Linear programming with limited workspace

Sándor Kisfaludi-Bak

Geometric algorithms with limited resources
Summer semester 2021
Overview

- Sorting with few passes
- A classic deterministic algorithm in $\mathbb{R}^2$
- Sublinear space LP (Chan–Chen ’07)

Low-dim linear programming
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.

Theorem (Munro, Paterson 1980)
A $p$-pass sorting algorithm needs $\Omega(n/p)$ space.
Linear Programming in low-dimensional space
Known LP algorithms

A: $n \times d$ matrix. ($d$ variables, $n$ constraints.)

max $cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination slow
Known LP algorithms

A: \( n \times d \) matrix. \((d \text{ variables, } n \text{ constraints})\)
\[
\max cx \text{ subject to } Ax \leq b.
\]

- Fourier-Motzkin elimination \( \text{slow} \)
- Simplex method \( \text{fast in practice, slow in worst-case} \)
Known LP algorithms

A: \( n \times d \) matrix. (\( d \) variables, \( n \) constraints.)
\[
\text{max } cx \text{ subject to } Ax \leq b.
\]

- Fourier-Motzkin elimination: slow
- Simplex method: fast in practice, slow in worst-case
- Ellipsoid method: slow in practice, poly time in worst-case
Known LP algorithms

A: $n \times d$ matrix. ($d$ variables, $n$ constraints.)

\[
\max cx \text{ subject to } Ax \leq b.
\]

- Fourier-Motzkin elimination \hspace{1cm} slow
- Simplex method \hspace{1cm} fast in practice, slow in worst-case
- Ellipsoid method \hspace{1cm} slow in practice, poly time in worst-case
- Interior point methods \hspace{1cm} poly time and practical

These use bit complexity!
Known LP algorithms

A: $n \times d$ matrix. ($d$ variables, $n$ constraints.)

\[
\max cx \text{ subject to } Ax \leq b.
\]

- Fourier-Motzkin elimination: slow
- Simplex method: fast in practice, slow in worst-case
- Ellipsoid method: slow in practice, poly time in worst-case
- Interior point methods: poly time and practical

These use bit complexity!

Open: poly LP solver for number of arithmetic operations. (e.g. Real RAM)

Best known by Clarkson, Matousek, Sharir, Welzl, Gärtner, Kalai (1996)

\[
O(d^2n) + 2^{O(\sqrt{d \log d})}
\]
LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\max c_1 x + c_2 y \text{ subject to}$$

$$a_{11}x + a_{12}y \leq b_1$$

$$a_{21}x + a_{22}y \leq b_2$$

$$\ldots$$

$$a_{n1}x + a_{n2}y \leq b_n$$
LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\begin{align*}
\max c_1 x + c_2 y & \text{ subject to } \\
 a_{11} x + a_{12} y & \leq b_1 \\
 a_{21} x + a_{22} y & \leq b_2 \\
 & \vdots \\
 a_{n1} x + a_{n2} y & \leq b_n
\end{align*}$$

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

\[\text{c}\]
LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

\[
\begin{align*}
\text{max } c_1 x + c_2 y & \quad \text{subject to} \\
 a_{11} x + a_{12} y & \leq b_1 \\
 a_{21} x + a_{22} y & \leq b_2 \\
 & \quad \vdots \\
 a_{n1} x + a_{n2} y & \leq b_n
\end{align*}
\]

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

Dual Graham’s scan solves it in $O(n \log n)$
Intro/reminder on geometric duality
Deterministic method: paired halfplanes

**Lemma** [Megiddo, Dyer 1984]

Assuming that $\bigcap_{h \in H} h \neq \emptyset$ is bounded from below, we can find $\text{OPT}$ 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
- $O\left(\frac{1}{\delta}n^{1+\delta}\right)$ time
- $O\left(\frac{1}{\delta}n^\delta\right)$ 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
- $O\left(\frac{1}{\delta}n^{1+\delta}\right)$ time
- $O\left(\frac{1}{\delta}n^\delta\right)$ space
- with $O\left(\frac{1}{\delta}\right)$ passes.

