CS 5330 - Randomized Algorithms May 5, 2020

L_p Sampling in Data Streams

Lecturer: Arnab Bhattacharyya *Student(s): Bao Jinge, Liu Tianyu, Zhang Dongping*

1 Introduction

The streaming model of computation has become increasingly important for the analysis of massive datasets. In this model the algorithm is typically given only a few passes (usually one) over a massive dataset, and must maintain a small randomized sketch of what it has seen so far. L_p sampling, and corresponding algorithms known as L_p samplers,have found a wide range of applications in the design of data stream algorithms and beyond. In this reading project we analysed and discussed two important papers in this field and developed some basic understanding of the development and application of L_p sampling algorithms.

In the fist paper, the author develops a working algorithm for 1-pass L_p sampling with $poly(\varepsilon^{-1}logn)$-space, relative error $(1 \pm \epsilon)$ and an n^{-C} probability of failure of the algorithm. In the second paper, the author designs a algorithm for *perfect* L_p sampling for $p \in (0, 2)$ in a stream and resolve the space and time complexity of that. By reading more reference papers, we list some applications modified by using L_p sampler.

2 Preliminaries

2.1 Count-Sketch and Count-Max

For $f \in R^n$, count-sketch is a algorithm to obtain an estimate vector $y \in R^n$ such that $\|y - f\|_\infty$ is small by using a table $A \in R^{d*k}$. This vector y satisfies the following guarantee.

For any vector $v \in R^n$ and $\gamma \in [n]$, we define $v_{tail(\gamma)}$ to be v but with the top γ coordinates set equal to 0. And for $i \in [n]$, we define v_{-i} to be v with the i-th coordinate set to 0.

Theorem 1 *If $d = \Theta(log(1/\delta))$ and $k = 6/\epsilon^2$, then for a fixed $i \in [n]$ we have $|y_i - f_i| < \epsilon\|f_{tail(1/\epsilon^2)}\|_2$ with probability $1 - \delta$.Moreover, if $d = \Theta(log(n))$ and $c \geq 1$ is any constant, then we have $\|y - f\|_\infty < \epsilon\|f_{tail(1/\epsilon^2)}\|_2$ with probability $1 - n^{-c}$. Furthermore, if we instead set $y_j = median_{i\in[d]}|A_{i,h_i(j)}|$, then the same two bounds above hold replacing f with $|f|$.*

In this work, we only care about the index of the heaviest item in f, i.e. $i^* = \arg\max_i |f_i|$. So we use the data structure from count-sketch to test whether a fixed $j \in [n]$, if $j = \arg\max_i |f_i|$. To distinguish this algorithm from count-sketch, we call it count-max. And we have this lemma for it.

Lemma 2 *Let $c \geq 1$ be an arbitrarily large constant, set $d = \Theta(log(n))$ and k=2, and let A be a $d * k$ instance of count-max run of $f \in R^n$ using fully independent hash functions h_i and Gaussian random variables $g_i \sim \mathcal{N}(0, 1)$. Then with probability $1 - n^{-c}$ the following holds : for every $i \in [n]$, if $f_i > 20\|f_{-i}\|_2$ then count-max declares i to be the maximum, and if $|f_i| \leq max_{j\in[n]/i}|f_i|$, then count-max does not declare i to be the maximum. Thus if count-max declares $|f_i|$ to be the largest coordinate of f, it will be correct with high probability. Moreover, this result still hold if each bucket $A_{i,j}$ is scaled by a $_i, j \sim Uniform(\frac{99}{100}, \frac{101}{100})$ before reporting.*

3 Main Content

3.1 Definition of Approximate L_p Sampling & Perfect L_p Sampling

Here we give a formal definition about the guarantee given by an approximate L_p sampling referring to [7]

Definition 3 ***(Approximate L_p Sampler)*** *Let $f \in R^n$ and $v \in [0,1)$. For $p > 0$, an approximate L_p sampler with v-relative error is an algorithm which returns an index $i \in \{1,2,...,n\}$ such that for every $j \in \{1,2,...,n\}$*

$$Pr[i = j] = \frac{|f_j|^p}{||f_j||_p^p}(1+v) + O(n^{-c})$$

where $c \geq 1$ is some arbitrary large constant. The sampler gives index j like this is called approximate L_p sampler.

For $p = 0$, the problem is to return j with probability $1 \pm v\frac{\max 1, |f_j|}{|j:f_j \neq 0|} + O(n^{-c})$, which is named approximate L_0 sampler. The definition of perfect L_p sampler is as follows.

