Geometric Algorithms with Limited Resources

Lecturers: Themistoklis Gouleakis (Themis)
tgouleak@mpi-inf.mpg.de
Sándor Kisfaludi-Bak (['ʃaːndor])
skisfalu@mpi-inf.mpg.de

TA: Hannaneh Akrami (Hana)
hannaneh.akrami95@gmail.com

Assignments: biweekly, hand in 50% of total point value to take exam.
Exam: oral, soon after end of teaching.

Lectures are recorded (without video) and uploaded, see mailing list for access.

Geometric Algorithms with Limited Resources

Typical input: set of points or a metric space.
But! Not proper computatinal geometry
course, only what we need.

Geometric Algorithms with Limited Resources

Typical input: set of points or a metric space.
But! Not proper computatinal geometry
course, only what we need.

Geometric Algorithms with Limited Resources

Sublinear time, property testing.
Sublinear space, streaming.

Typical input: set of points or a metric space.
But! Not proper computatinal geometry course, only what we need.

Geometric Algorithms with Limited Resources

Sublinear time, property testing.
Sublinear space, streaming.

Recent (mostly after 2000) results, fresh research questions!

Introduction, concepts from computational geometry

Sándor Kisfaludi-Bak

Geometric algorithms with limited resources
Summer semester 2021

max planck institut
informatik

Overview

Overview

- Computational models, limitations in space

Overview

- Computational models, limitations in space

- Naive convex hull, gift wrapping in O(1) space

Overview

- Computational models, limitations in space
- Naive convex hull, gift wrapping in O(1) space
- Classic algorithm: Graham's scan, $O(n \log n)$ time

Overview

- Computational models, limitations in space
- Naive convex hull, gift wrapping in O(1) space
- Classic algorithm: Graham's scan, $O(n \log n)$ time
- Time-space tradeoff: Chan and Chen's algorithm

} Convex hull

Overview

- Computational models, limitations in space
- Naive convex hull, gift wrapping in O(1) space
- Classic algorithm: Graham's scan, $O(n \log n)$ time
- Time-space tradeoff: Chan and Chen's algorithm
- A classic deterministic algorithm in $\mathbb{R}^2$

Convex hull

Low-dim linear programming

Overview

- Computational models, limitations in space
- Naive convex hull, gift wrapping in O(1) space
- Classic algorithm: Graham's scan, $O(n \log n)$ time
- Time-space tradeoff: Chan and Chen's algorithm

} Convex hull

- A classic deterministic algorithm in $\mathbb{R}^2$
- Sublinear space LP (Chan–Chen '07)

} Low-dim linear programming

Real RAM vs. Word RAM

Real RAM vs. Word RAM

Real RAM	Word RAM
arbitrary real numbers	words of size $\Theta(\log n)$

Real RAM vs. Word RAM

Real RAM	Word RAM
arbitrary real numbers	words of size $\Theta(\log n)$
no rounding/floor, no modulo	realistic*operations (shifts, etc)

Real RAM vs. Word RAM

Real RAM	Word RAM
arbitrary real numbers	words of size $\Theta(\log n)$
no rounding/floor, no modulo	realistic* operations (shifts, etc)
Real inputs and outputs, can extend with $\sqrt{.}, \ln(.)$	Exact arithmetic for rational inputs with $+ - * /$

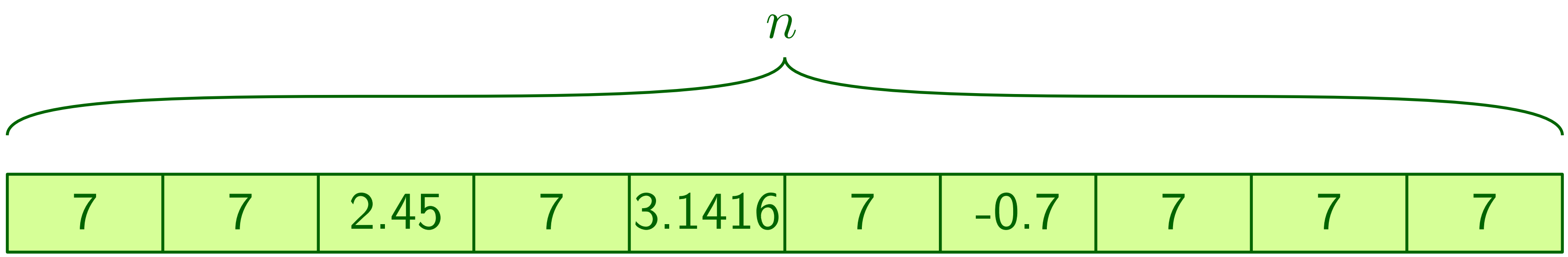
Real RAM vs. Word RAM

Real RAM	Word RAM
arbitrary real numbers	words of size $\Theta(\log n)$
no rounding/floor, no modulo	realistic* operations (shifts, etc)
Real inputs and outputs, can extend with $\sqrt{.}, \ln(.)$	Exact arithmetic for rational inputs with $+ - * /$
Unrealistic power	Too restrictive?

Real RAM vs. Word RAM

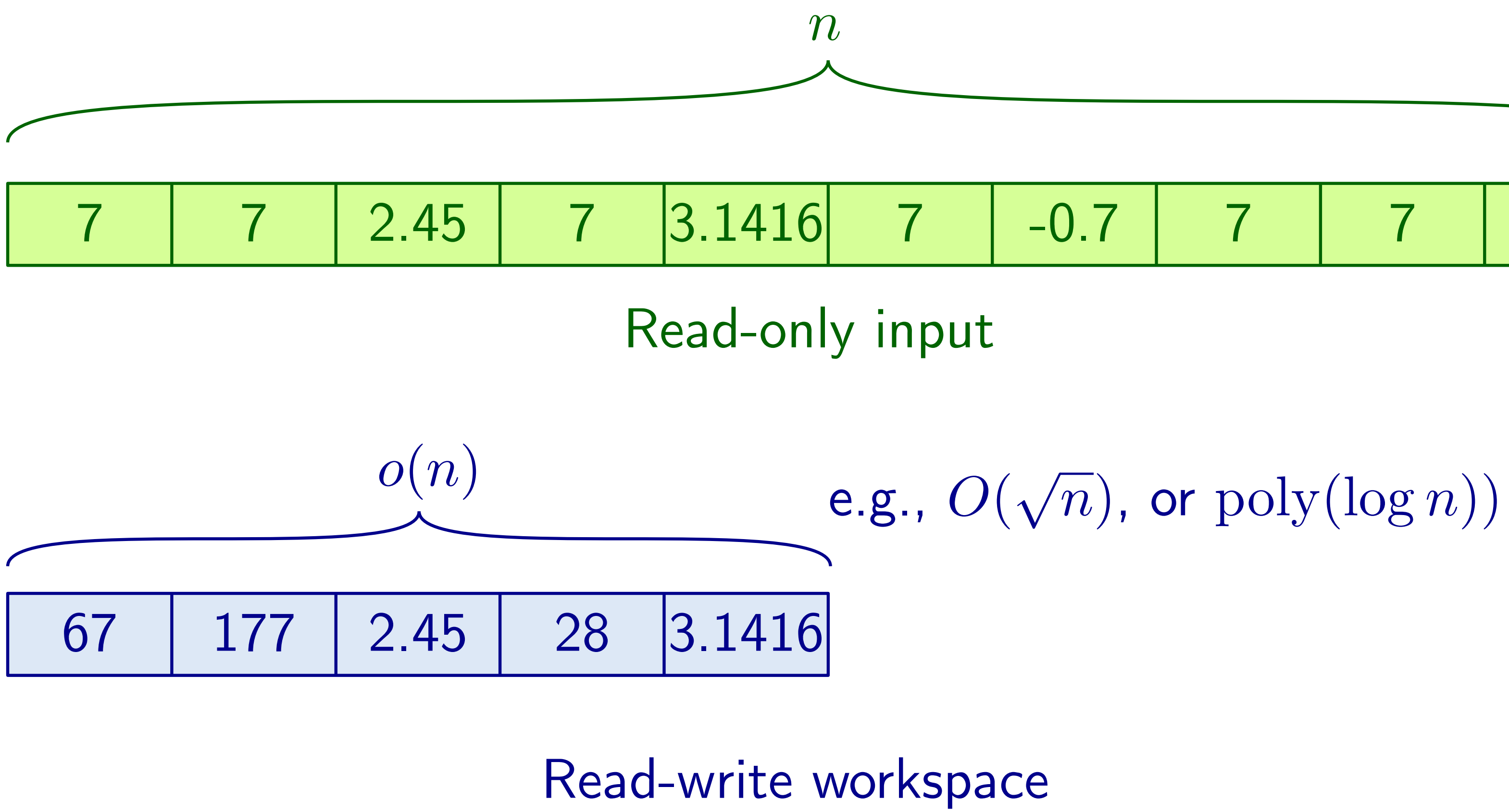
Real RAM	Word RAM
arbitrary real numbers	words of size $\Theta(\log n)$
no rounding/floor, no modulo	realistic* operations (shifts, etc)
Real inputs and outputs, can extend with $\sqrt{.}, \ln(.)$	Exact arithmetic for rational inputs with $+ - * /$
Unrealistic power	Too restrictive?
Often needed for exact computation	Usually enough for approximations!

