

---

# Implicit Computation Geometry

---

Henrik Blunck

Department of Computer Science  
WWU Münster

# Overview

---

1. Introduction: Motivation for implicit computation
2. Skylines and convex hulls

# Motivation

---

- Traditional focus in algorithm design: Running Time
- Here: Second core issue: Memory utilization.
- Historically: Space-Efficiency considered due to high memory prices.
- Nowadays: Space-Efficiency considered due to:
  - Larger datasets.
    - \* High-resolution surveillance data
    - \* Temporal and spatio-temporal data
  - Smaller computing devices.
    - \* Location based services for mobile communication networks
    - \* Data analysis and propagation in sensor networks
  - Limited (read/write)-memory



Sensor Networks



Car Navigation

## In-Place Algorithms

---

### Definition 1.1

An algorithm  $A$  is called **in-place** iff during its execution  $A$  occupies  $\mathcal{O}(1)$  words, i.e.  $\mathcal{O}(\log_2 n)$  bits, in addition to the space required by the input.

(Assumption: Any pointer or data item occupies  $\mathcal{O}(1)$  words.)

### Consequences:

- Classic **recursive** algorithms are not in-place.
  - Need to maintain a **call stack** of size  $\Omega(\log n)$ .
- Algorithms using auxiliary pointer-based data structures (such as balanced binary trees or linked lists) are not in-place.
  - Need to resort to **implicit** data structures.

### Example:

- Heapsort is an in-place algorithm (uses in-place data structure).
-

## Motivation: Dealing with Large Datasets

---

### Algorithmic concepts for different scenarios:

- Small, fast working memory. Data resides on slow disks.
  - Cache-oblivious and I/O-efficient Algorithms: Minimize data (block) movement.
- Data is streamed and not constantly available.
  - Streaming algorithms: (Approximation of) data aggregates.
- (Almost) no memory to use additional to the given input.
- Implicit data structures, in-place algorithms.

### More motivation:

- “The less memory used, the faster ..”
  - Because of: Memory-, disk-, network latencies, less garbage to collect, larger basecases ...
  - In-place model “in between” I/O- and Streaming-Model ...
  - May provide insights in computational complexity of problems.
-

## Previous Results

---

### In-Place Sorting and Related Problems:

- Heapsort [Floyd, 1964].
- Linear-time [merging/partitioning](#) [Mannila & Ukkonen, 1984; Geffert et al., 2000; Katajainen & Pasanen, 1999]. . .
- Linear-time [k-selection](#) [Carlsson & Sundström, 1995; Geffert & Kollar, 2001; Bose et al., 2006].

### In-Place, Cache-Oblivious(!) Dictionary:

- $\mathcal{O}(\log n)$  update/queries [Franceschini & Grossi, 2003].

# Sapce-efficient Computational Geometry Results

---

## In-Place Computational Geometry:

- Closest Pair etc. [Bose et al., 2006].
- Line-Segment Intersection [Bose et al., 2006; Vahrenhold, 2005].
- Convex Hull and Maxima problems etc. [Brönnimann et al., 2004b; Brönnimann & M.Chan, 2004; Blunck & Vahrenhold, 2006].

## “Use-Polylog-Extra-Space-And-Time” Geometry Results:

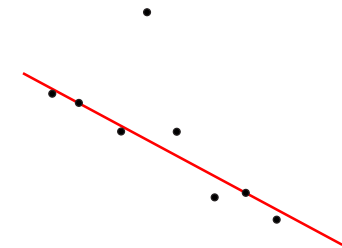
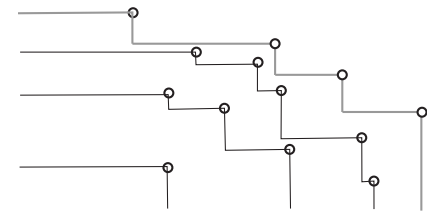
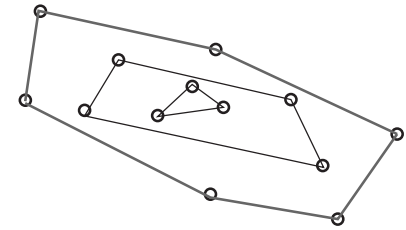
- $3d$ -convex hull and related [Brönnimann et al., 2004c].
- Multidimensional search sctructures [Brönnimann et al., 2004a].
- Klee’s Measure Problem [Chen & M.Chan, 2005].

## In this lecture . . .

---

### In-place techniques and algorithms for:

- Convex hulls and sets of maxima ('skylines')
- Layers of convex hulls and maxima
- Regression Analysis: Estimating linear correlations





