The CVM Algorithm for Estimating Distinct Elements in Streams

(Don Knuth, Stanford Computer Science Department)

(25 May 2023, revised 29 December 2023)

Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel [2] have recently proposed an interesting algorithm for the following problem: A stream of elements $(a_1, a_2, \ldots, a_m)$ is input, one at a time, and we want to know how many of them are distinct. In other words, if $A = \{a_1, a_2, \ldots, a_m\}$ is the set of elements in the stream, with multiplicities ignored, we want to know $|A|$, the size of that set. But we don't have much memory; in fact, $|A|$ is probably a lot larger than the number of elements that we can hold in memory at any one time. What is a good strategy for computing an unbiased estimate of $|A|$?

Their algorithm is not only interesting, it is extremely simple. Furthermore, it's wonderfully suited to teaching students who are learning the basics of computer science. (Indeed, ever since I saw it, a few days ago, I've been unable to resist trying to explain the ideas to just about everybody I meet.) Therefore I'm pretty sure that something like this will eventually become a standard textbook topic. This note is an initial approximation to what I might write about it if I were preparing a textbook about data streams.

Of course every important algorithm needs a name. Following a well-established precedent for other three-author procedures, such as the AKS algorithm, the KMP algorithm, and the LLL algorithm, I shall call it the *CVM algorithm* in the present note.

Let $A_t = \{a_1, \ldots, a_t\}$ denote the first t elements of the stream, ignoring multiplicity. The CVM algorithm maintains a buffer of limited size, which remembers a randomly chosen subset of those elements; let B_t be the contents of that buffer at time t. The algorithm also maintains a number p_t such that the following property always holds:

$$\Pr(a_j \in B_t) \;=\; p_t, \qquad \text{for } 1 \le j \le t. \tag{1}$$

It follows that the expected value of $|B_t|$ is $p_t\,|A_t|$; in other words,

$$|B_t|/p_t \text{ is an unbiased estimate of } |A_t|. \tag{2}$$

(The situation is actually more delicate than the informal equation (1) suggests, because the left-hand side of (1) is a probability while the right-hand side is a random variable! However, we shall see that the ideas can be justified and that (2) is 100% correct.)

The algorithm in [2] was devised so that property (1) would be invariant. But the original formulation contained a subtle bug, which is discussed in the appendix below. Fortunately there is a simple way to avoid that problem, and I shall present an amended algorithm here. Algorithm D is what I propose to call the CVM algorithm, and I believe it is indeed a candidate for inclusion in future textbooks.

There is one parameter, s, which denotes the maximum buffer size. It can be any positive integer. If the buffer overflows on the tth step — that is, if $|B_t| = s$ and we want to insert another element — we randomly delete one of those $s+1$ elements, and adjust p_t so that (1) remains true. The proper choice of p_t requires care; we use uniform deviates (uniformly random real fractions) for this purpose.

Algorithm D (*Distinct element estimation*). Given an arbitrary data stream $A = (a_1, a_2, \ldots, a_m)$ and a buffer size $s \ge 1$ as described above, this algorithm returns an unbiased estimate of $|A| = |\{a_1, a_2, \ldots, a_m\}|$. It uses a buffer B that's capable of holding up to s ordered pairs (a, u), where a is an element of the stream and u is a real number, $0 \le u < 1$.

D1. [Initialize.] Set $t \leftarrow 0$, $p \leftarrow 1$, and $B \leftarrow \emptyset$.

D2. [Done?] Terminate if $t = m$, returning the estimate $|B|/p$.

D3. [Input a.] Set $t \leftarrow t + 1$ and $a \leftarrow a_t$, the next element of the stream.

D4. [Remove a from B.] If B contains the pair (a, u), for any u, delete it.

D5. [Maybe put a in B.] Let u be a uniform deviate, independent of all others (namely a random real number in the range $0 \le u < 1$). If $u \ge p$, go back to step D2. Otherwise, if $|B| < s$, insert (a, u) into B and go back to step D2.

D6. [Maybe swap a into B.] (At this point $u < p$ and $|B| = s$.) Let (a', u') be the element of B for which u' is maximum. If $u > u'$, set $p \leftarrow u$. Otherwise replace (a', u') in B by (a, u) and set $p \leftarrow u'$. Then go back to step D2. ▮

• **An Example.** In order to become familiar with this algorithm, let's watch it at slow speed in one of the simplest possible cases. Assume that $s = 1$, and that the input stream consists of just $m = 3$ distinct elements (a_1, a_2, a_3). Since p is initially 1 and B is initially empty, step D5 will insert (a_1, u_1) into B, where u_1 is a uniform deviate.