Limited workspace model

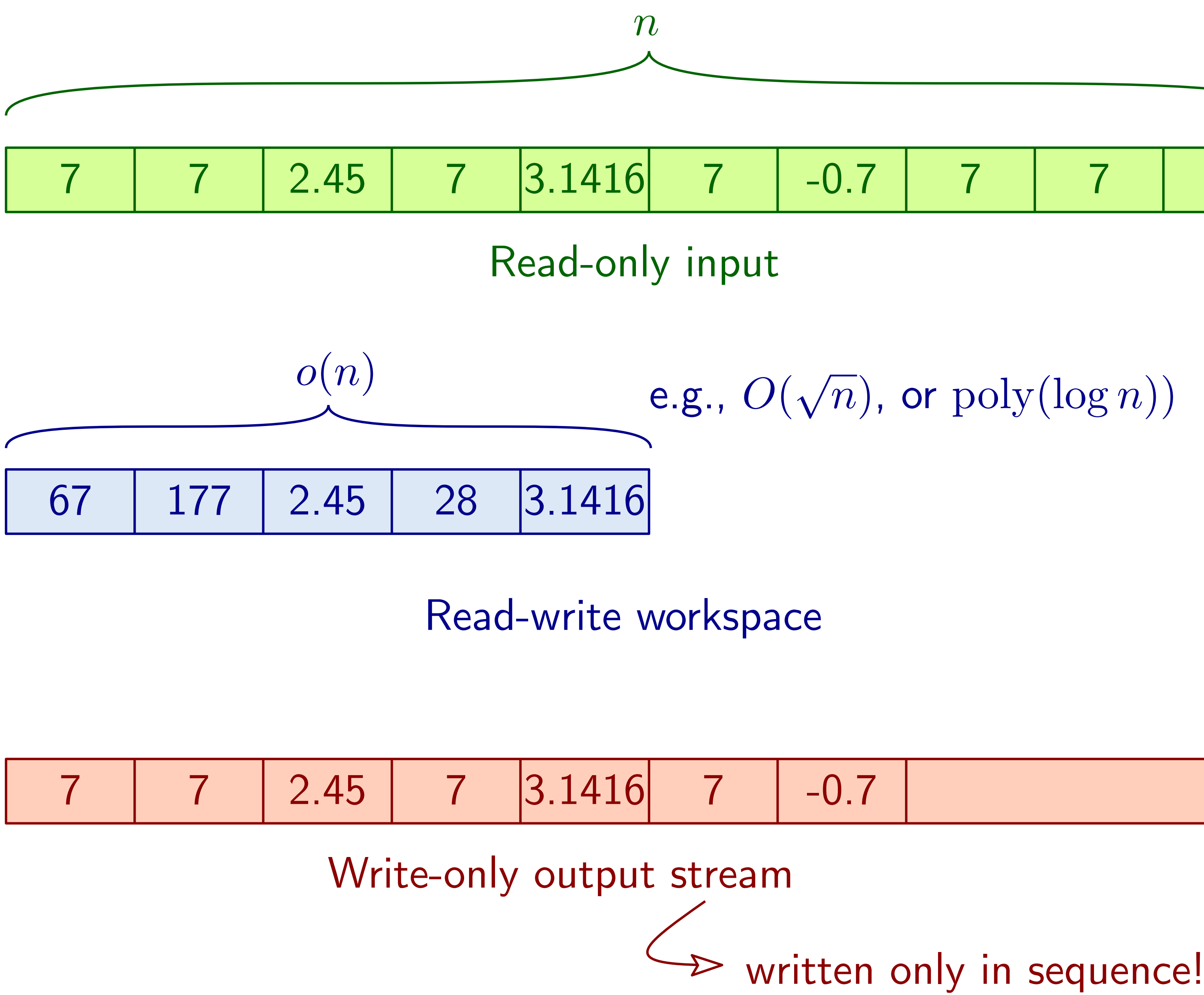


Read-only input

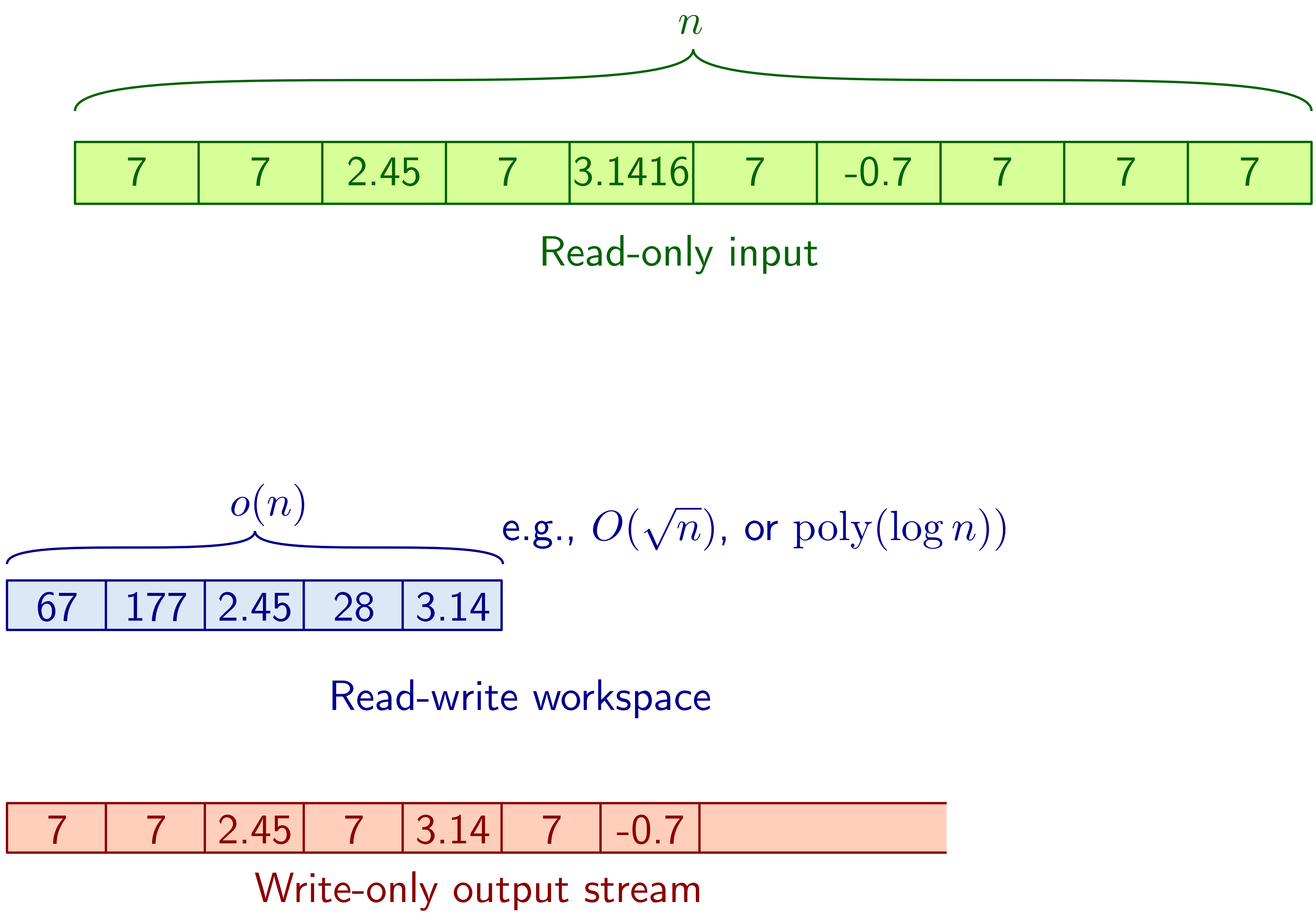
Limited workspace model



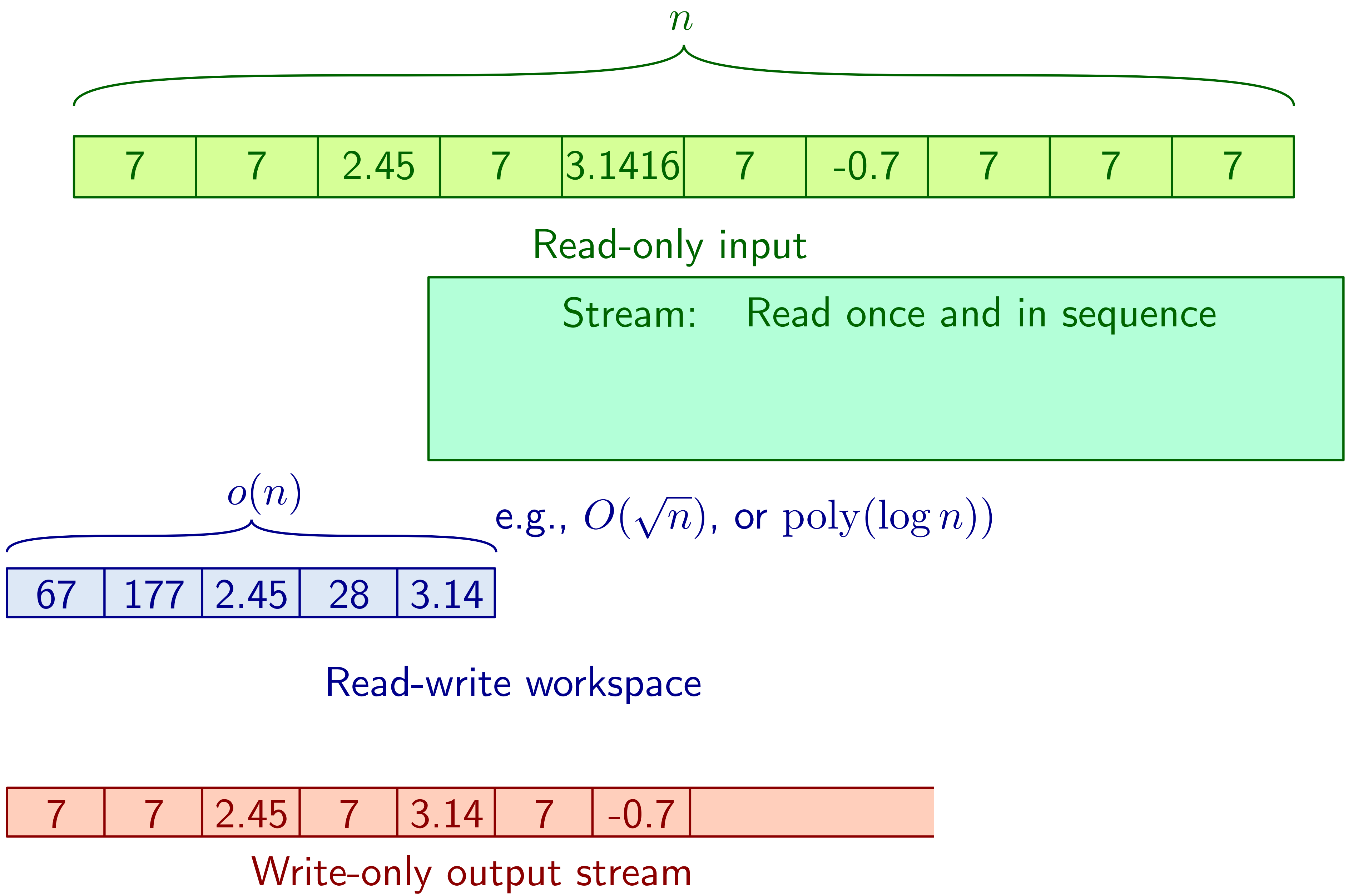
Limited workspace model



Streaming and multi-pass model



Streaming and multi-pass model



Streaming and multi-pass model

n

7	7	2.45	7	3.1416	7	-0.7	7	7	7

Read-only input

Stream:	Read once and in sequence
multi-pass, k-pass:	Read k times and in sequence