# Overview

---

1. Introduction: Motivation for implicit computation
2. Skylines and convex hulls

# Computing the Skyline

---

## Maximal Points:

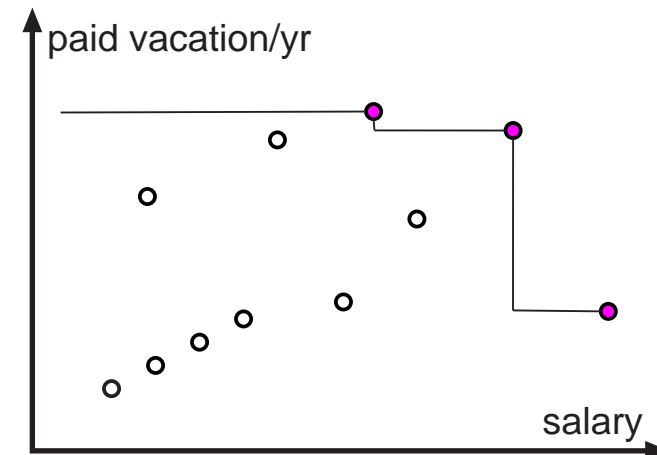
- Given: Set  $\mathcal{P}$  of  $n$  points in the plane.
- $p \in \mathcal{P}$  is **maximal**  $\Leftrightarrow$   
 $\forall q \in \mathcal{P} : p.x \geq q.x \vee p.y \geq q.y$
- i.e.,  $p \in \mathcal{P}$  is **maximal** iff no other  $q \in \mathcal{P}$  in “upper-right quadrant” of  $p$ .
- Union of maximal points:  
‘skyline’, ‘pareto-optimal points’.

## Objective:

- Find “points” that cannot be “optimized” in *all*  $d$  dimensions.

## Generalizations:

- Definition generalizes to:
  - Arbitrary dimensions  $d$ .
  - ‘Maxima’ w.r.t.  $d$  arbitrary chosen coordinate axes.



## Computing all skylines in-place

---

### Layers of Maxima:

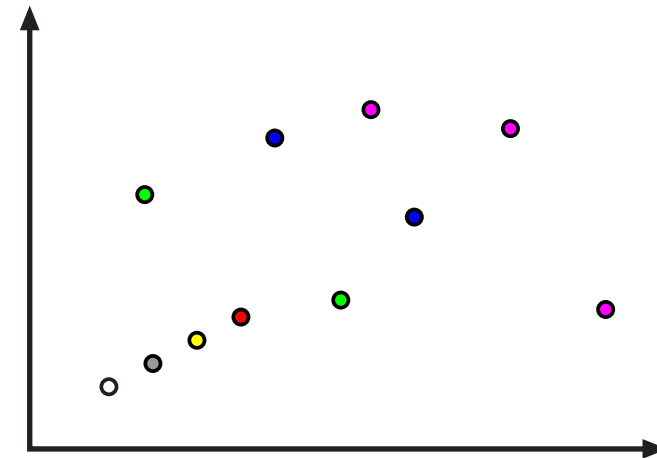
- Compute 'skyline'  $\text{MAX}(\mathcal{P})$  of  $\mathcal{P}$ .
- If  $\mathcal{P} \setminus \text{MAX}(\mathcal{P})$  not empty, set  $\mathcal{P} := \text{MAX}(\mathcal{P})$  and repeat.
- Number of iterations (layers) can be linear in  $n$ .

### Convex Layers:

- defined analogical.

### In-place setting:

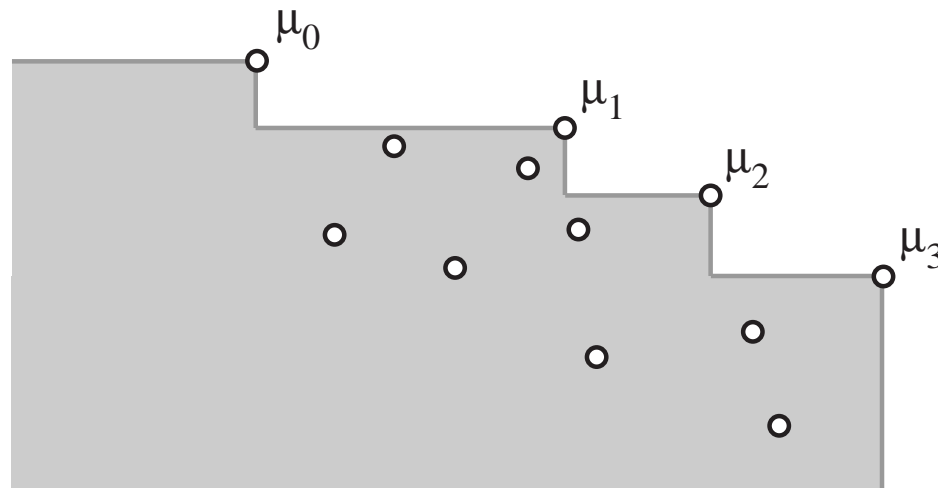
- Group points by layer.
- In each layer: points sorted (by  $x$ ).



## Computing the Skyline: Selecting Maximal Points in 2D

---

- Presort points by  $y$ -coordinate using, e.g., heapsort.
- Sweep top-down, maintain lowest maximal point (= 'tail') seen so far.
  - **Invariant**: Next point  $p$ : is maximum  $\Rightarrow p$  right of tail.
- Swap maximal points to the front of the array.