Definition 4 ***(Perfect L_p Sampler)*** *When $v = 0$ in Definition 3, the sampler depicted is called perfect L_p sampler.*

An L_p sampler is allowed to output *FAIL* with some δ. However, in this case it must not output any index.

3.2 1-Pass Relative Error L_p Sampling

For any $p \in [0,2]$, we give a 1-pass poly(ε^{-1}log n)-space algorithm which, given a data stream of length m with insertions and deletions of an n-dimensional vector a, with updates in the range $\{-M, -M+1, ..., M-1, M\}$, outputs a sample of $[n] = 1, 2, ..., n$ for which for all i the probability that i is returned is $(1 \pm \epsilon)\frac{|a_i|^p}{F_p(a)} \pm n^{-C}$, where a_i denotes the (possibly negative) value of coordinate i, $F_p(a) = \|a\|_p^p$ denotes the $p-th$ frequency moment (i.e., the $p-th$ power of the L_p norm), and $C > 0$ is an arbitrarily large constant. Here we assume that n, m,and M are polynomially related.

3.2.1 Sampling Algorithm

Theorem 5 *For any $p \in [0,2]$, there is a 1-pass α-relative-error augmented L_p sampler that runs in poly$(\alpha^{-1}\log(n))$ bits of space. Ignoring an initial data structure initialization stage (which doesn't look at the stream), the update time of the L_p sampler is poly$(\alpha^{-1}\log(n))$. There is also an n^{-C} probability of failure of the algorithm, in which case it can output anything. Here $C > 0$ is an arbitrarily large constant.*

In the construction of the above algorithm, we need an algorithm approximating heavy hitters:

Lemma 6 *Let $0 < \delta < 1$. Let a be a vector of length n initialized to zero. Let S be the stream mentioned above. There is an algorithm Heavy Hitters(S, B, δ, ϵ) that, with probability at least $1 - \delta$, returns all coordinates i for which $|a_i|^p \geq \frac{F_p}{B^{p/2}}$, together with an approximation $\tilde{a}_i$ such that $|a_i| \leq |\tilde{a}_i| \leq (1+\epsilon)|a_i|$*

The algorithm conceptually divide the coordinates into classes:

Definition 7 *Fix an $\eta = 1 + \Theta(\varepsilon)$. Let $B \geq 1$ be a parameter. For $log_n\tau + 1 = log_\eta(C'\varepsilon^{-1}) + 1 \leq t \leq C_\eta logn$, for some constant C_η that depends on η, define*

$$S_t = \{i \in [n] : [a_i] \in [\eta^{t-1}, \eta^t)\}$$

Call a level t $(1/B) - contributing$ if $|S_t| \cdot \eta^{pt} \geq \frac{F_p}{B}$

Let $h : [n] -> [n]$ be a hash function with some amount of limited independence.We form $r = O(logn)$ substreams $S_0, S_1, ...S_r$ where S_j denotes the stream updates which pertain only to coordinates i for which $h(i) \leq n2^{-j}$. We say that such an i is a *survivor* (with respect to a given S_j). By zooming in on the appropriate substream and counting the number of survivors that are heavy hitters, we can estimate S_j to within $(1 \pm \Theta(\varepsilon))$ for all S_t that contribute.

By Jayram and the second author this approach yields a 1-pass-$\Theta(\varepsilon)$-additive-error augmented sampler in the following way. We simply ignore the S_t that do not contribute, and let the $\tilde{s}_t$ be our approximations to the set sizes $|S_t|$ for contributing S_t. We sample a contributing St with probability

$$\frac{\tilde{s}_t \eta^{pt}}{\Sigma_{contributing\ S_t} \tilde{s}_t \eta^{pt}}$$

Given that we chose S_t, we output a heavy hitter in S_t found in the substream S_j used to estimate S_t. However, this only gives as additive error rather that a relative error, as items in non- contributing classes will never be sampled.

One main novelty of the paper is the following idea. Suppose we force every class to contribute. If we could do this, then we could try to apply the above sampling procedure to obtain a $\Theta(\varepsilon) - relative - error$ augmented sampler. To force each class to contribute, the idea is to inject new coordinates into the stream.

3.2.2 Time and Space Complexity

This section shows that the sampling algorithm can be implemented with the desired time and space complexity.