$o(n)$ e.g., $O(\sqrt{n})$, or $\mathrm{poly}(\log n))$

67	177	2.45	28	3.14

Read-write workspace

7	7	2.45	7	3.14	7	-0.7	

Write-only output stream

Convex hull

Notations, definitions

$\mathbb{R}^d$ is d-dimensional Euclidean space

$P = \{p_1, \ldots, p_n\}$ set of n points

$X \subseteq \mathbb{R}^d$ is *convex* if for any $p, q \in X$ we have $pq \subseteq X$

Convex hull

Notations, definitions

$\mathbb{R}^d$ is d-dimensional Euclidean space

$P = \{p_1, \ldots, p_n\}$ set of n points

$X \subseteq \mathbb{R}^d$ is *convex* if for any $p, q \in X$ we have $pq \subseteq X$

Convex hull:

$$\mathrm{conv}(P) = \begin{cases} \text{minimum convex set containing } P \\ \text{intersection of convex sets containing } P \\ \{\alpha_1 p_1 + \cdots + \alpha_n p_n \mid \alpha_i \geq 0 \text{ and } \sum_{i=1}^n \alpha_i = 1\} \end{cases}$$

Convex hull

Notations, definitions

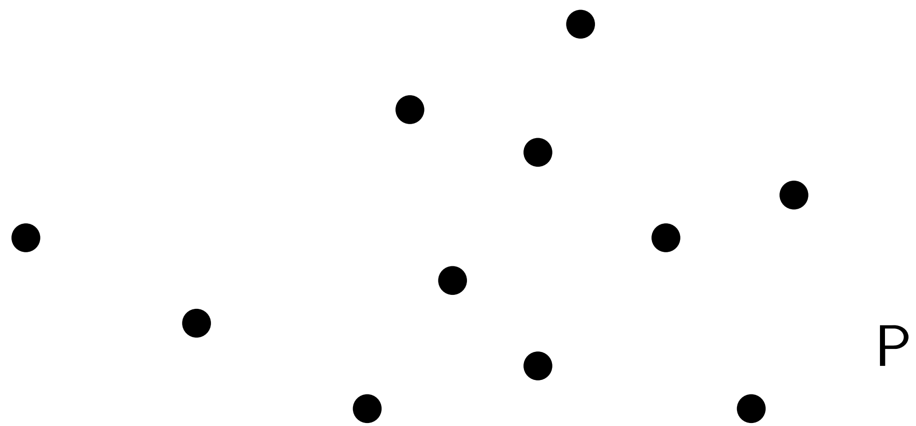
$\mathbb{R}^d$ is d-dimensional Euclidean space

$P = \{p_1, \ldots, p_n\}$ set of n points

$X \subseteq \mathbb{R}^d$ is *convex* if for any $p, q \in X$ we have $pq \subseteq X$

Convex hull:

$$\mathrm{conv}(P) = \begin{cases} \text{minimum convex set containing } P \\ \text{intersection of convex sets containing } P \\ \{\alpha_1 p_1 + \cdots + \alpha_n p_n \mid \alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i = 1\} \end{cases}$$



Convex hull

Notations, definitions

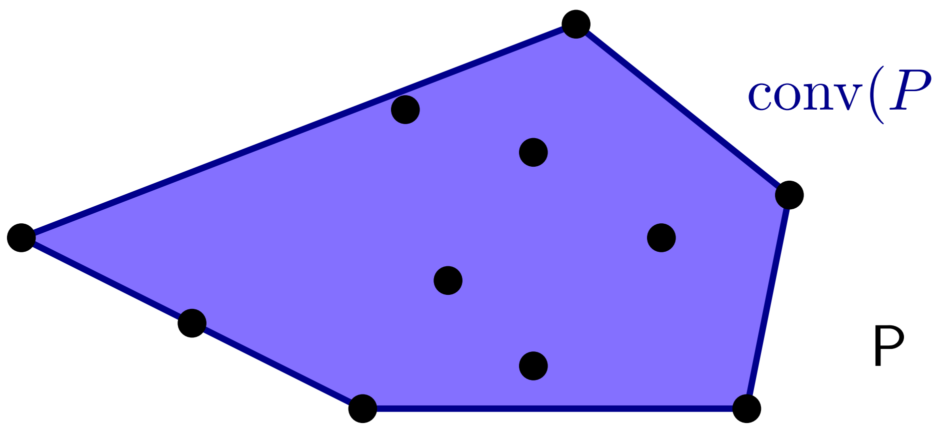
$\mathbb{R}^d$ is d-dimensional Euclidean space

$P = \{p_1, \ldots, p_n\}$ set of n points

$X \subseteq \mathbb{R}^d$ is *convex* if for any $p, q \in X$ we have $pq \subseteq X$

Convex hull:

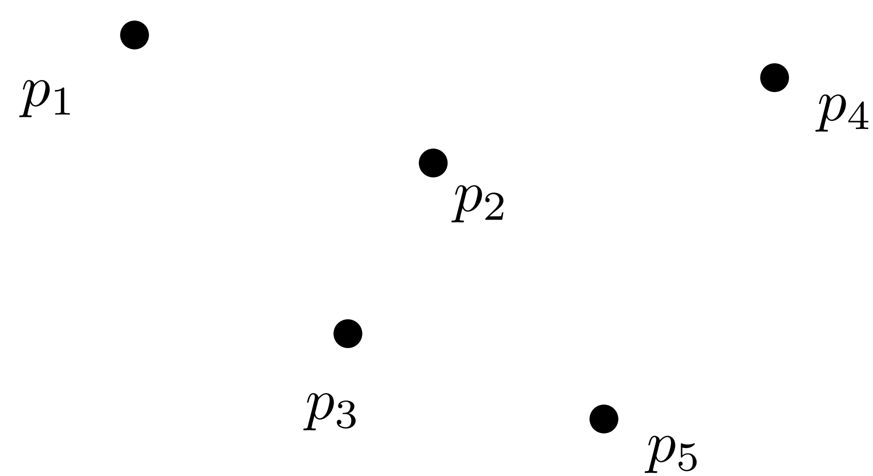
$$\mathrm{conv}(P) = \begin{cases} \text{minimum convex set containing } P \\ \text{intersection of convex sets containing } P \\ \{\alpha_1 p_1 + \cdots + \alpha_n p_n \mid \alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i = 1\} \end{cases}$$



Convex hull: input and output

Input: Points with coordianate pairs $(x, y) \in \mathbb{R}^2$

$$(e, \pi), (3, 3), (2.95, 2.9), (\sqrt{11}, 3.05), (\pi, e)$$



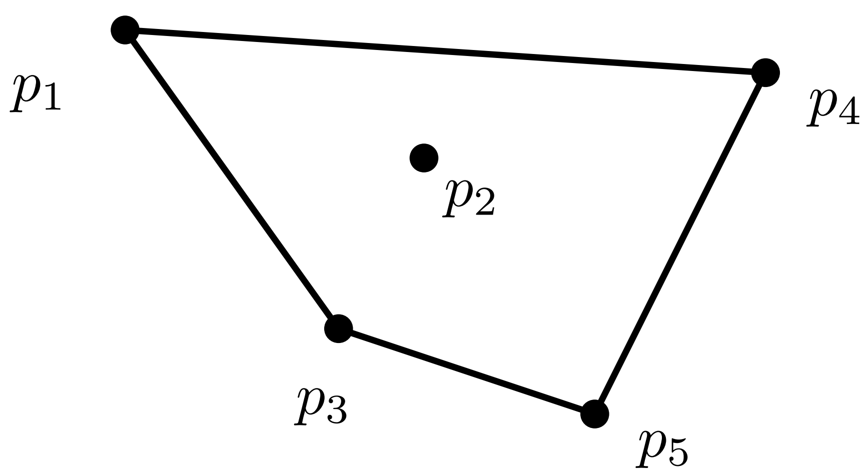
Convex hull: input and output

Input: Points with coordianate pairs $(x, y) \in \mathbb{R}^2$

$$(e, \pi), (3, 3), (2.95, 2.9), (\sqrt{11}, 3.05), (\pi, e)$$

Output: "corners" in clockwise order
smallest $Q \subseteq P$ s.t. $\mathrm{conv}(Q) = \mathrm{conv}(P)$