$\mu_i$  : maximal  
points

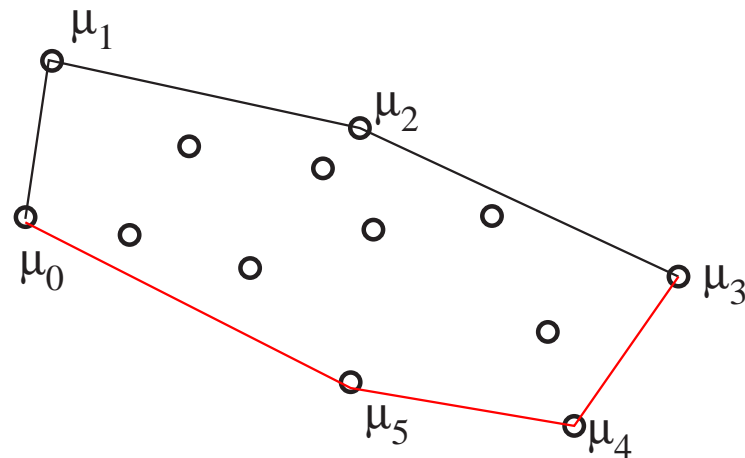


- “Skyline” (in  $<_x$ -order): In-place,  $\mathcal{O}(n \log n)$  (sort & scan).

## Computing the Convex Hull in 2D:Graham's Scan

---

- Presort points lexicographically using, e.g., heapsort.
- Sweep left-right, maintain last two convex hull points seen so far.
  - **Invariant**: Point  $q$  **not** on upper convex hull iff  $(p, q, r)$  form a left-turn.
- Swap upper hull points to the front of the array; then compute lower hull ..



$\mu_i$  : convex hull points

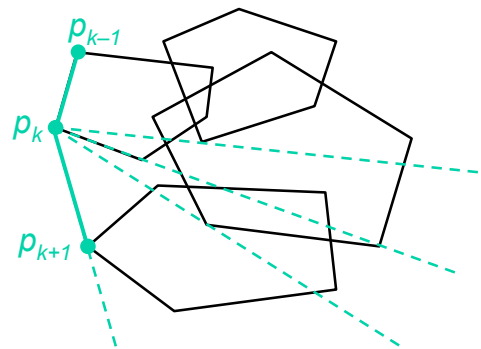
$\mu_0$	$\mu_1$	$\mu_2$	$\mu_3$	$\mu_4$	$\mu_5$	○	○	○	○	○	○
---------	---------	---------	---------	---------	---------	---	---	---	---	---	---

- Convex hull: In-place,  $\mathcal{O}(n \log n)$  (sort & scan) [Graham, 1972].

# Computing the Convex Hull in 2D output-sensitive [Chan, 1996]

## Chan's output-sensitive algorithm:

- Avoid global presorting (if  $h \ll n$ ).
- Speed up Jarvis' March:
  - Bundle input in  $H$  groups  $G_1, \dots, G_H$
  - Graham-Scan each group to build  $CH(G_1), \dots, CH(G_H)$ .
  - Jarvis-march with  $CH(G_1), \dots, CH(G_H)$  (instead of points) as input-objects.
    - \* To find next CH-vertex  $p_k + 1$ : Compute tangents to  $G_1, \dots, G_H$ , each by binary search on a  $CH(G_i)$ .
  - $\Rightarrow$  Running time:  $\mathcal{O}(n/H \cdot H \cdot \log_2 H + H \cdot n/H \cdot \log_2 H) = \mathcal{O}(n \cdot \log_2 H)$
- Doing rounds: "Guess"  $h$  by choosing  $H = 2^{2^1}, 2^{2^2}, \dots$
- $\Rightarrow$  Global running time (with  $\log_2 H = 2^t$ ):  
 $\mathcal{O}\left(\sum_t^{\lceil \log_2 \log_2 h \rceil} n \cdot 2^t\right) = \mathcal{O}\left(n \cdot 2^{\lceil \log_2 \log_2 h \rceil + 1}\right) = \mathcal{O}(n \cdot \log_2 h)$ .



## How to run Chan's algorithm in-place?

---

### Problems to solve:

- $H$  can be stored in single extra word  $\Rightarrow$  Only one round to consider.
- Where to (out)put convex hull vertices?
- Where to store convex hulls of the groups  $G_1, \dots, G_H$ ?
  - How to arrange w/o extra space  $G_i$ 's *and* global convex hull?
  - How to store sizes  $|G_i|$  and  $|CH(G_i)|$ ?

### Extra run time cost of in-place solution:

- Binary search on  $G_i$  (instead of on  $CH(G_i)$ ).  $\mathcal{O}(1)$  per search step.
- Recompute two  $CH(G_i)$  for every output vertex  $p_k$ .  $\mathcal{O}(H^2 \log_2 H)$
- $\Rightarrow$  Global costs still in  $\mathcal{O}(n \cdot \log_2 h)$  for  $h < \sqrt{n / \log_2 n}$ .
- For larger  $h$ : Just run Graham's Scan.
- $\Rightarrow \mathcal{O}(n \cdot \log_2 h)$  in-place convex hull computation.