**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
- $O_d\left(\frac{1}{\delta^{O(1)}}n\right)$ time
- $O_d\left(\frac{1}{\delta^{O(1)}}n^\delta\right)$ space
- with $O\left(\frac{1}{\delta^{d-1}}\right)$ 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.

\[
\text{List}(r, \sigma, H)
\]

\[
\text{while } H \text{ not read through do}
\]

\[
h_1, \ldots, h_r := \text{next } r \text{ halfplanes from } H
\]

\[
\text{Compute } I = h_1 \cap \cdots \cap h_r.
\]

\[
\text{Print vertical lines through vertices of } I \text{ that fall in } \sigma
\]
Towards sublinear space LP: filtering and listing

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

$$\text{List}(r, \sigma, H)$$

while $H$ not read through do

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

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

Print vertical lines through vertices of $I$ that fall in $\sigma$

Given stream $H$ of halfplanes, produce stream of halfplanes.

$$\text{Filter}(r, \sigma, H)$$

while $H$ not read through do

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

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

Print halfplanes involved in $\partial (I \cap \sigma)$
Towards sublinear space LP: filtering and listing

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

\begin{enumerate}
\item \textbf{List($r, \sigma, H$)}
\begin{enumerate}
\item \textbf{while} $H$ not read through \textbf{do}
\item $h_1, \ldots, h_r := \text{next } r \text{ halfplanes from } H$
\item Compute $I = h_1 \cap \cdots \cap h_r$
\item Print vertical lines through vertices of $I$ that fall in $\sigma$
\end{enumerate}
\end{enumerate}

Given stream $H$ of halfplanes, produce stream of halfplanes.

\begin{enumerate}
\item \textbf{Filter($r, \sigma, H$)}
\begin{enumerate}
\item \textbf{while} $H$ not read through \textbf{do}
\item $h_1, \ldots, h_r := \text{next } r \text{ halfplanes from } H$
\item Compute $I = h_1 \cap \cdots \cap h_r$
\item Print halfplanes involved in $\partial(I \cap \sigma)$
\end{enumerate}
\end{enumerate}

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 $\sigma_i$ and defined by halfplanes in $H_i$
LP($r, \sigma, H$)

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

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

if $|H_i| = O(1)$ then

    return brute force solution for $H_i$

Divide $\sigma_i$ into $r$ slabs with roughly same # of lines from $List_{r, \sigma_i}(H_i)$

Decide which subslab has the solution, let that be $\sigma_{i+1}$.

$H_{i+1} := Filter_{r, \sigma_{i+1}}(H_i)$
Pseudocode

\[
\text{LP}(r, \sigma, H) \\
\sigma_0 := \mathbb{R}^2 \\
\text{for } i = 0, 1, \ldots \text{ do} \\
\quad \text{if } |H_i| = O(1) \text{ then} \\
\quad \quad \text{return brute force solution for } H_i \\
\quad \text{Divide } \sigma_i \text{ into } r \text{ slabs with roughly same # of lines from } \text{List}_{r, \sigma_i}(H_i) \\
\quad \text{Decide which subslab has the solution, let that be } \sigma_{i+1}. \\
\quad H_{i+1} := \text{Filter}_{r, \sigma_{i+1}}(H_i)
\]

Issue: \( H_i \) can’t be stored. We need to recompute it every time.
LP\((r, \sigma, H)\)

\[
\sigma_0 := \mathbb{R}^2
\]

for \(i = 0, 1, \ldots\) do

if \(\lvert H_i \rvert = O(1)\) then

return brute force solution for \(H_i\)

Divide \(\sigma_i\) into \(r\) slabs with roughly same \# of lines from \(List_{r,\sigma_i}(H_i)\)

Decide which subslab has the solution, let that be \(\sigma_{i+1}\).

\[
H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)
\]

Issue: \(H_i\) can’t be stored. We need to recompute it every time.