$$p_1, p_4, p_5, p_3$$



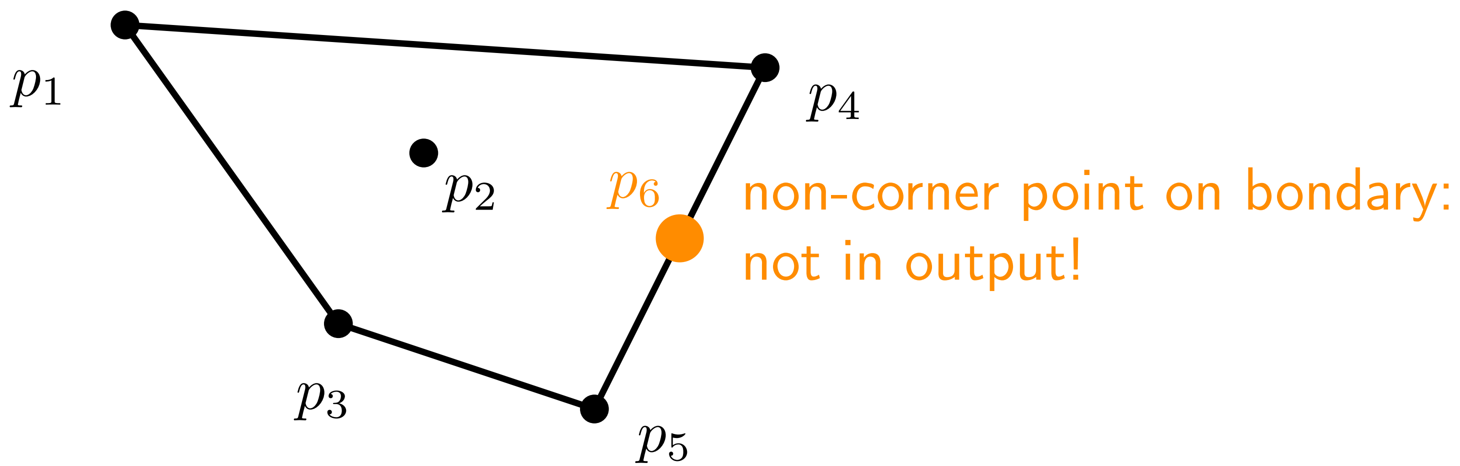
Convex hull: input and output

Input: Points with coordianate pairs $(x, y) \in \mathbb{R}^2$

$$(e, \pi), (3, 3), (2.95, 2.9), (\sqrt{11}, 3.05), (\pi, e)$$

Output: "corners" in clockwise order
smallest $Q \subseteq P$ s.t. $\mathrm{conv}(Q) = \mathrm{conv}(P)$

p_1, p_4, p_5, p_3



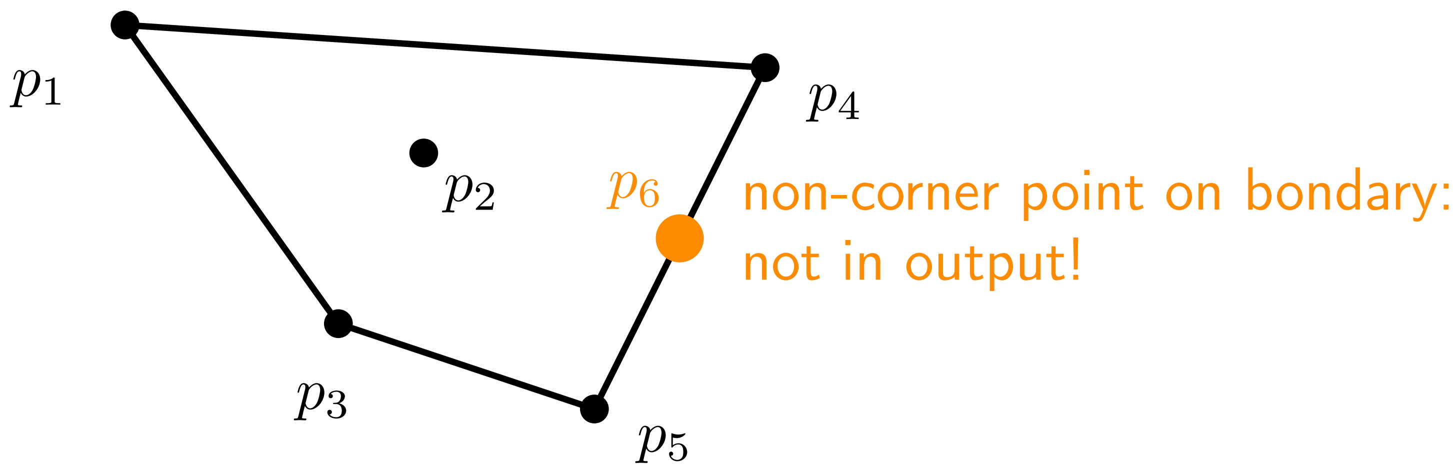
Convex hull: input and output

Input: Points with coordianate pairs $(x, y) \in \mathbb{R}^2$

$$(e, \pi), (3, 3), (2.95, 2.9), (\sqrt{11}, 3.05), (\pi, e)$$

Output: "corners" in clockwise order
smallest $Q \subseteq P$ s.t. $\mathrm{conv}(Q) = \mathrm{conv}(P)$

p_1, p_4, p_5, p_3



Everything works with rational inputs on Word RAM!

Naive algorithms, workspace O(1)

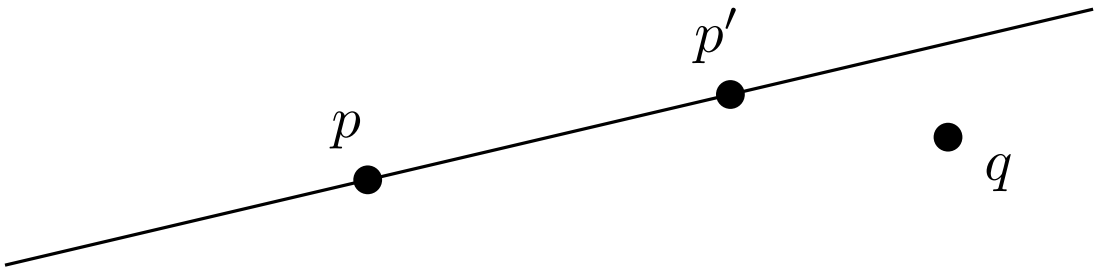
Naive Algorithm

Suppose no 3 points on one line. (no *collinear triples*)

Naive Algorithm

Suppose no 3 points on one line. (no *collinear triples*)

In $O(1)$ time, decide if q is on left or right side of line pp'

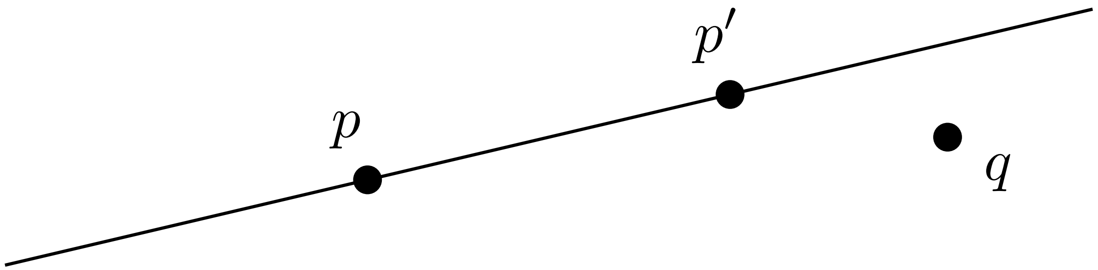


$$\operatorname{sgn}\left(\begin{vmatrix} p_x & p_y & 1 \\ p'_x & p'_y & 1 \\ q_x & q_y & 1 \end{vmatrix}\right)$$

Naive Algorithm

Suppose no 3 points on one line. (no *collinear triples*)

In $O(1)$ time, decide if q is on left or right side of line pp'



$$\operatorname{sgn}\left(\begin{vmatrix} p_x & p_y & 1 \\ p'_x & p'_y & 1 \\ q_x & q_y & 1 \end{vmatrix}\right)$$

Naive Convex Hull in $\mathbb{R}^2$

For each $p, p' \in P$,

check if all $q \in P \setminus \{p, p'\}$ is on the left of line pp'.

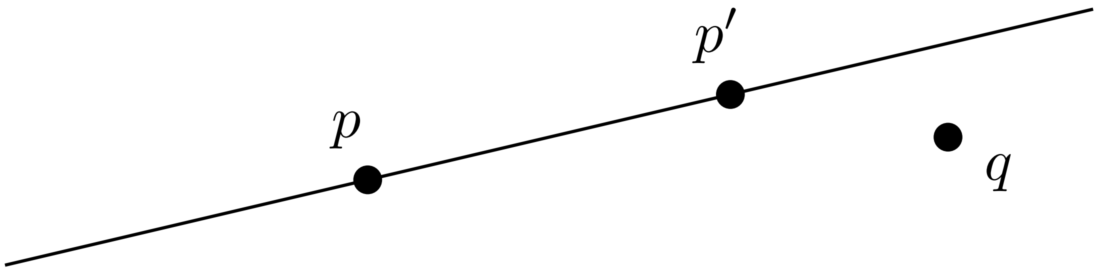
If yes, then p' follows p in $\operatorname{conv}(P)$.

Assemble and output the hull

Naive Algorithm

Suppose no 3 points on one line. (no *collinear triples*)

In $O(1)$ time, decide if q is on left or right side of line pp'



$$\operatorname{sgn}\left(\begin{vmatrix} p_x & p_y & 1 \\ p'_x & p'_y & 1 \\ q_x & q_y & 1 \end{vmatrix}\right)$$

Naive Convex Hull in $\mathbb{R}^2$

For each $p, p' \in P$,

check if all $q \in P \setminus \{p, p'\}$ is on the left of line pp'.

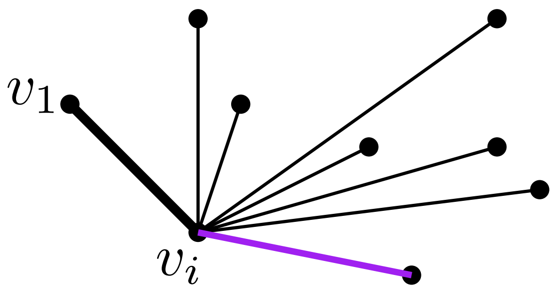
If yes, then p' follows p in $\operatorname{conv}(P)$.