The next time we reach step D5, however, we'll have $(a, u) = (a_2, u_2)$, where $u_2 < p = 1$ but $|B| = 1 = s$. So the buffer is full, and we'll proceed to step D6 for the first time. If $u_1 < u_2$, step D6 will set $p \leftarrow u_2$; otherwise it will set $p \leftarrow u_1$ and $B \leftarrow \{(a_2, u_2)\}$.

And when we reach step D5 with $t = 3$ and $(a, u) = (a_3, u_3)$, the possibilities multiply. Six scenarios are possible before termination, depending on the relative order between u_1, u_2, and u_3: Either

- $u_1 < u_2 < u_3$ and $p = u_2$ and $B = \{(a_1, u_1)\}$; or
- $u_1 < u_3 < u_2$ and $p = u_3$ and $B = \{(a_1, u_1)\}$; or
- $u_2 < u_1 < u_3$ and $p = u_1$ and $B = \{(a_2, u_2)\}$; or
- $u_2 < u_3 < u_1$ and $p = u_3$ and $B = \{(a_2, u_2)\}$; or
- $u_3 < u_1 < u_2$ and $p = u_1$ and $B = \{(a_3, u_3)\}$; or
- $u_3 < u_2 < u_1$ and $p = u_2$ and $B = \{(a_3, u_3)\}$.

Each of these six cases is equally likely. And in each case, B holds the single pair (a_j, u_j), where u_j is the minimum of $\{u_1, u_2, u_3\}$. Furthermore, p is always the *second* smallest of $\{u_1, u_2, u_3\}$.

We'll see in a moment that the average value of $1/p$ is 3 in each case (although the average value of p itself is $1/2$). Therefore the result returned by the algorithm, namely $|B|/p$, is unbiased: It has the correct expected value, namely $|A| = 3$.

Now let's make a slight change. Suppose $a_2 \neq a_1$, as before, but $a_3 = a_1$. Thus $|A|$ is now 2. Again the algorithm will base its computation on three uniform deviates, u_1, u_2, and u_3; and it's not difficult to see that the six equiprobable outcomes will now be either

- $u_1 < u_2 < u_3$ and $p = u_2$ and $B = \emptyset$; or
- $u_1 < u_3 < u_2$ and $p = u_2$ and $B = \{(a_3, u_3)\}$; or
- $u_2 < u_1 < u_3$ and $p = u_1$ and $B = \{(a_2, u_2)\}$; or
- $u_2 < u_3 < u_1$ and $p = u_3$ and $B = \{(a_2, u_2)\}$; or
- $u_3 < u_1 < u_2$ and $p = u_2$ and $B = \{(a_3, u_3)\}$; or
- $u_3 < u_2 < u_1$ and $p = u_2$ and $B = \{(a_3, u_3)\}$.

This time there's much more variety than before. The value that will be returned in step D2 turns out to be

$$0/U_{3,2} \quad \text{or} \quad 1/U_{3,3} \quad \text{or} \quad 1/U_{3,2} \quad \text{or} \quad 1/U_{3,2} \quad \text{or} \quad 1/U_{3,3} \quad \text{or} \quad 1/U_{3,2}, \tag{3}$$

each with probability $1/6$, where $U_{m,k}$ is a random variable:

$$U_{m,k} \;=\; \text{the } k\text{th smallest of } m \text{ independent uniform deviates.} \tag{4}$$

The probability distribution of $U_{m,k}$ is well known to be the "beta distribution with parameters k and $m + 1 - k$," namely

$$\Pr(U_{m,k} \le x) \;=\; k\binom{m}{k}\int_0^x t^{k-1}(1-t)^{m-k}dt \quad \text{for } 0 \le x \le 1; \tag{5}$$

this can be proved, for example, by the principle of inclusion and exclusion. Consequently

$$\mathrm{E}\,(1/U_{m,k}) \;=\; k\binom{m}{k}\int_0^1 t^{k-2}(1-t)^{m-k}dt \;=\; \frac{m}{k-1}, \quad \text{for } 1 \le k \le m. \tag{6}$$

(A similar argument shows that $\mathrm{E}\;U_{m,k} = k/(m+1)$.) The expected value of (3) therefore comes to

$$\frac{1}{6}\Bigl(0\Bigl(\frac{3}{1}\Bigr) + 1\Bigl(\frac{3}{2}\Bigr) + 1\Bigl(\frac{3}{1}\Bigr) + 1\Bigl(\frac{3}{1}\Bigr) + 1\Bigl(\frac{3}{2}\Bigr) + 1\Bigl(\frac{3}{1}\Bigr)\Bigr) \;=\; 2, \tag{7}$$

as it should. Algorithm D works, in this case!

