the same x-coordinate!
    - Thus, the chosen order is not well-defined.
    - Solution: Sorting the points *lexicographically*.

      If two points have equal x-coordinates, sort them by their y-coordinate.
- Another special case occurs if three points in the right-turn-test lie on a line!



2.5 Jarvis´ March



- 1973 developed by Ray A. Jarvis.
- „Gift Wrapping“-method
- **Idea:** Wrap a cord around the set of points until it coincides with the convex hull.
  - Analog to Graham´s Scan start with the lexicographically smallest point. This ensures that it lies on the convex hull!
    Assumption: Search first for the smallest y-coordinate and then for the smallest x-coordinate.
  - Jarvis´ March computes a sequence of „extreme“ points of the convex hull in counter-clockwise order.

2.5.1 Algorithm

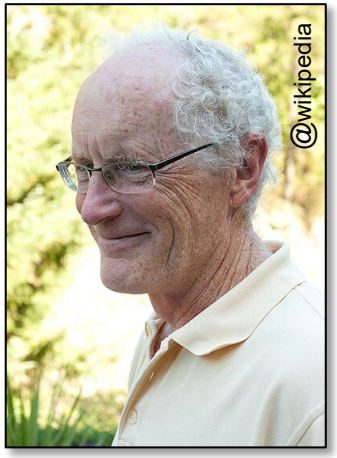
```
 1  Polygon JarvisMarch(Points P)
 2  {
 3     Find lexicographically smallest point p_1;
 4     q_1 = p_1;
 5     i = 2;
 6     q_i = end point of line p_1q_i with smallest angle to
           horizontal line through p_1;
 7     repeat {
 8        i++;
 9        q_i = end point of line q_{i-1}q_i with smallest angle
              to line q_{i-2}q_{i-1} at q_{i-1};
10     } until (q_i == p_1);
11     Add all points q_1,...,q_k to a list L;
12     return L;
13  }
```

2.5.2 Run time

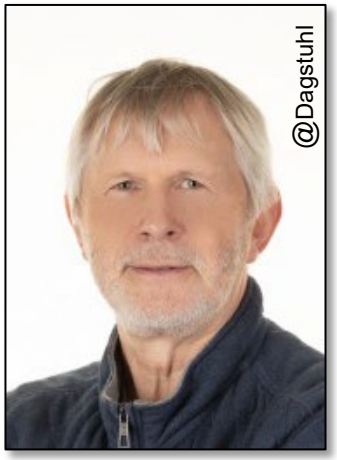
- Denote by h the number of corners of the convex hull.
  - Computing q_i has runtime $O(n)$, because a smallest angle is computed by arithmetic operations and comparisons only.
- The total run time is now $O(h \cdot n)$
  - For small h this algorithm is very efficient!
  - For large h it is inefficient, because its worst-case run time is $O(n^2)$.
  - Such a behavior is called *output-size sensitive*.
- What is the complexity for output-size sensitive convex-hull algorithms?

2.6 Chan's algorithm

- What is the complexity for output-size sensitive convex-hull algorithms?
  - The complexity for output-size sensitive convex-hull algorithms is $\Omega(n \cdot \log h)$ [Kirkpatrick/Seidel 1986].
  - This bound is achieved by Chan's algorithm, aka Nielson's algorithm, 1995/96, which is a combination of Graham's scan and Jarvis' march.
    - Optimizations by Emo Welzl.



David Galen Kirkpatrick



Raimund Seidel



Timothy Chan



Frank Nielsen



Emo Welzl

2.6 Chan's Algorithm

```
 1 Polygon ChansAlgorithm(Points P)
 2 {
 3   for (t=1,2,… until hull is found) {
 4     H = min(2^(2^t),n);  // estimate for number corners h
 5     Partition P into k=n/m subsets P_1,…,P_k of size ≤m;
 6     Compute CH(P_1),…,CH(P_k) using Graham's scan;
 7     Find lexicogr. smallest point p_1; p_0=(-∞,0);
 8     for (j=1,…,H) {  // try to find H-corner-hull
 9       for (i=1,…,k) // wrapping of CH(P_1),…,CH(P_k)
10         Compute q_i∈CH(P_i)\{p_j} that maximizes ∢p_(j-1)p_jq_i
           using "binary search" on CH(P_i);
11       p_(j+1)=argmax(q∈{q_1,…,q_k} that maximizes ∢p_(j-1)p_jq);
12       if (p_(j+1)=p_1) return hull is found;
13     }  // H too small, could not find H-corner-hull
14   }    // improve estimate for h
15 }
```

2.6 Chan's Algorithm

- Structure of the algorithm

```
1  Polygon ChansAlgorithm(Points P)
2  {
3    for (t=1,2,… until hull is found) {
4      H = min(2^(2^t),n); // estimate for number corners h
5      Partition P into k=n/m subsets P_1,…,P_k of size ≤m;
6      Compute CH(P_1),…,CH(P_k) using Graham's scan;
7      Find lexicogr. smallest point p_1; p_0=(-∞,0);
8      for (j=1,…,H) {// try to find H-corner-hull
9        for (i=1,…,k)// wrapping of CH(P_1),…,CH(P_k)
10         Compute q_i∈CH(P_i)\{p_j} that maximizes ∢p_(j-1)p_jq_i
           using "binary search" on CH(P_i);
11       p_(j+1)=argmax(q∈{q_1,…,q_k} that maximizes ∢p_(j-1)p_jq);
12       if (p_(j+1)=p_1) return hull is found;
13     } // H too small, could not find H-corner-hull
14   }   // improve estimate for h
15 }
```

Pre-processing

Graham scan

Jarvis's march

Iterate to find the unknown number of corners h.

2.6.1 Run time

- Line 5, aka partitioning: $O(n)$
- Line 6, aka convex hull of partitions: $O(n/m \cdot m \log m)$
- Line 7, aka initialize wrapping: $O(n)$
- Line 8, number of iterations $O(H)$
- Line 9, number of iterations $O(n/m)$
- Line 10, aka wrap every hull: $O(\log m)$
- Line 11, aka wrap of hulls: $O(n/m)$
- Total runtime for lines 4-13:

$$O\left(n + n\log m + n + H\left(\frac{n}{m}\log m + \frac{n}{m}\right)\right) = O\left(n\log m + H\frac{n}{m}\log m\right)$$

2.6.1 Run time

Reminder: $O\left(n \log m + H \frac{n}{m} \log m\right)$

- How to choose m and H?

1. For $m = H$, lines 3+4 iterate until (at worst $H = n$ or)

$$h \leq H = 2^{2^t} \quad \Leftrightarrow \quad \log_2 h \leq 2^t \quad \Leftrightarrow \quad \log_2(\log_2 h) \leq t.$$

- This yields for the complete algorithm the runtime

$$\sum_{t=1}^{\log\log h} O\left(n \log 2^{2^t}\right) = O(n) \sum_{t=1}^{\log\log h} 2^t = O\left(n \cdot 2^{\log\log h}\right) = O(n \log h).$$

2. A heuristically better choice in line 4 is $H = \min\left(2^{t^2}, n\right)$.
3. A good balance of m and H is given by $m = \min(H \log H, n)$.
4. Use in line 10 linear search (aka convex hull of convex hulls), because it is simpler to implement and does not affect the total runtime [Welzl].
   - The runtime for the wrapping is smaller.