# Output-sensitive skyline computation

---

## Idea applicable also to skyline problem?

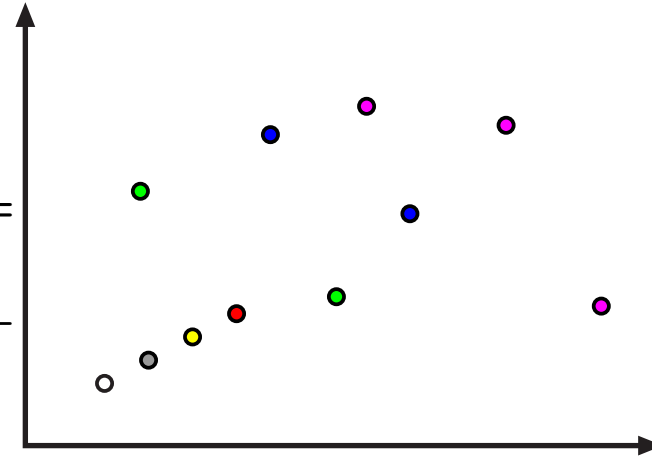
- Applicable:
  - Graham-like Scan
  - Jarvis-like March
  - Binary search on skylines
  - “Recognizing” the end of skylines
- $\Rightarrow \mathcal{O}(n \cdot \log_2 h)$  in-place skyline computation.



# Computing Layers of Maxima

## Def. of 'Layers of Maxima':

- Compute 'skyline'  $\text{MAX}(\mathcal{P})$  of  $\mathcal{P}$ .
- If  $\mathcal{P} \setminus \text{MAX}(\mathcal{P})$  not empty, set  $\mathcal{P} := \text{MAX}(\mathcal{P})$  and repeat.
- Number of iterations (layers) can be linear in  $n$ .



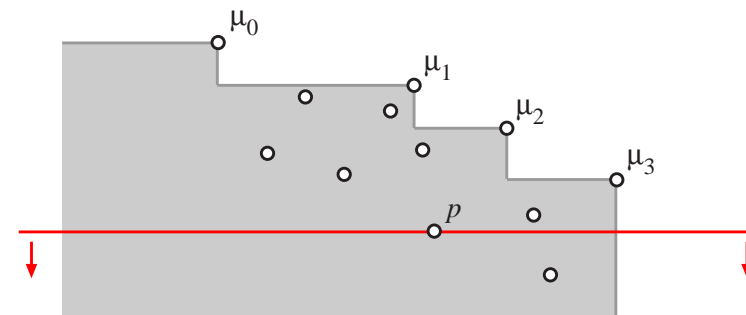
## To achieve (in in-place setting):

- Group points by layer.
- In each layer, sort by  $x$ .

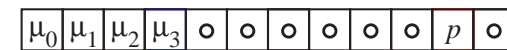


## Caveat:

- Naïve approach: Iteratively compute skylines.
- Cost:  $\mathcal{O}(n \log n)$  time per layer, i.e.,  $\mathcal{O}(n^2 \log n)$  time in total.



$\mu_i$  : maximal points



current

# Computing Layers of Maxima in-place: Overview

---

## Agenda:

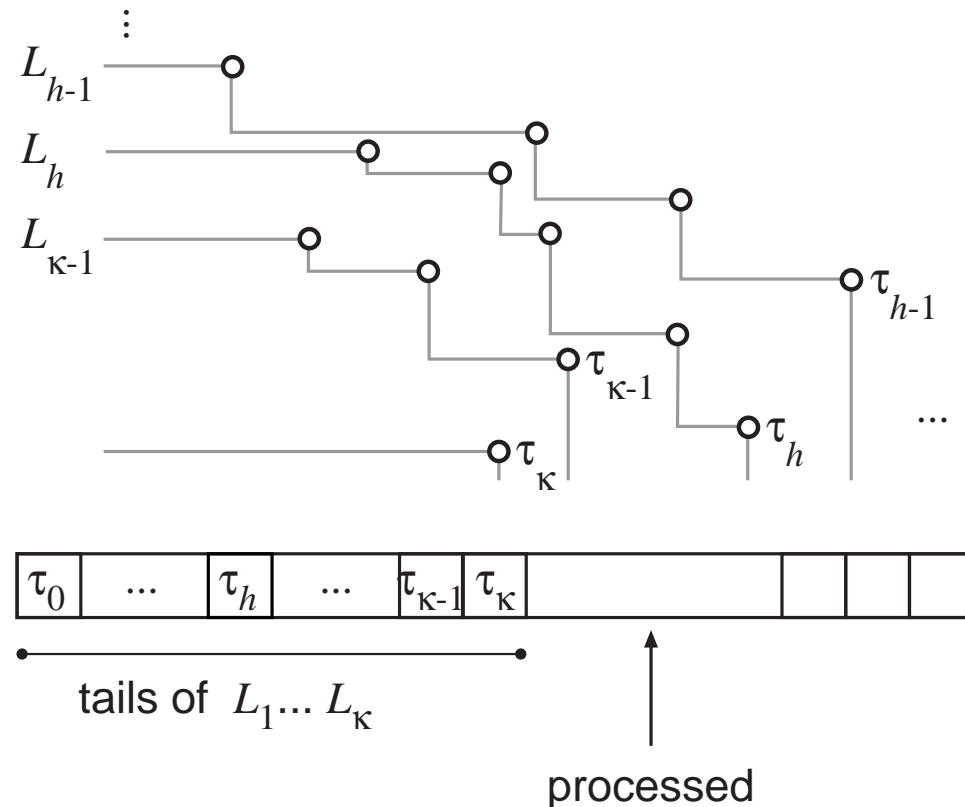
- Compute (and arrange!) all points on each of the  $\mathcal{O}(n)$  layers in-place.