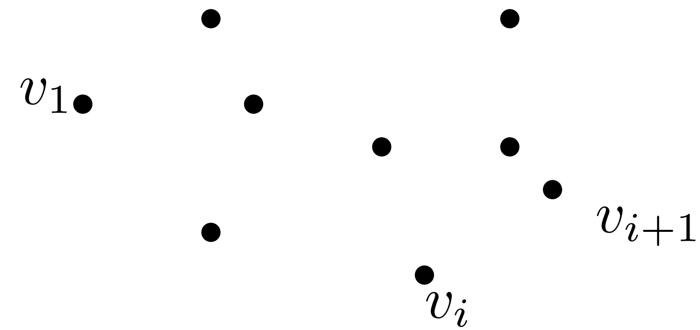
Assemble and output the hull

Running time: $\binom{n}{2} \cdot (n-2) \cdot O(1) = O(n^3)$

Less naive algorithm: Jarvis' March – aka gift wrapping

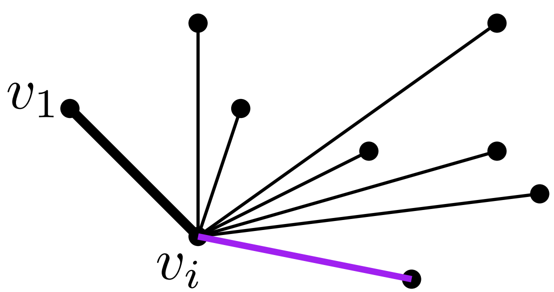
Algorithm:
Start at leftmost point, find next point with minimum (maxmum) slope.

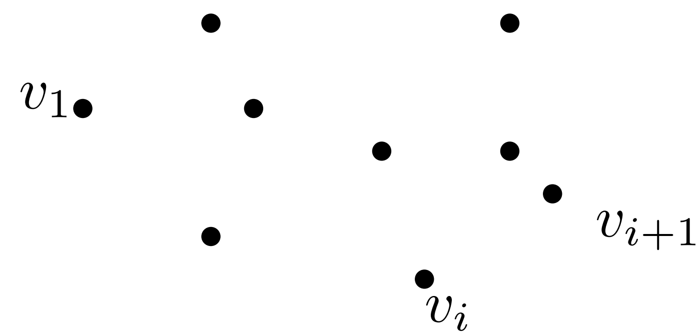




Less naive algorithm: Jarvis' March – aka gift wrapping

Algorithm:
Start at leftmost point, find next point with minimum (maxmum) slope.

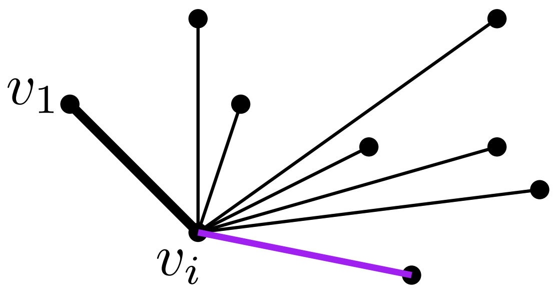


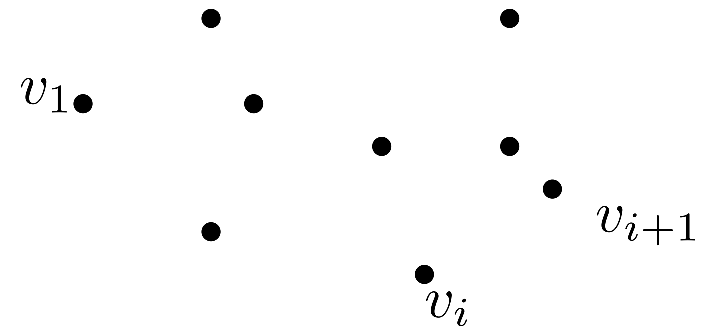


$O(hn)$ time, and enough to keep track of v_1, v_i, v_{i+1}. $O(1)$ space.

Less naive algorithm: Jarvis' March – aka gift wrapping

Algorithm:
Start at leftmost point, find next point with minimum (maxmum) slope.





$O(hn)$ time, and enough to keep track of v_1, v_i, v_{i+1}. $O(1)$ space.

$h =$ size of convex hull. Output-sensitive algorithm.

Graham's scan (1972)

Graham's Scan idea

Suppose points have distinct x-coordinates.

Let $p_1, \ldots, p_n$: points sorted with increasing x-coordinates.

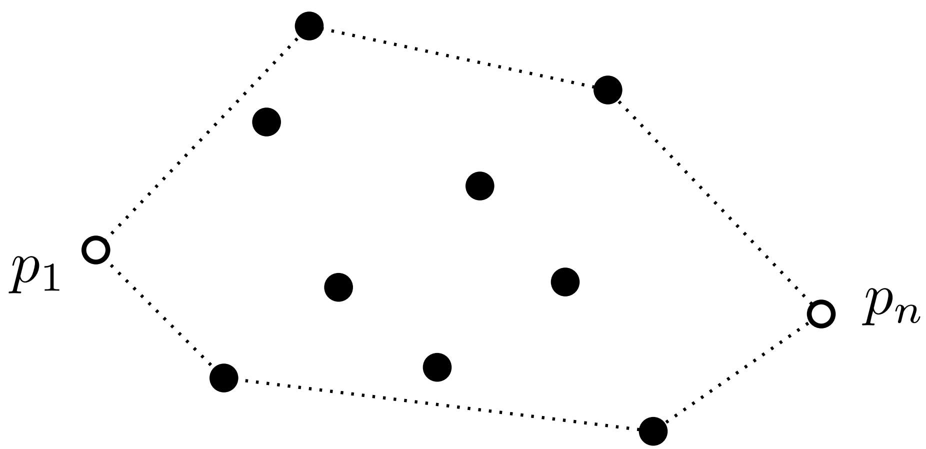
Graham's Scan idea

Suppose points have distinct x-coordinates.

Let $p_1, \ldots, p_n$: points sorted with increasing x-coordinates.

$\longrightarrow$ p_1, p_n are on convex hull

Upper hull
part of the hull after p_1 and before p_n in clockwise order



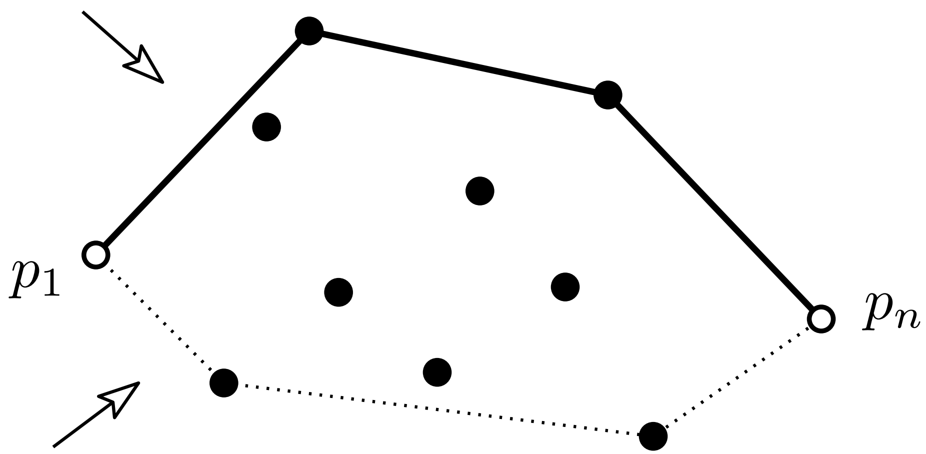
Graham's Scan idea

Suppose points have distinct x-coordinates.

Let $p_1, \ldots, p_n$: points sorted with increasing x-coordinates.

$\longrightarrow$ p_1, p_n are on convex hull

Upper hull
part of the hull after p_1 and before p_n in clockwise order



Lower hull

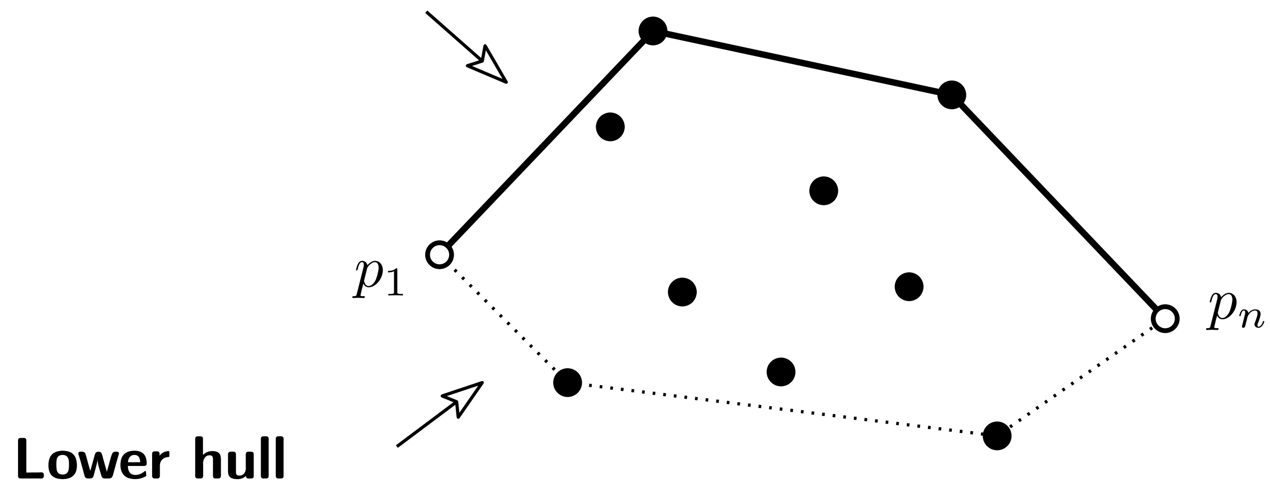
Graham's Scan idea

Suppose points have distinct x-coordinates.

Let $p_1, \ldots, p_n$: points sorted with increasing x-coordinates.