Lemma 8 *For any $p \in [0,2]$, the $L_p - sampler(S', p)$ has space complexity $poly(\varepsilon^{-1} logn)$ bits. Ignoring the time of Stream Transformation, which can be performed without looking at the data stream, the update time is $poly(\epsilon^{-1} logn)$.*

The space used by the algorithm for injecting coordinates and Heavy Hitters subroutine is $poly(\varepsilon^{-1} logn)$. The update time is dominated by that of the Heavy Hitters subroutines, and is $poly(\varepsilon^{-1} logn)$.

3.3 Perfect L_p sampling

This section summarise the main content of [7]. So far, for the perfect L_p sampling problem, no algorithm can solve it exactly using $poly(logn)$-bits of space. To tackle this problem, this paper design a perfect sampling algorithm and demonstrate the existence of perfect L_p samplers using $O(log^2 n * log(1/\upsilon)(loglogn)^2)$-bits of space for $p \in (0,2)$ and $O(log^3(n))$-bits for $p = 2$.

3.3.1 Sampling Algorithm

Below is the main sampling algorithm for this problem.

(1) Set $d = \theta(log(n))$, instantiate a $d * 2$ count-max table A, and set $\mu_{i,j} \sim Uniform[\frac{99}{100}, \frac{101}{100}]$ for each $(i,j) \in [d] * [2]$

(2) Duplicate updates to f to obtain the vector $F \in R^{n^c}$ so that $f_i = F_{i_j}$ for all $i \in [n]$ and $j = 1, 2, ..., n^{c-1}$, for some fixed constant c

(3) Choose i.i.d. exponential random variables $t = (t_1, t_2, ..., t_{n^c})$, and construct the stream $\zeta_i = F_i * rnd_v(1/t_i^{1/p})$

(4) Run A on the stream ζ, upon the end of the stream, set $A_{i,j} = \mu_{i,j} A_{i,j}$ for all $(i,j) \in [d] * [2]$

(5) If count-max declares that an index $i_j \in [n^c]$ is the max for some $j \in [n^{c-1}]$ based on the data structure A, then output $i \in [n]$. If A does not declare any index to be the max, output FAIL.

Let $f \in R^n$ be the input vector of the stream, we first duplicate updates to each f_i of n^{c-1} times to obtain a new vector $F \in R^{n^c}$, which means for each $i \in [n]$ we set $i_j = (i-1)*n^{c-1}+j, for j = 1, 2, ..., n^{c-1}$ to make $F_{i_j} = f_i$. Now we call F as the duplication of f, and thus we will have $|F_i|^p \leq n^{-c+1}\|F\|_p^p$ for all $p > 0$ and $i \in [n^c]$

The we take i.i.d. exponential random variables $(t_1, ..., t_n)$ with rate 1, and define $z \in R^{n^c}$ by $z_i = F_i/t_i^{1/p}$. Then we have $Pr[i_j = \arg\max_{i',j'} \{|z_{i'_{j'}}|\}] = |F_{i_j}|^p/\|F\|_p^p$. And we also have $\sum_{j \in [n^{c-1}]} |F_{i_j}|^p/\|F\|_p^p = |f_i|^p/\|f\|_p^p$, so after we get $i_j \in [n^c]$ satisfying $i_j = \arg\max_{i',j'} \{|z_{i'_{j'}}|\}$, we can return the index $i \in [n]$.

In the third step, for any large constant c, fix $\upsilon > n^{-c}$. To speed up the update time, we scale F_i by $rnd_\upsilon(1/t_i^{1/p})$, where $rnd_\upsilon(x)$ rounds $x > 0$ down to the nearest value in $\{.., (1+\upsilon)^{-1}, 1, (1+\upsilon), (1+\upsilon)^2, ...\}$ (i.e., the nearest power of $(1+\upsilon)^j (for j \in Z)$. This will generate a separated stream τ, where $\tau_i = F_i * rnd_\upsilon(1/t_i^{1/p})$, and we also notice that $\tau_i = (1 \pm O(\tau))z_i$. It's important that this is an order preserving transform, so the index of the largest element τ_{i*} in τ should also be the index of largest element z_{i*} in z.