## Approach:

- Process **multiple** layers at a time.
- Number of batches:  $\mathcal{O}(\log n)$ .
- Per batch:
  - Process  $\mathcal{O}(n/\log n)$  layers at a time by ...
  - sweeping the input array (similar to skyline sweep).
  - Maximum allowed cost:  $\mathcal{O}(n)$  time **per batch**, i.e., **no pre-sorting**.
- Charging scheme: Spend extra  $\mathcal{O}(\log n)$  time per point processed.

## For starters: Counting the Number of Layers

- Sweep points top-down, maintain tail  $\tau_i$  for each layer  $L_i$  in (inverse)  $<_x$ -order.



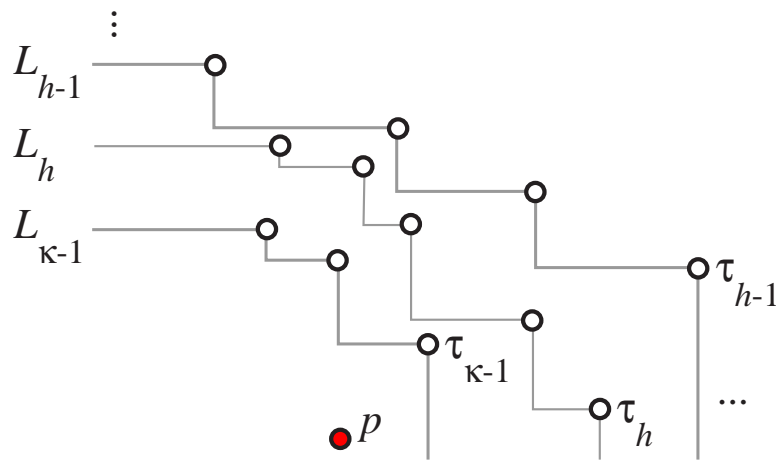
- In-place,  $\mathcal{O}(\log n)$  time per point  $\Rightarrow \mathcal{O}(n \log n)$  overall.
- Use this algorithm to count points on topmost layers.  $\Rightarrow$

## Counting the Points on the Topmost $\kappa$ Layers

- Fix  $\kappa \in \omega(1)$  and run essentially the same algorithm.
- Increment a global counter per “tail”-update.  $\mathcal{O}(\log n)$ /point

### A closer look:

- Query: Is some point  $p$  **not** on topmost  $\kappa$  layers?  $\mathcal{O}(1)$ /point



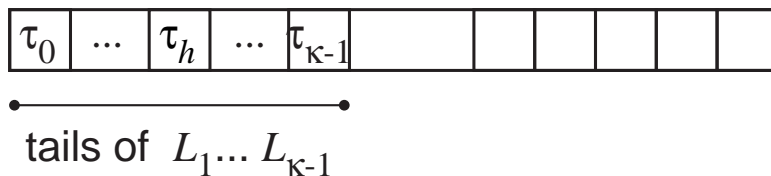
- Sum:  $\mathcal{O}(n + \xi \log n)$ ,  $\xi = \sum_{i=0}^{\kappa-1} |L_i|$ .

### To compute each $|L_i| \dots$

- $\dots$  we need to increment a **counter**  $c_i$  for each update of  $\tau_i$ .

### Wait a minute!

- Did you say “ $\kappa \in \omega(1)$ ”?
- Where/how to store  $\kappa$  counters?



## Finding “extra” space

---

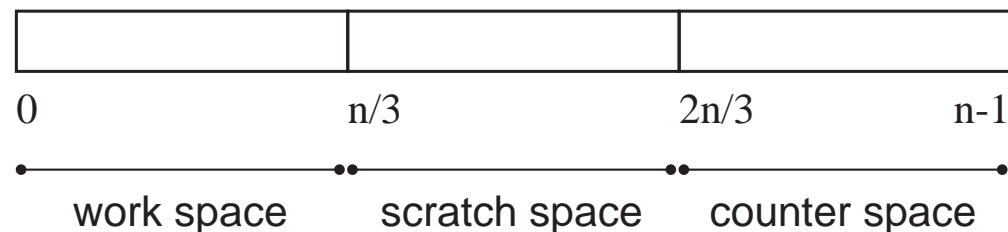
**Space needed:**  $\kappa \in \omega(1)$  counters, i.e.,  $\kappa \cdot \lceil \log_2 n \rceil$  bits.

### Bit-encoding technique [Munro, 1986]:

- Use permutation of two adjacent elements to encode one bit.
- $p <_y q$ :  $pq \equiv 0$ ,  $qp \equiv 1$ . Counter:  $2 \lceil \log_2 n \rceil$  elements.
- Set  $\kappa = \frac{1}{6}n / \log_2 n \Rightarrow \kappa$  counters need  $\frac{1}{3}n$  representing points.

### Partitioning the input array:

- Start working on the first  $\frac{1}{3}n$  entries.
- Use last  $\frac{1}{3}n$  entries for counters.