LP\((r, \sigma, H)\)

\[
\sigma_0 := \mathbb{R}^2
\]

for \(i = 0, 1, \ldots\) do

if \(\lvert Filter_{r,\sigma_{i}}(\ldots(Filter_{r,\sigma_{1}}(H)))\rvert = O(1)\) then

return brute force solution for \(Filter_{r,\sigma_{i}}(\ldots(Filter_{r,\sigma_{1}}(H)))\)

Divide \(\sigma_i\) 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 \(\sigma_{i+1}\)
Pseudocode

\[ \text{LP}(r, \sigma, H) \]
\[ \sigma_0 := \mathbb{R}^2 \]
\[ \text{for } i = 0, 1, \ldots \text{ do} \]
\[ \quad \text{if } |H_i| = O(1) \text{ then} \]
\[ \quad \quad \text{return brute force solution for } H_i \]
\[ \quad \text{Divide } \sigma_i \text{ into } r \text{ slabs with roughly same \# of lines from } \text{List}_{r, \sigma_i}(H_i) \]
\[ \quad \text{Decide which subslab has the solution, let that be } \sigma_{i+1}. \]
\[ H_{i+1} := \text{Filter}_{r, \sigma_{i+1}}(H_i) \]

Issue: \( H_i \) can’t be stored. We need to recompute it every time.

\[ \text{LP}(r, \sigma, H) \]
\[ \sigma_0 := \mathbb{R}^2 \]
\[ \text{for } i = 0, 1, \ldots \text{ do} \]
\[ \quad \text{if } |\text{Filter}_{r, \sigma_i}(...) \text{Filter}_{r, \sigma_1}(H))| = O(1) \text{ then} \]
\[ \quad \quad \text{return brute force solution for } \text{Filter}_{r, \sigma_i}(...) \text{Filter}_{r, \sigma_1}(H)) \]
\[ \quad \text{Divide } \sigma_i \text{ into } r \text{ slabs with roughly same \# of lines from } \]
\[ \quad \text{\text{List}_{r, \sigma_i}(\text{Filter}_{r, \sigma_i}(...) \text{Filter}_{r, \sigma_1}(H)))} \]
\[ \quad \text{Decide which subslab has the solution, let that be } \sigma_{i+1} \]

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 workspace $s_j$ and time $t_j$?

Pipeline:

$$\quad x \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \ldots \rightarrow P_i$$

$$\quad s_1 \quad s_2 \quad s_3 \quad \ldots \quad s_i$$
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 workspace \( s_j \) and time \( t_j \)?

Pipeline:

Each \( P_j \) is either waiting for input, or ready to execute.
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 workspace $s_j$ and time $t_j$?

Pipeline:

Each $P_j$ is either waiting for input, or ready to execute.
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 workspace $s_j$ and time $t_j$?

Pipeline:

Each $P_j$ is either waiting for input, or ready to execute.
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 workspace $s_j$ and time $t_j$?

Pipeline:

$x \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \ldots \rightarrow P_i$

Each $P_j$ is either waiting for input, or ready to execute.
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

\begin{center}
\begin{tabular}{|c|}
\hline
Filter($r, \sigma, H$) \\
\hline
$\sigma_0 := \mathbb{R}^2$ \\
\hline
\textbf{for } $i = 0, 1, \ldots$ \textbf{ do} \\
\quad \textbf{if } $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ \textbf{ then} \\
\quad \quad \textbf{return} brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$ \\
\quad \textbf{Divide } $\sigma_i$ \textbf{ into } $r$ \textbf{ slabs:} \\
\quad \quad \textbf{ApproxQuant}_r(\text{List}_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))))$ \\
\quad \textbf{Decide which subslab has the solution, let that be } $\sigma_{i+1}$ \\
\hline
\end{tabular}
\end{center}
Time and space needs

**Filter**($r, \sigma, H$)

\[ \sigma_0 := \mathbb{R}^2 \]

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

if \(|\text{Filter}_{r,\sigma_i}(...(\text{Filter}_{r,\sigma_1}(H)))| = O(1)\) then

return brute force solution for \(\text{Filter}_{r,\sigma_i}(...(\text{Filter}_{r,\sigma_1}(H)))\)

Divide \(\sigma_i\) into \(r\) slabs:

**ApproxQuant**$_r$\((\text{List}_{r,\sigma_i}(\text{Filter}_{r,\sigma_i}(...(\text{Filter}_{r,\sigma_1}(H))))))\)

Decide which subslab has the solution, let that be \(\sigma_{i+1}\)