In the fourth step, we run a $d*2$ instance $A \in R^{d*2}$ of count-max, with $d = \theta(log(n))$ on τ. After the whole stream process, we scale each $A_{i,j}$ by a uniform random variable $\epsilon_{i,j}$ from interval $[\frac{99}{100}, \frac{101}{100}]$. This can ensure that the failure threshold is randomized, and also reduce the probability of result influenced by a small error. The count-max algorithm will either output an index $i_j \in [n^c]$ as the maximum, or report nothing. If an index i_j is reported, our algorithm will return index i. If no index reported, our algorithm return FAIL. Let $i^* = \arg\max_i |\tau_i| = D(1)$, where $D(1)$ is the first anti-rank. From Lemma 1 (todo), we know that if $|\tau_i| \geq 20\|\tau_{-i^*}\|_2$, count-max will return $i^* \in [n^c]$ with probability $1 - n^{-c}$

3.3.2 Correctness Analyse

Now we want to find the correctness of this algorithm. First, we need to analyze the conditional probability of failure given $D(1) = i$.

Lemma 9 *For every $1 \leq k < N - n^{9c/10}$, we have*

$$|z_{D(k)}| = [(1 \pm O(n^{-c/10})) \sum_{\tau=1}^{k} \frac{E_\tau}{E(\sum_{j=\tau}^{N} |F_{D(j)}|^p)}]^{-1/p}$$

This shows that the failure condition is nearly-independent of the value $D(1)$

Let ϵ_1 be the event that Lemma 9 holds. Let $\neg FAIL$ be the event that the algorithm L_p sampler does not output FAIL.

Lemma 10 *For $p \in (0, 2]$ a constant bounded away from 0 and any $\upsilon \geq n^{-c/60}, Pr[\neg FAIL|D(1)] = Pr[\neg FAIL] \pm \widetilde{O}(\upsilon)$ for every possible $D(1) \in [N]$*

Lemma 10 shows that the probability of failure can only change by an additive $\widetilde{O}(\upsilon)$ term given that any one value of $i \in [N]$ achieved the maximum (i.e., $D(1) = i$), which will lead to a $(1 \pm \widetilde{O}(\upsilon))$-relative error.

And we use the lemma below to bound the probability that the algorithm fails at all.

Lemma 11 *For $0 < p < 2$ a constant bounded away from 0 and 2, the probability that L_p Sampler outputs FAIL is at most $1 - \Omega(1)$, and for $p = 2$ is $1 - \Omega(1/log(n))$*

Put together those lemmas, we obtain the correctness of the algorithm.

Theorem 12 *Given any constant $c \geq 2, \upsilon \geq n^{-c}$, and $0 < p < 2$ there is a one-pass L_p sampler which returns an index $i \in [n]$ such that $Pr[i = j] = \frac{|f_j|^p}{\|f\|_p^p}(1 \pm \upsilon) \pm n^{-c}$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(log^2(n)log(1/\delta)(loglogn)^2)$ bits for $p < 2$, and $O(log^3(n)log(1/\delta))$ bits for $p = 2$. For $p < 2$ and $\delta = 1/poly(n)$, the space is $O(log^3(n)) - bits$.*

This follows that perfect L_p samplers exist using $O(log^2(n)log(1/\delta)(loglogn)^2)$ and $O(log^3(n)log(1/\delta))$ bits of space for $p < 2$ and $p = 2$ respectively.

Theorem 13 *Given $0 < p \leq 2$, for any constant $c \geq 2$ there is a perfect L_p sampler which returns an index $i \in [n]$ such that $Pr[i = j] = \frac{|f_i|^p}{||F||_p^p} \pm O(n^{-c})$ for all $j \in [n]$, and which fails with probability $\delta > 0$. The space required is $O(\log^2(n)\log(1/\delta))$ bits for $p < 2$ and $O(\log^3(n)log(1/\delta))$ bits for $p = 2$.*

3.3.3 Time and Space Complexity

This sampling algorithm can be implemented with the desired time and space complexity in the following way. First, we can optimize the update time by using a procedure called Fast-Update throughout the stream. Then, to optimize the space, We can use a random seed of length $O(log^2(n))$-bits to derandomize the L_p Sampler with Fast-Update to get the desired space cost. Finally,by using an additional auxiliary heavy-hitters data structure of [9] (called ExSk here) as a subroutine, the update time complexity can be reduced to $O(log(n))$ and this additional data structure will not increase the space or update time complexity of the entire algorithm.

3.4 Application

As is what mentioned in [10], L_p sampler leads to many improvements and a unification of well-studied streaming problems such as weighted sampling in turnstile model, cascaded norms, heavy hitters & block heavy hitters, L_p norm estimation. Besides, L_p sampler are often used as a black-box subroutine to design other algorithms. Moreover, applications in privacy is an another motivation for utilizing perfect L_p samplers[7].