## Extracting the Topmost $\kappa$ Layers

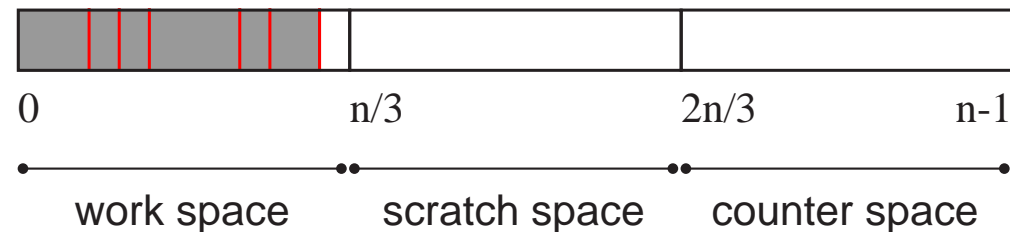
---

### Setup:

- Compute size  $c_i$  of  $i$ -th layer,  $0 \leq i < \kappa = \frac{1}{6}n / \log_2 n$ .
- Compute maximal  $j$  s.t.  $\sum_{i=0}^j c_i \leq \frac{1}{3}n$ .

### Extracting the topmost $j$ layers:

- Combine extraction with *counting sort*.
- Maintain “tails” in “work space”; construct layers *in sorted order* in “scratch space”.
- Move constructed layers to front.

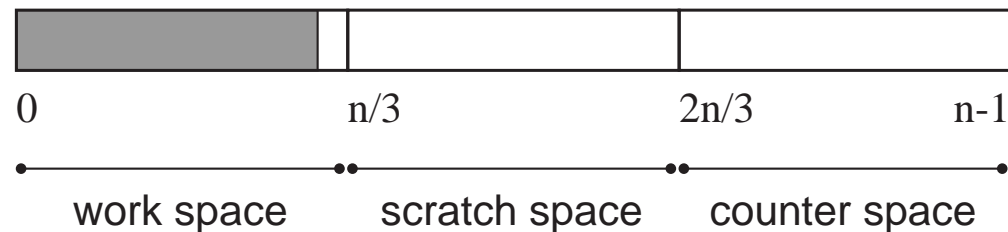


## Extracting All Layers—I

---

**First phase, i.e., for earlier iterations:**

- Construct (groups of  $\frac{1}{6} \cdot \frac{n}{\log n}$ ) layers in first part of the array.



**Analysis:**

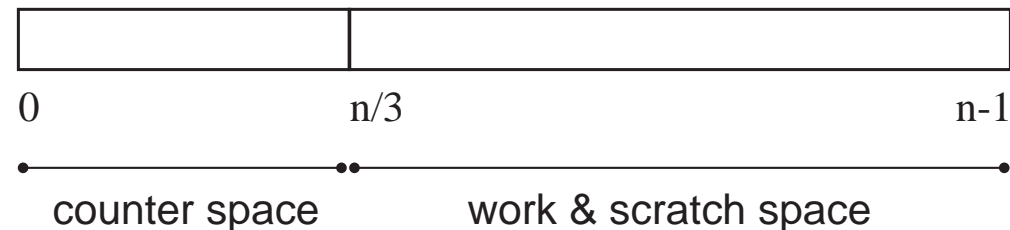
- Cost per iteration that processes all  $\xi$  points on  $\frac{1}{6} \cdot \frac{n}{\log n}$  layers:  $\mathcal{O}(n + \xi \log n)$ .
- Invariant: Keep unprocessed(!) points in sorted  $<_y$ -order.
- No more than  $\mathcal{O}(\log n)$  such iterations, i.e.,  $\mathcal{O}(n \log n)$  global cost.

## Extracting All Layers–II

---

### Second phase (with a grain of salt):

- Construct (in  $\mathcal{O}(n \log n)$  time) one layer crossing the boundary between work and scratch space using the [skyline](#) algorithm.



### Third phase i.e., for later iterations:

- Construct (groups of  $\frac{1}{6} \cdot \frac{n}{\log n}$ ) layers in second part of the array.
- $\mathcal{O}(n + \xi \log n)$  time to process  $\frac{1}{6} \cdot \frac{n}{\log n}$  layers with  $\xi$  points.
- Whenever scratch space is too small, perform [skyline](#) computations on subarrays of [geometrically decreasing size](#)  $\Rightarrow \mathcal{O}(n \log n)$ .