$\rightarrow$ p_1, p_n are on convex hull

Upper hull
part of the hull after p_1 and before p_n in clockwise order

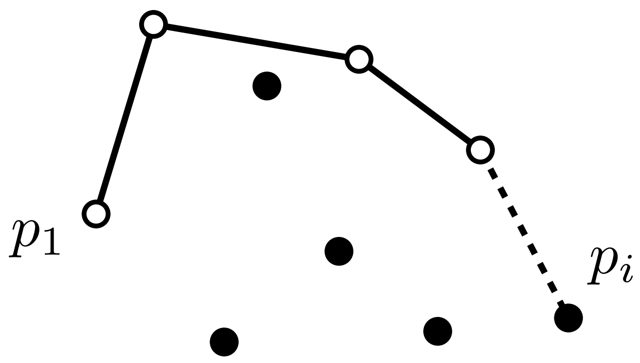


Idea:
Add points left to right, update upper hull after each addition

Graham's Scan: update

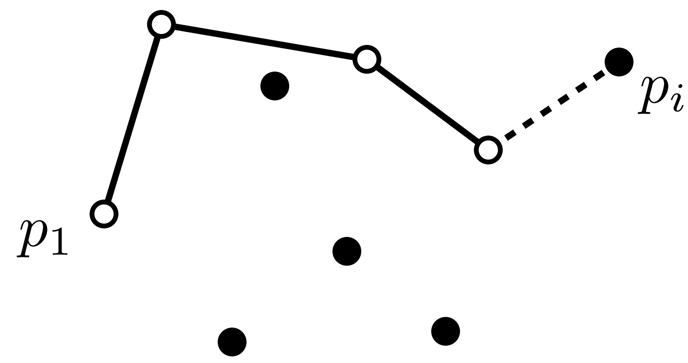
Right turn

(p_i is below last hull segment)



Left turn

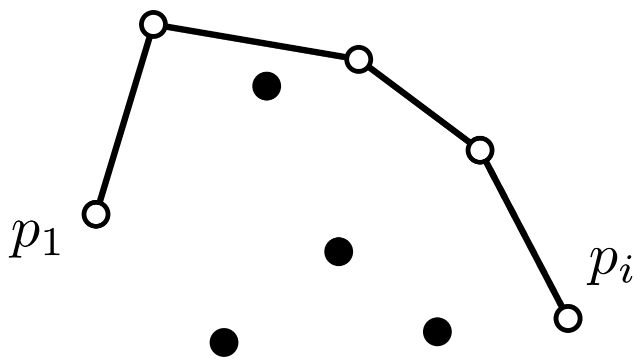
(p_i is above last hull segment)



Graham's Scan: update

Right turn

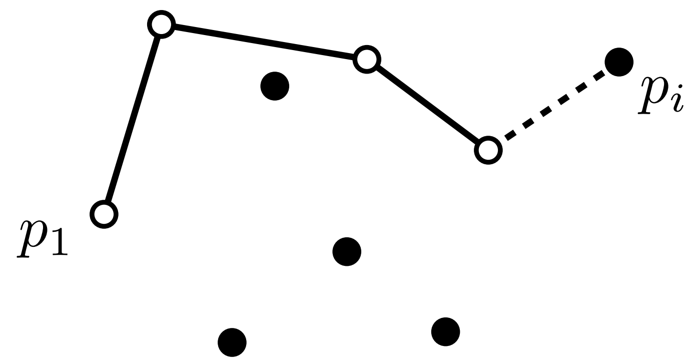
(p_i is below last hull segment)



Add p_i to the upper hull

Left turn

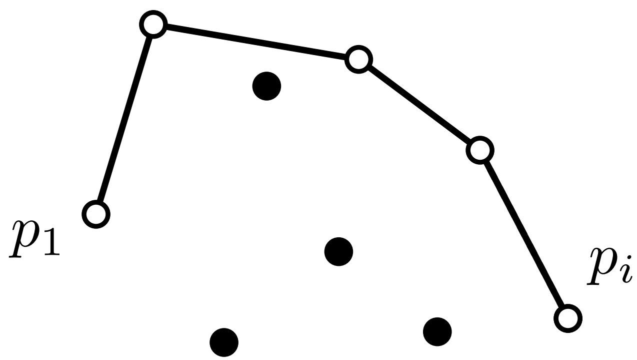
(p_i is above last hull segment)



Graham's Scan: update

Right turn

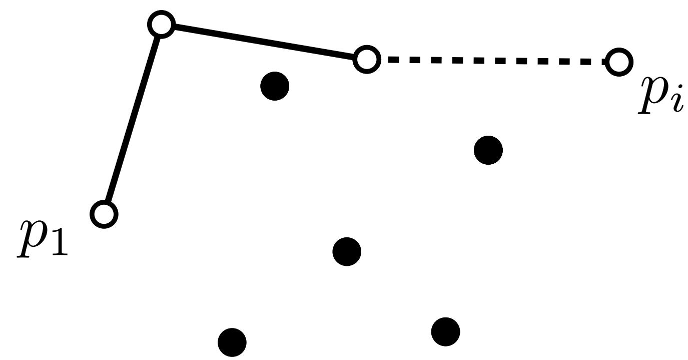
(p_i is below last hull segment)



Add p_i to the upper hull

Left turn

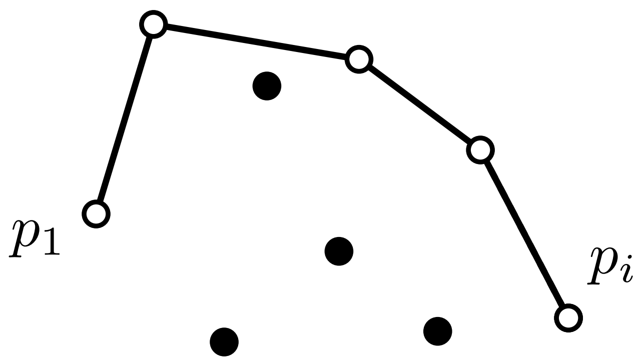
(p_i is above last hull segment)



Graham's Scan: update

Right turn

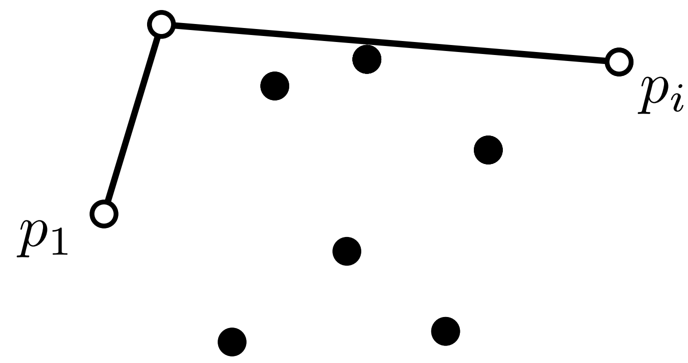
(p_i is below last hull segment)



Add p_i to the upper hull

Left turn

(p_i is above last hull segment)

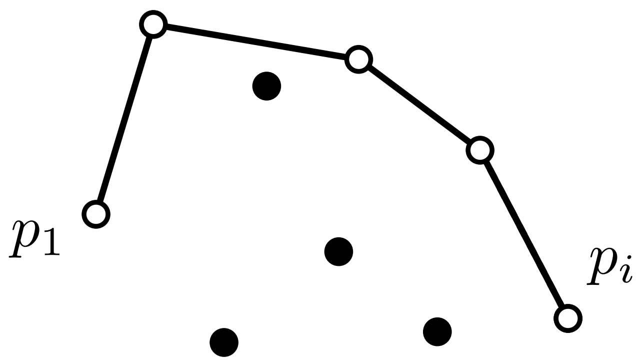


Add p_i but remove previous hull point until left turn disappears

Graham's Scan: update

Right turn

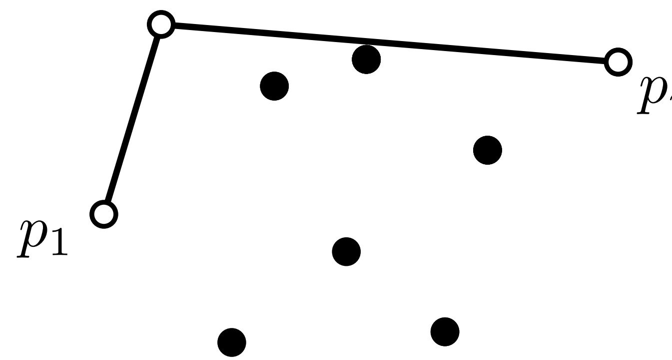
(p_i is below last hull segment)



Add p_i to the upper hull

Left turn

(p_i is above last hull segment)



Add p_i but remove previous hull point until left turn disappears

Similalrly for lower hull, after adding p_i:
while last three points of lower hull q, q', p_i are a right turn:
remove the middle point q'

Graham's Scan: pseudocode + runtime

```
Sort P by increasing x-coordinates
Add p_1, p_2 to U and L
for i = 3 to n do
    Add p_i to U and L
    while last three pts of U form left turn do
        Remove pt preceding p_i from U
    while last three pts of L form right turn do
        Remove pt preceding p_i from L
return L and reverse of U
```

Graham's Scan: pseudocode + runtime

```
Sort P by increasing x-coordinates
Add p_1, p_2 to U and L
for i = 3 to n do
    Add p_i to U and L
    while last three pts of U form left turn do
        Remove pt preceding p_i from U
    while last three pts of L form right turn do
        Remove pt preceding p_i from L
return L and reverse of U
```

Running time:

Sorting ⟶ $O(n \log n)$

Graham's Scan: pseudocode + runtime

```
Sort P by increasing x-coordinates
Add p1, p2 to U and L
for i = 3 to n do
    Add pi to U and L
    while last three pts of U form left turn do
        Remove pt preceding pi from U
    while last three pts of L form right turn do
        Remove pt preceding pi from L
return L and reverse of U
```

Running time:

Sorting $\longrightarrow O(n \log n)$

Each $p \in P$ is:

- added once to U (same for L) $\longrightarrow O(n)$
- removed at most once from U (same for L) $\longrightarrow O(n)$

Triplets checked in While loop heads $\longrightarrow O(n)$

Graham's Scan: pseudocode + runtime

Sort P by increasing x-coordinates
Add p_1, p_2 to U and L
for $i = 3$ to n **do**
    Add p_i to U and L
    while last three pts of U form left turn **do**
        Remove pt preceding p_i from U
    while last three pts of L form right turn **do**
        Remove pt preceding p_i from L
return L and reverse of U

$O(n \log n)$ time, but $O(n)$ space.