Let \(n_i = |H_i|\). There are \(\log_r(n)\) iterations, \(O(\log_r n)\) passes.
Time and space needs

\[ \text{Filter}(r, \sigma, H) \]

\[ \sigma_0 := \mathbb{R}^2 \]

\textbf{for} \ i = 0, 1, \ldots \ \textbf{do}

\textbf{if} \ \left| \text{Filter}_{r, \sigma_i}(...) \right| = O(1) \ \textbf{then}

\hspace{1em} \textbf{return} \ \text{brute force solution for } \text{Filter}_{r, \sigma_i}(...) \]

\text{Divide } \sigma_i \ \text{into } r \ \text{slabs:}

\[ \text{ApproxQuant}_r \left( \text{List}_{r, \sigma_i} \left( \text{Filter}_{r, \sigma_i}(...) \right) \right) \]

\text{Decide which subslab has the solution, let that be } \sigma_{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:

\( O(r^i) = O(r \log_r n) \) space, \( O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r) \) time
Time and space needs

\[ \text{Filter}(r, \sigma, H) \]
\[ \sigma_0 := \mathbb{R}^2 \]
\[ \text{for } i = 0, 1, \ldots \text{ do} \]
\[ \quad \text{if } |\text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H)))| = O(1) \text{ then} \]
\[ \quad \quad \text{return brute force solution for } \text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H))) \]
\[ \quad \text{Divide } \sigma_i \text{ into } r \text{ slabs:} \]
\[ \quad \quad \text{ApproxQuant}_r(L_{r,\sigma_i}(\text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H)))))) \]
\[ \quad \text{Decide which subslab has the solution, let that be } \sigma_{i+1} \]

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

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

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

\[ \text{Filter}(r, \sigma, H) \]
\[ \sigma_0 := \mathbb{R}^2 \]
\[ \text{for } i = 0, 1, \ldots \text{ do} \]
\[ \quad \text{if } |\text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H)))| = O(1) \text{ then} \]
\[ \quad \quad \text{return brute force solution for } \text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H))) \]
\[ \quad \text{Divide } \sigma_i \text{ into } r \text{ slabs:} \]
\[ \text{ApproxQuant}_r(\text{List}_{r,\sigma_i}(\text{Filter}_{r,\sigma_i}(\ldots(\text{Filter}_{r,\sigma_1}(H)))))) \]
\[ \text{Decide which subslab has the solution, let that be } \sigma_{i+1} \]

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

Filter pipeline (plus List) in iteration \( i \) needs:
\( O(r i) = O(r \log_r n) \) 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\( (r, \sigma, H) \)

\[
\sigma_0 := \mathbb{R}^2
\]

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 \( \sigma_i \) into \( r \) slabs:

\[
\text{ApproxQuant}_r(List_{r, \sigma_i}(Filter_{r, \sigma_i}(\ldots(Filter_{r, \sigma_1}(H))))))
\]

Decide which subslab has the solution, let that be \( \sigma_{i+1} \)

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

Filter pipeline (plus List) in iteration \( i \) needs:

\[
O(r^i) = O(r \log_r n) \quad \text{space, } O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r) \quad \text{time}
\]

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.
Goal:
Given unsorted stream $S$ of $n$ numbers, output $r$ entries $a_1 \leq \cdots \leq a_r$ so that the rank of $a_i$ and $a_{i+1}$ in the sorting of $S$ differ by at most $O(n/r)$. 
Approximate quantiles

Goal:
Given unsorted stream $S$ of $n$ numbers, output $r$ entries $a_1 \leq \cdots \leq a_r$ so that the rank of $a_i$ and $a_{i+1}$ in the sorting of $S$ differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose $n/k$ is power of 2.

$PartSort_k$ : Repeatedly read next $k$ elements, output them in sorted order.

- Space: $O(k)$, Time: $O\left(\frac{n}{k} \cdot k \log k\right) = O(n \log k)$
Approximate quantiles

**Goal:**
Given unsorted stream $S$ of $n$ numbers, output $r$ entries $a_1 \leq \cdots \leq a_r$ so that the rank of $a_i$ and $a_{i+1}$ in the sorting of $S$ differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose $n/k$ is power of 2.