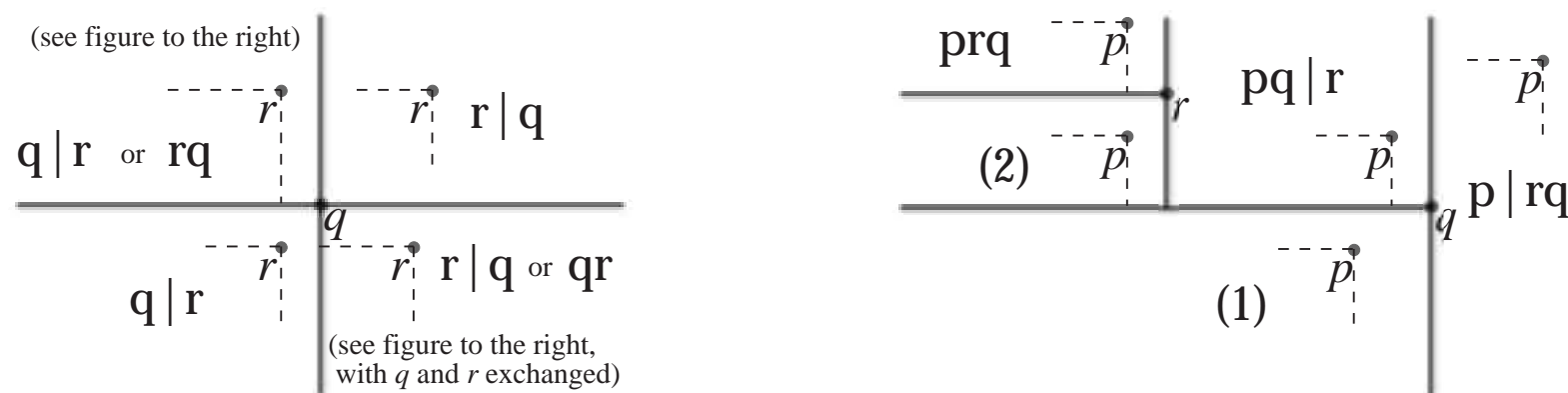


## Repairing the Layer Order

---

### Finishing up:

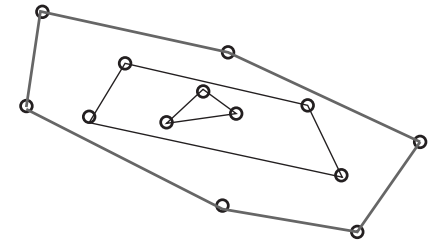
- Bit-encoding corrupts layer order (locally).
- Repairing after last iteration by linear time sweep:
- Each bit-neighbour pair  $(q, r)$  can be correctly ordered by only looking at  $q$ ,  $r$  and predecessor  $p$ :



- $\Rightarrow$  Layer order repairable by linear scan.
  - $\Rightarrow \mathcal{O}(n \cdot \log_2 n)$  in-place computation of layers of maxima.
-

# Convex layers computation

---

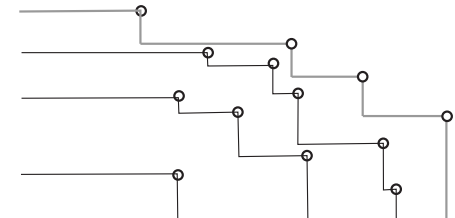


## Ideas applicable also to convex layers problem?

- Applicable:
  - Sweep-framework.
  - Maintenance of tails for each layer.

## Problem:

- A point changes its convex layer  $\mathcal{O}(n)$  times during sweep.



## Solutions (for 2D layers and 3D hull)?

- for 2D convex layers?
- for 3D convex hull?

## Computing the skyline in 3D

---

### Time-Optimal algorithm [Kung et al., 1975]:

- Do divide-and-conquer along  $z$ .
- For each conquer-step:
  - Problem broken down to 2D ...
  - Divide in upper and lower part.
  - Simultaneous  $y$ -sweep over upper and lower maxima and exploit:
  - For each maximum of  $\mathcal{P}_{|z>\zeta}$ : on skyline (of whole input).
  - For each maximum of  $\mathcal{P}_{|z\leq\zeta}$ : on skyline  $\Rightarrow x$ -larger than actual tail.

### Making it in-place:

- Use in-place recursion framework [Bose et al., 2006].
- For each conquering: Explicit reconstruction of skyline bounds.

...	Maxima of $\mathcal{P}_{ z\leq\zeta}$		Maxima of $\mathcal{P}_{ z>\zeta}$		...
	$l_b$	$l'_b$	$\lfloor (l_b + l_e)/2 \rfloor$	$l'_e$	$l_e$

⇓

...	Maxima of $\mathcal{P}_{ z\leq\zeta} \cup \mathcal{P}_{ z>\zeta}$		...
	$l_b$	$l'$	$l_e$