3.4.1 Weighted Sampling

"A fundamental question that arises is to design algorithms to maintain a uniform sample of the forward distribution under both insertions and deletions or show that it is impossible", which is a famous question stated in [5]. By forward distribution, the authors mean to return a sample i with probability $|a_i|/||a||_1$, which is a L_1 sampling problem in turnstile model. Setting $p = 1$ in perfect L_p sampler, we can resolve this main open question. In computational geometry, [6] proposed a problem such as maintaining approximate range spaces and costs of Euclidean spanning trees, which needs a subroutine that maintains a random element from a given point set P undergoing multiple insertions and deletions. Setting $p = 0$ in the approximate L_p sampler in Theorem 5 can bring us the solution for the problem with $(1 + v)$ relative error. Although relative error exists in approximate L_0 sampler. In some case, when we know the support of the underlying vector A is at most k, by running the algorithm at most $O(k \log k)$ times, with constant probability all k non-zero items will be found.

Alongside this, we can also use the algorithm in [6], which gives a perfect L_0 sampler with $\delta = 1/poly(n)$ and space complexity $O(\log^3(n))$.

3.4.2 Cascaded Norms

Cascaded norms estimation is a problem comes from cascaded aggregates. Initial work in data stream processing considered how to estimate simple aggregates over the input. A more general question is what more complicated functions can be valued. For example, consider compound computations that first apply one function on subsets of the raw data, then aggregate partial results via additional functions, i.e. cascaded aggregates. A canonical example are the cascaded frequency moments applied to a matrix of data. Formally, given a $n \times d$ matrix A, the aggregate $F_k(F_p)(A)$ is defined as $\sum_{i=1}^{n}(\sum_{j=1}^{d}|A_{i,j}^p|)^k$ in [4]. [10] suggested an algorithm for approximating $F_k(F_p)(A)$ that uses L_p sampling as subroutine, which algorithm is a clever generalization of earlier F_k estimation algorithm proposed in [8].

3.4.3 Heavy Hitters & Block Heavy Hitters

The classical heavy hitters problem is to report all coordinates i for which $|a_i| \geq \Phi||a||_p$, where ϕ is an input parameter. For $p = 1$ this is solved by the **Count-Min** data structure, and for $p = 2$ by the **Count Sketch** data structure. The L_p sampler can give more general result for $p \in [0, 2]$. Indeed, run the L_p sampler $O(\Phi^{-1} \log \Phi^{-1})$ times, the list of samples will contain all heavy hitters. One can be sure that with high probability that if $|a_i| \leq \Phi||a||_p$, then i will not be reported.

An extended applications is that of computing block heavy hitters, which is the problem of reporting all rows a^i of an $n \times d$ matrix A for which $||a^i||1$ is at least a Φ fraction of the L_1 norm $||A||_1$. There rows are called block heavy hitters. The problem is first proposed in [1] and solved by a 1-pass algorithm using $poly(\Phi^{-1} \log n)$ bits of space in [2].

3.4.4 L_p Norm Estimation

The problem is to estimate L_p-sums defined as $\sum_i |A[i]|^p$(L_p sums are p-th powers of L_p norms). Moment estimation methods that work in the turnstile model can be used to estimate the L_p sums. Algorithms for estimating the L_p norms of vectors have had a central role in shaping the progress of data stream algorithms. In fact, L_p sampling can be regarded as an influential byproduct of research in L_p estimation algorithms. Not surprisingly, L_p estimators are also used in the design of L_p samplers as a subroutine. The L_p sampler gives an arbitrary F_k ($k > 2$) estimation and can be implemented in a single pass, based on [3], which tell us the problem of estimating F_k of a vector can be reduced to the problem of sampling according to F_2. It is worth noting that the algorithm in [10] and [7] does not use Nisan's pseudo random generator as a subroutine, potentially making it more practical.

3.4.5 Other Applications

In addition to applications mentioned above, there are other important applications. Organizing the applications into three main categories based on the nature of problems and application areas: Matrix sampling and computation encompassing problems in regression and low-rank approximation; Graph Stream covering connectivity and sub-graph counting; Dynamic geometric streams concerning spanning trees and width of a point set.

4 Open Problems

Although L_p sampler theory is perfect now, there are still some open problems remaining.