Time-optimal because of sorting (was exercise last year)

Graham's Scan: pseudocode + runtime

Sort P by increasing x-coordinates
Add p_1, p_2 to U and L
for $i = 3$ to n **do**
    Add p_i to U and L
    while last three pts of U form left turn **do**
        Remove pt preceding p_i from U
    while last three pts of L form right turn **do**
        Remove pt preceding p_i from L
return L and reverse of U

$O(n \log n)$ time, but $O(n)$ space.

Time-optimal because of sorting (was exercise last year)

Near-optimal time-space tradeoff:
sorting on RAM requires $T \cdot S = \Omega(n^2 / \log n)$. [Borodin–Cook '82]

Convex hull with good time-space tradeoff

Sorting in sublinear space

Theorem (Munro, Paterson 1980)
Given x and an unsorted array A, we can find the s smallest elements greater than x in A in a single pass, in $O(s)$ space and $O(n)$ time.
We can also sort in

- $O(n^2/s + n \log s)$ time
- $O(s)$ space
- with n/s passes.

Convex hull in sublinear space

Theorem (Chan–Chen 2007)
Given n points in $\mathbb{R}^2$, the convex hull can be computed in

- $O(n^2/s + n \log s)$ time
- $O(s)$ space
- with n/s passes.

Sublinear space convex hull pseudocode

$v :=$ leftmost point
while $v \neq$ rightmost point **do**
    Find vertical slab σ with s pts whose left wall contains v
    $q_0, \ldots, q_j =$ upper hull of $P \cap \sigma$
    for all $p \in P$ to the right of σ **do**
        while $q_{j-1} q_j p$ is left turn **do**
            $j := j - 1$
        $j := j + 1, \quad q_j := p$
    Print$(q_0, \ldots, q_j)$
    $v := q_j$

Sublinear space convex hull pseudocode

$v :=$ leftmost point
while $v \neq$ rightmost point **do**
    Find vertical slab σ with s pts whose left wall contains v
    $q_0, \ldots, q_j =$ upper hull of $P \cap \sigma$
    for all $p \in P$ to the right of σ **do**
        while $q_{j-1} q_j p$ is left turn **do**
            $j := j - 1$
        $j := j + 1, \quad q_j := p$
    Print($q_0, \ldots, q_j$)
    $v := q_j$

$2\lceil n/s \rceil$ passes, $O(s)$ space, $O((n/s) \cdot (n + s \log s))$ time

Linear Programming in low-dimensional space

LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

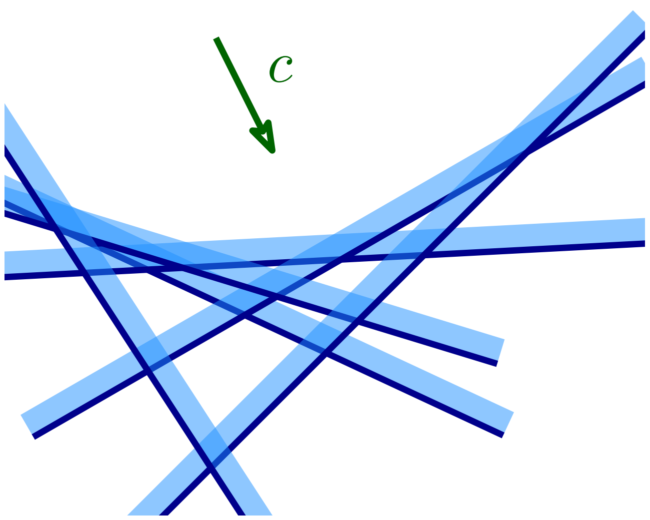
$$\min c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \leq b_1$$
$$a_{21}x + a_{22}y \leq b_2$$
$$\cdots$$
$$a_{n1}x + a_{n2}y \leq b_n$$

LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\min c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \leq b_1$$
$$a_{21}x + a_{22}y \leq b_2$$
$$\cdots$$
$$a_{n1}x + a_{n2}y \leq b_n$$

Given set H of n halfplanes, find extreme point in direction c.

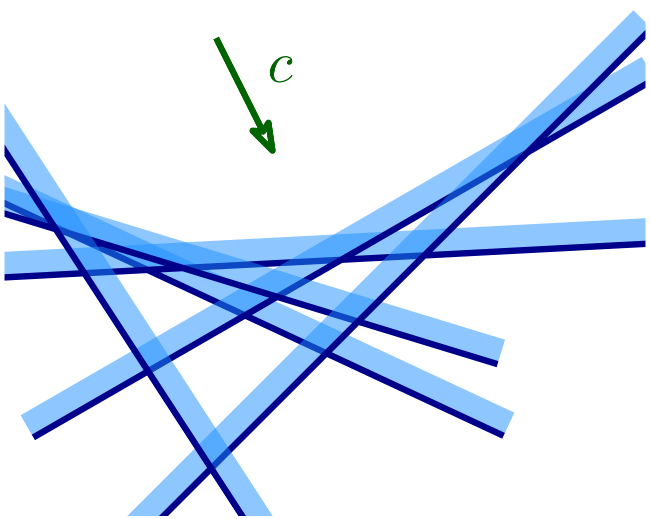


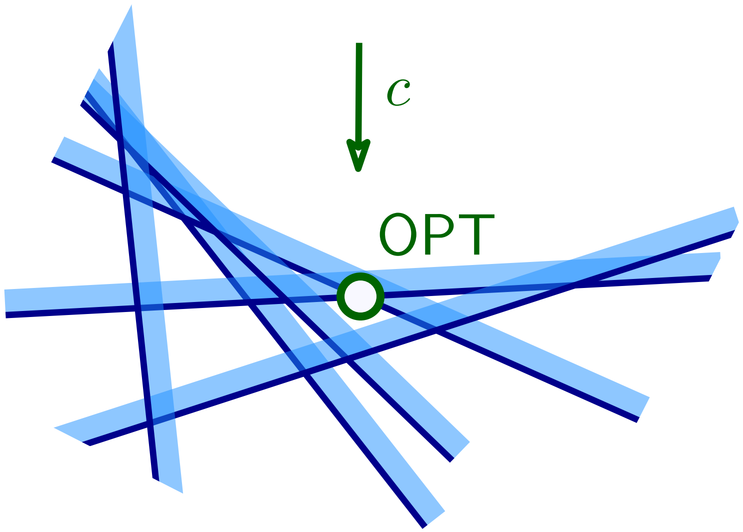
LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\min c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \leq b_1$$
$$a_{21}x + a_{22}y \leq b_2$$
$$\dots$$
$$a_{n1}x + a_{n2}y \leq b_n$$

Given set H of n halfplanes, find extreme point in direction c.





Dual Graham's scan solves it in $O(n \log n)$

Deterministic method: paired halfplanes

Lemma [Megiddo, Dyer 1984]
Assuming that $\bigcap_{h \in H} H \neq \emptyset$ is bounded from below, we can find OPT in $O(n)$ time.

Sublinear space low-dimensional LP

We prove:

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta}n^{1+\delta})$ time
- $O(\frac{1}{\delta}n^{\delta})$ space
- with $O(1/\delta)$ passes.

Sublinear space low-dimensional LP

We prove:

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta} n^{1+\delta})$ time
- $O(\frac{1}{\delta} n^{\delta})$ space
- with $O(1/\delta)$ passes.

We might return to:

Theorem (Chan–Chen 2007)
Given n half-spaces in $\mathbb{R}^d$ and $\delta > 0$, the lowest point of their intersection can be computed in

- $O_d(\frac{1}{\delta^{O(1)}} n)$ time
- $O_d(\frac{1}{\delta^{O(1)}} n^{\delta})$ space
- with $O(1/\delta^{d-1})$ passes.

Theorem (Chan–Chen 2007)
Given n half-spaces in $\mathbb{R}^d$, the lowest point of their intersection can be computed in $O_d(n)$ time and $O_d(\log n)$ space.

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

List(r, σ, H)

while H not read through **do**

$h_1, \ldots, h_r :=$ next r halfplanes from H

Compute $I = h_1 \cap \cdots \cap h_r$

Print vertical lines through vertices of I that fall in σ

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

```
List(r, σ, H)
  while H not read through do
     h_1, ..., h_r := next r halfplanes from H
     Compute I = h_1 ∩ ··· ∩ h_r
     Print vertical lines through vertices of I that fall in σ
```

Given stream H of halfplanes, produce stream of halfplanes.

```
Filter(r, σ, H)
  while H not read through do
     h_1, ..., h_r := next r halfplanes from H
     Compute I = h_1 ∩ ··· ∩ h_r
     Print halfplanes involved in ∂(I ∩ σ)
```

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

List(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print vertical lines through vertices of I that fall in σ

Given stream H of halfplanes, produce stream of halfplanes.

Filter(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print halfplanes involved in $\partial(I \cap \sigma)$

List and Filter work in one pass, in $O(r)$ space and $O(n \log r)$ time.