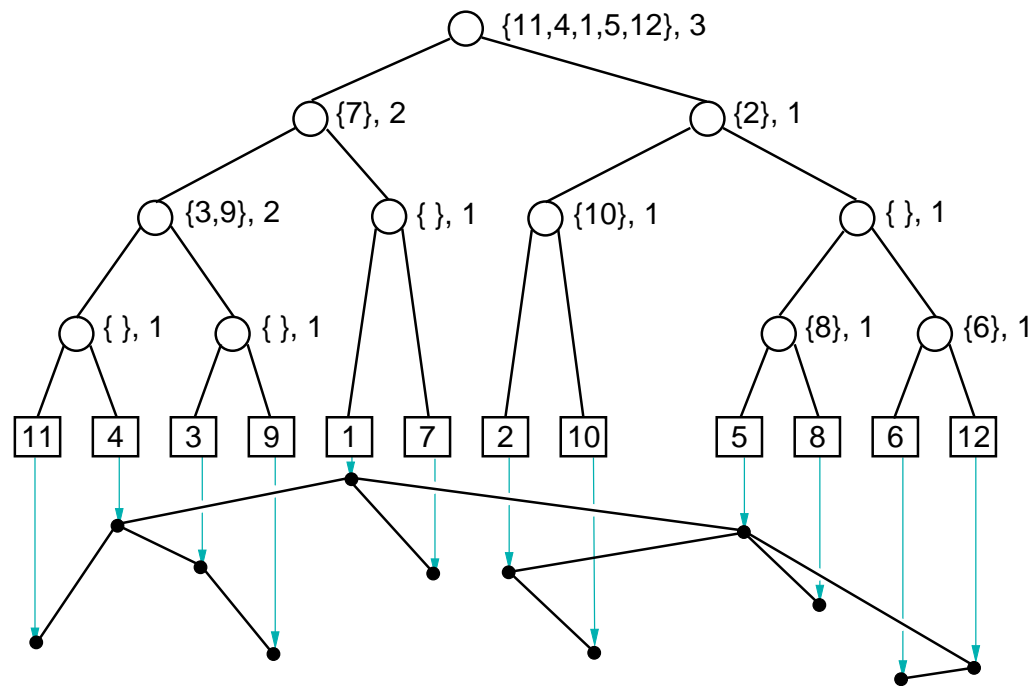
---

# Convex hull and layers computation

---

## Solutions (for 2D layers and 3D hull):

- Build recursively defined hull data structure ...
- ... then remove points iteratively.
- $\Rightarrow$  2D Convex layers in  $\mathcal{O}(n \log_2 n)$  time [Chazelle, 1985].
- $\Rightarrow$  3D Convex hull in  $\mathcal{O}(n \log_2^3 n)$  time and  $\mathcal{O}(1)$  space [Brönnimann et al., 2004a].



# Overview

---

1. Introduction: Motivation for implicit computation
2. Skylines and convex hulls

## Bibliography

- [Blunck & Vahrenhold, 2006]** H. Blunck and J. Vahrenhold. In-place algorithms for computing (layers of) maxima. In: *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT '06)*, volume 4059 of *Lecture Notes of Computer Science*, pages 363–374. Springer, Berlin, 2006.
- [Bose et al., 2006]** P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geometry: Theory & Applications*, 2006.
- [Brönnimann & M.Chan, 2004]** H. Brönnimann and T. M.Chan. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time. In: *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN '04)*, volume 2976 of *Lecture Notes in Computer Science*, pages 162–171, Berlin, 2004. Springer.
- [Brönnimann et al., 2004a]** H. Brönnimann, T. M. Chan, and E. Y. Chen. Towards in-place geometric algorithms and data structures. In: *Proceedings of the 20th Annual ACM Symposium on Computational Geometry (SoCG '04)*, pages 239–246, New York, NY, USA, 2004. ACM Press.
- [Brönnimann et al., 2004b]** H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint. Space-efficient planar convex hull algorithms. *Theoretical Computer Science*, 321(1):25–40, June 2004.

- [Brönnimann et al., 2004c]** H. Brönnimann, T. M.Chan, and E. Y. Chen. Towards in-place geometric algorithms. In: *Proceedings of the 20th Annual ACM Symposium on Computational Geometry (SoCG '04)*, pages 239–246. ACM Press, 2004.
- [Carlsson & Sundström, 1995]** S. Carlsson and M. Sundström. Linear-time in-place selection in less than  $3n$  comparisons. In: *Proceedings of the 6th Annual International Symposium on Algorithms and Computation (ISAAC'02)*, volume 3827 of *Lecture Notes in Computer Science*, pages 244–253, Berlin, 1995. Springer.
- [Chan, 1996]** T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.
- [Chazelle, 1985]** B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31:509–517, 1985.
- [Chen & M.Chan, 2005]** E. Y. Chen and T. M.Chan. Space-efficient algorithms for klee's measure problem. In: *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG '05)*, 2005.
- [Floyd, 1964]** R. W. Floyd. Algorithm 245: Treesort. *Communications of the ACM*, 7(12):701, December 1964.
- [Franceschini & Grossi, 2003]** G. Franceschini and R. Grossi. Optimal worst-case operations for implicit cache-oblivious search trees. In: *Proceedings of*

*the 8th International Workshop on Algorithms and Data Structures (WADS '03)*, pages 114–126, 2003.

- [Geffert & Kollar, 2001]** V. Geffert and J. Kollar. Linear-time in-place selection in  $\epsilon \cdot n$  element moves. Technical report, P. J. Safarik University, 2001.
- [Geffert et al., 2000]** V. Geffert, J. Katajainen, and T. Pasanen. Asymptotically efficient in-place merging. *Theoretical Computer Science*, 237(1–2):159–181, April 2000.
- [Graham, 1972]** R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.
- [Katajainen & Pasanen, 1999]** J. Katajainen and T. A. Pasanen. In-place sorting with fewer moves. *Information Processing Letters*, 70(1):31–37, 1999.
- [Kung et al., 1975]** H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975.
- [Mannila & Ukkonen, 1984]** H. Mannila and E. Ukkonen. A simple linear-time algorithm for in situ merging. *Information Processing Letters*, 18(4):203–208, 1984.
- [Munro, 1986]** J. I. Munro. An implicit data structure supporting insertion, deletion, and search in  $o(\log n)$  time. *Journal of Computer and System Sciences*, 33(1):66–74, 1986.



**[Vahrenhold, 2005]** J. Vahrenhold. Line-segment intersection made in-place. In: *Proceedings of the 9th International Workshop on Algorithms and Data Structures (WADS '05)*, volume 3608 of *Lecture Notes in Computer Science*, pages 146–157, Berlin, 2005. Springer.