One notable shortcoming of the perfect sampler presented is the large update time. To obtain a perfect sampler as defined, the algorithm takes $poly(n)$ time to update its data structures after each entry in the stream. It's still an open problem to design an L_p sampler with optimal space dependency as well as poly-logarithmic update time.

Second, there is still a $log(n)$ factor gap between the upper and lower bound for perfect L_2 sampler, as the best known lower bound for any $p \geq 0$ is $\Omega(\log^2 n)$. Therefore, an open problem is whether this additional factor of $\log n$ is required in the space complexity of an L_2 sampler, perfect or otherwise.

Thirdly, we can find that L_0 and L_1 sampler are most popular for applications. We don't find any applications which use L_p sampler when p is a real value. It's worth thinking whether these real-value samplers can be applied to other problems.

Finally, we have proved L_p samplers when $p \in [0, 2]$. Meanwhile, there is no paper talking about L_p samplers with $p > 2$. It's worth thinking whether L_p samplers where $p > 2$ have application scenarios and whether there are some other L_p samplers with $p > 2$. Furthermore, there is an more general open problem: Can we design an L_p sampling algorithm which $p \geq 0$.

References

[1] Alexandr Andoni, Khanh Do Ba, and Piotr Indyk. Block heavy hitters. 2008.

[2] Alexandr Andoni and Piotr Indyk. Overcoming the 1 non-embeddability barrier: Algorithms for product metrics. 2008.

[3] Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 151–156. Society for Industrial and Applied Mathematics, 2004.

[4] Graham Cormode and Hossein Jowhari. L p samplers and their applications: A survey. *ACM Computing Surveys (CSUR)*, 52(1):1–31, 2019.

[5] Graham Cormode, Shanmugavelayutham Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st international conference on Very large data bases*, pages 25–36. VLDB Endowment, 2005.

[6] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

[7] Rajesh Jayaram and David P Woodruff. Perfect lp sampling in a data stream. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 544–555. IEEE, 2018.

[8] Thathachar S Jayram and David P Woodruff. The data stream space complexity of cascaded norms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 765–774. IEEE, 2009.

[9] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. *CoRR*, abs/1604.01357, 2016.

[10] Morteza Monemizadeh and David P Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010.

[11] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

A Exponential Order Statistics

There are some useful properties of the order statistics of n independent non-identically distributed exponential random variables. Let $(t_1, ..., t_n)$ be independent exponential random variables where t_i has mean $1/\lambda_i$ (i.e., t_i has rate λ_i). And t_i is given by the cumulative distribution function $Pr(t_i < x) = 1 - e^{-\lambda_i x}$

Definition 14 *Let $(t_1, ..., t_n)$ be independent exponentials. For k = 1,2,...,n, we define the k-th anti-rank $D(k) \in [n]$ of $(t_1, ..., t_n)$ to be the values $D(k)$ such that $t_{D(1)} \leq t_{D(2)} \leq ... \leq t_{D(n)}$*

Fact 15 *Let $(t_1, ..., t_n)$ be independent distributed exponentials, where t_i has rate $\lambda_i > 0$. For any $k = 1, 2, ..., n$, we have $t_{D(k)} = \sum_{i=1}^{k} \frac{E_i}{\sum_{j=i}^{n} \lambda_{D(j)}}$*

Fact 16 *For any $i = 1, 2, ..., n$,we have $Pr[D(1) = i] = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$*

B Data Steam Models

Data stream represents input data that comes at very high rate. High rate means it stresses communication and computing infrastructure, so it may hard to

- transmit(T) the entire input to the program
- compute(C) sophisticated functions on large pieces of the input at the rate it is presented
- store(S) capture temporarily or archive all of it long term

There are various models for dealing with data streams. Here we give a definition about the most general data dream model named **turnstile model**

Definition 17 ***(Turnstile Model)*** *The input stream $a1, a2, ...$ arrives sequentially, item by item, and describes an underlying signal A, a one-dimensional function $A : [1...N] \rightarrow R$. Input may comprise multiple streams or multidimensional signals. When a_i's are updates to $A[j]$'s, namely $a_i = (j, I_i)$, and $A_i[j] = A_{i-1}[j] + I_i$, where A_i is the signal after seeing the ith item in the stream, and I_i might be positive or negative, this model is called turnstile model.*

The model as above is inspired by a busy NY subway train station where the turnstile keeps track of people arriving and departing continuously. Obviously, a large number of people are in the subway. This is the appropriate model to study fully dynamic situations where there are inserts as well as deletes[11].