Sublinear time LP in $\mathbb{R}^2$

Parameter: r

Invariant: solution is in σ_i and defined by halfplanes in H_i

Pseudocode

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|H_i| = O(1)$ **then**
      return brute force solution for H_i
    Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$
    Decide which subslab has the solution, let that be σ_{i+1}.
    $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Pseudocode

LP(r, σ, H)
    $\sigma_0 := \mathbb{R}^2$
    for $i = 0, 1, \ldots$ **do**
        if $|H_i| = O(1)$ **then**
            return brute force solution for H_i
        Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$
        Decide which subslab has the solution, let that be σ_{i+1}.
        $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

Pseudocode

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|H_i| = O(1)$ **then**
      return brute force solution for H_i
    Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$
    Decide which subslab has the solution, let that be σ_{i+1}.
    $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**
      return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$
    Divide σ_i into r slabs with roughly same # of lines from
      $List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))$
    Decide which subslab has the solution, let that be σ_{i+1}

Pseudocode

```
LP(r, σ, H)
   σ_0 := ℝ^2
   for i = 0, 1, ... do
      if |H_i| = O(1) then
         return brute force solution for H_i
      Divide σ_i into r slabs with roughly same # of lines from List_{r,σ_i}(H_i)
      Decide which subslab has the solution, let that be σ_{i+1}.
      H_{i+1} := Filter_{r,σ_{i+1}}(H_i)
```

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

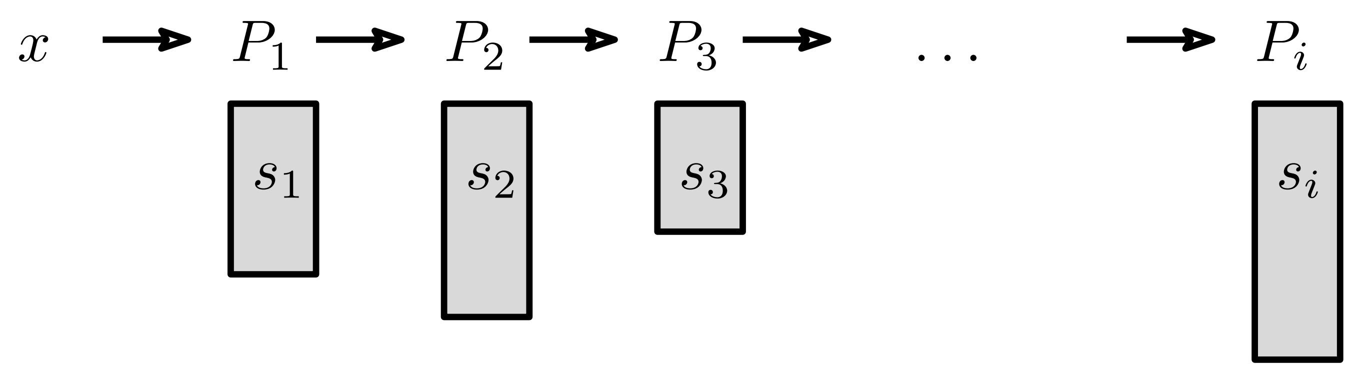
```
LP(r, σ, H)
   σ_0 := ℝ^2
   for i = 0, 1, ... do
      if |Filter_{r,σ_i}(...(Filter_{r,σ_1}(H)))| = O(1) then
         return brute force solution for Filter_{r,σ_i}(...(Filter_{r,σ_1}(H)))
      Divide σ_i into r slabs with roughly same # of lines from
                  List_{r,σ_i}(Filter_{r,σ_i}(...(Filter_{r,σ_1}(H))))
      Decide which subslab has the solution, let that be σ_{i+1}
```

One pass, maintain $r-1$ minima at inner slab walls

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

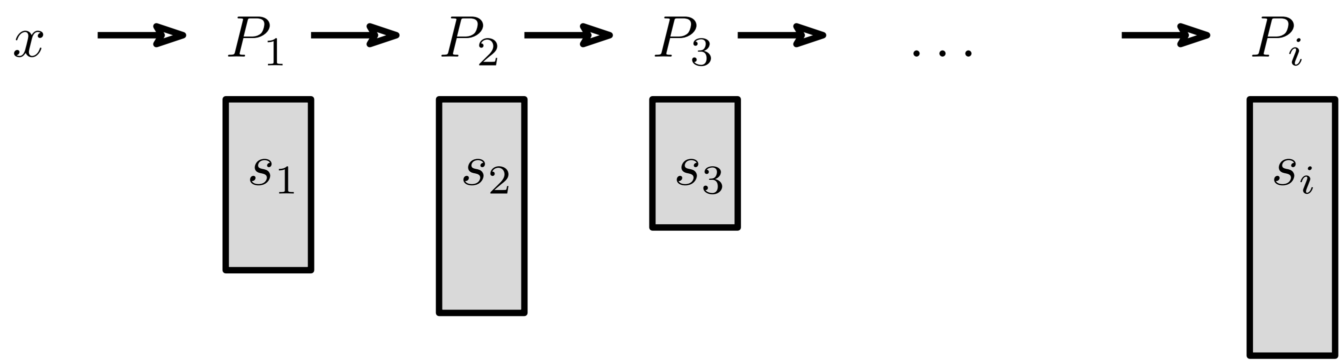
Pipeline:



Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:

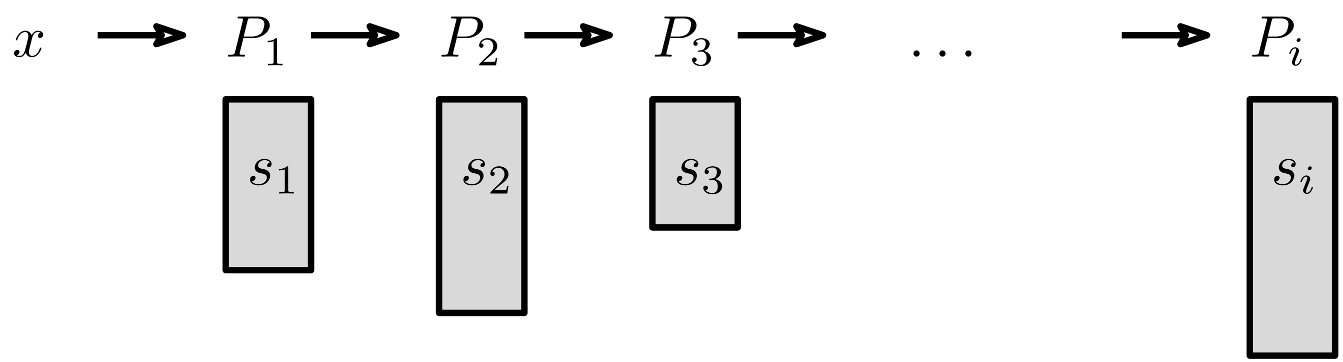


Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

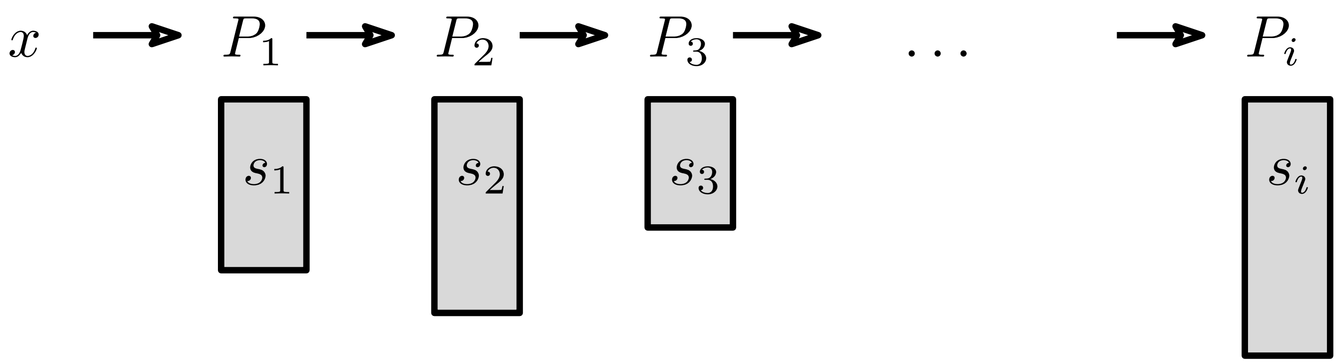
Simulation:

- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Simulation:

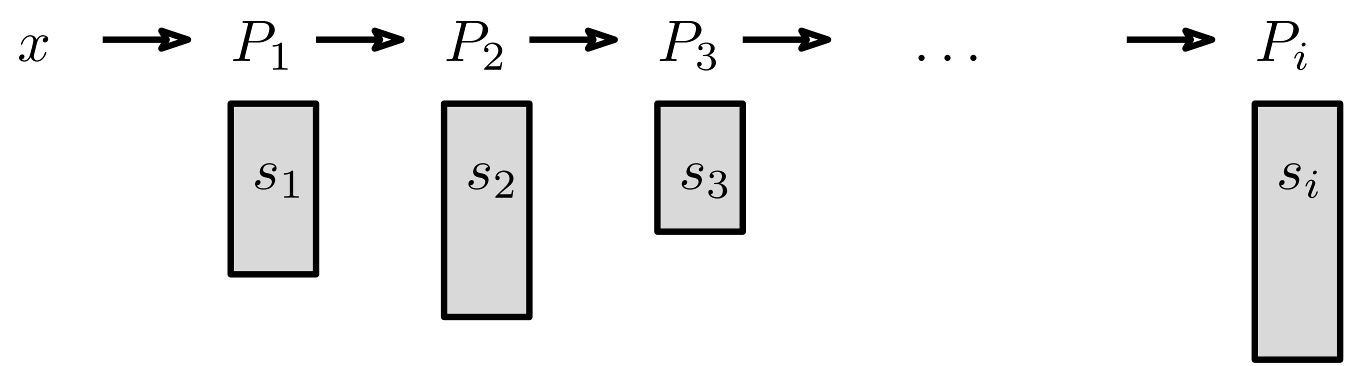
- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space: $\sum_j s_j + O(1)$

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Simulation:

- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space: $\sum_j s_j + O(1)$ Time (mini-hw): $O(\sum_j t_j)$

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}\big(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))\big)$

Decide which subslab has the solution, let that be σ_{i+1}

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}\left(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))\right)$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Subslab selection needs $O(r)$ space and $O(nr)$ time.

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Subslab selection needs $O(r)$ space and $O(nr)$ time.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Chan-Chen simple LP wrap-up

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta}n^{1+\delta})$ time
- $O(\frac{1}{\delta}n^{\delta})$ space
- with $O(1/\delta)$ passes.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Chan-Chen simple LP wrap-up

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta} n^{1+\delta})$ time
- $O(\frac{1}{\delta} n^{\delta})$ space
- with $O(1/\delta)$ passes.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Set $r = n^{\delta/2}$.

$$O\left(n^{\delta/2} \cdot \tfrac{2}{\delta} + n^{\delta/2} \log^2 n\right) = O(\tfrac{1}{\delta} n^{\delta}) \qquad O\left(n \cdot n^{\delta/2} \cdot \frac{\log n}{\log n^{\delta/2}}\right) = O(\tfrac{1}{\delta} n^{1+\delta})$$

Approximate quantiles