**PartSort$_k$:** Repeatedly read next $k$ elements, output them in sorted order.

Space: $O(k)$, Time: $O\left(\frac{n}{k} \cdot k \log k\right) = O(n \log k)$

**SampleMerge$_k$:** Read in next $k + k$ elements $a_1, \ldots, a_k, b_1, \ldots, b_k$ ($a$ and $b$ are sorted). Merge the sequences $a_2, a_4, \ldots, a_k$ and $b_2, b_4, \ldots, b_k$. Output sorted merged sequence.

Space: $O(k)$, Time: $O(n)$
Approximate quantiles

**Goal:**
Given unsorted stream $S$ of $n$ numbers, output $r$ entries $a_1 \leq \cdots \leq a_r$ so that the rank of $a_i$ and $a_{i+1}$ in the sorting of $S$ differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose $n/k$ is power of 2.

PartSort$_k$:
Repeatedly read next $k$ elements, output them in sorted order.
Space: $O(k)$, Time: $O\left(\frac{n}{k} \cdot k \log k\right) = O(n \log k)$

SampleMerge$_k$:
Read in next $k + k$ elements $a_1, \ldots, a_k, b_1, \ldots, b_k$ ($a$ and $b$ are sorted). Merge the sequences $a_2, a_4, \ldots, a_k$ and $b_2, b_4, \ldots, b_k$. Output sorted merged sequence.
Space: $O(k)$, Time: $O(n)$

Idea:
Do PartSort$_k$, then repeatedly run SampleMerge$_k$ to get sample of size $k$.

$$\text{SampleMerge}_k(\text{SampleMerge}_k(\ldots ((\text{PartSort}_k(S))) \ldots )) \underbrace{\log(n/k)}_{\log(n/k)}$$
Lemma (assignment)
Let $a_1, \ldots, a_k$ be the result of $Sort_k$ and $\log(n/k)$ runs of $SampleMerge_k$. Then the rank of $a_i$ and $a_{i+1}$ differ by at most $O(\frac{n}{k} \log n)$ for all $i \in [k-1]$. 
Approximate quantiles as a pipeline

**Lemma** (assignment)

Let $a_1, \ldots, a_k$ be the result of $\text{Sort}_k$ and $\log(n/k)$ runs of $\text{SampleMerge}_k$. Then the rank of $a_i$ and $a_{i+1}$ differ by at most $O\left(\frac{n}{k} \log n\right)$ for all $i \in [k - 1]$.

Set $k = r \log n$.

*PostSelect*$_r$: 


**Approximate quantiles as a pipeline**

**Lemma (assignment)**
Let $a_1, \ldots, a_k$ be the result of $\text{Sort}_k$ and $\log(n/k)$ runs of $\text{SampleMerge}_k$. Then the rank of $a_i$ and $a_{i+1}$ differ by at most $O\left(\frac{n}{k} \log n\right)$ for all $i \in [k-1]$.

Set $k = r \log n$.

**PostSelect$_r$:**

**ApproxQuant$_r$:**

$$\text{PostSelect}_r(\text{SampleMerge}_k(\text{SampleMerge}_k(\ldots(\text{PartSort}_k(S)))\ldots))$$

$\log(n/k)$

**Time:** $O(n \log k + n + n/2 + n/4 + \cdots + k + n \log k) = O(n \log(r \log n))$

**Space:** $O(k + k + \cdots + k + k) = O(k \log(n/k)) = O(r \log^2 n)$
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
- $O\left(\frac{1}{\delta} n^{1+\delta}\right)$ time
- $O\left(\frac{1}{\delta} n^\delta\right)$ space
- with $O(1/\delta)$ passes.

Altogether: $O\left(r \log_r n + r \log^2 n\right)$ space and $O(\left(nr \log_r n\right)$ time.
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
- $O\left(\frac{1}{\delta} n^{1+\delta}\right)$ time
- $O\left(\frac{1}{\delta} n^\delta\right)$ space
- with $O(1/\delta)$ passes.

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

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

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

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