• **Validity.** In the discussion that follows, it will be convenient to have a name for the uniform deviate u_t that is associated with a_t in step D5. Let's call it the "volatility" of a_t, because an element with low volatility will remain in B with high probability (until it occurs again in the stream and gets a brand new volatility).

We want to verify that condition (1) holds, where p_t is the value of p at the beginning of step D2. In fact we can verify a more precise statement:

Lemma M. *Let u_t be the uniform deviate generated in step D5. For every $a \in \{a_1, \ldots, a_t\}$, let j be maximum such that $a_j = a$. Then $(a, u_j) \in B$ in step D2 if and only if $u_j < p_t$.*

Proof. This statement holds vacuously when $t = 0$; we shall use induction on t. Notice that the value of p changes only in step D6.

Assuming that the lemma holds at time t, and that $t < m$, we need to prove that it's true also at time $t + 1$. After step D3 advances t, the logic of steps D4 and D5 guarantees that it remains true whenever $u \geq p$, and whenever $|B| < s$ after step D4.

The remaining case takes us to step D6. Then the buffer, together with (a, u), contains $s + 1$ pairs $\{(b_1, v_1), \ldots, (b_{s+1}, v_{s+1})\}$, whose volatilities are $s + 1$ independent uniform random numbers in the range $0 \leq v_j < p$, by the induction hypothesis. The effect of step D6 is to discard the pair (b_j, v_j) for which v_j is largest, and to set $p \leftarrow v_j$. And this change clearly preserves the assertion of the lemma, because the volatilities of the s remaining pairs are independent uniform random fractions less than the new p. ▮

• **Implementation.** To carry out Algorithm D we need a data structure for B that is capable of holding up to s pairs, where each pair (a, u) consists of a key a and its volatility u. We use this data structure as a dictionary in step D4, when we need to know if a given key is present. We also use it as a priority queue (sometimes called a "heap") in step D6, when we need to know the pair with maximum volatility. Both of those operations, as well as the insertion of new pairs and the deletion of existing pairs, should be efficient.

Thus the *treap* structure, introduced by Cecilia Aragon and Raimund Seidel in [1], and previously discovered by Jean Vuillemin with completely different motivations under the name "Cartesian trees" [10], is a perfect fit for Algorithm D. Treaps are binary trees in which every node carries two items of data, (a, u). Treap nodes are symmetrically ordered left-to-right on their first components, and heap-ordered on their second components. That is, every node in the left subtree of node (a, u) has a key less than a; every node in the right subtree has a key greater than a; and every ancestor node has a volatility greater than u.

It turns out that a *unique* binary tree satisfies both of those requirements, for every given set of (a, u) pairs in which all keys are distinct and all volatilities are distinct. It's the search tree that we get when keys are inserted in decreasing order of volatility. When the volatilities are random, as they are in our application, the shape of the treap is therefore a random binary search tree. Consequently, by the well-known theory of such trees (see for instance [5, Section 6.2.2]), the depths of its nodes tend to be at most about $2 \ln s$, when the size is bounded by s, and all of the operations that we need are quite efficient on average.

There's not space here for further details about treap manipulation. But I had lots of fun preparing the sample implementation [9]; interested readers are invited to download that program and to play with it, possibly even to read it. The algorithms are short and elegant, though nontrivial.

The program in [9] uses 31-bit pseudorandom binary fractions, between 0 and $1 - 2^{-31}$, instead of truly uniform random real-valued fractions. Therefore it will usually encounter volatilities u_t that exactly match previous volatilities $u_{t'}$, after t gets larger than 50000 or so. But the effects of such birthday-paradoxical coincidences on the final estimate are negligible, because they matter only when a "collision" happens at the precise probability cutoff (the final value of p). Rare events can occur when $u = p$ in step D5, or when more than one pair (a', u') has maximum volatility in step D6. They are reported so that the user can verify their insignificance. But such reports are hardly ever seen in practice. (Notice that several nodes may actually have to be removed in step D6, until all remaining volatilities are strictly less than the new value of p.)

• **Empirical Results.** Let's look at some numbers, in order to understand how Algorithm D actually works in practice. I'll consider several kinds of streams whose length, m, is 1,000,000 — a modest number, yet large enough to give realistic insights.

Stream $A1 = (8777534, 4951232, 6208406, \ldots)$ simply consists of a million 7-digit numbers chosen at random. (I used the random number generator of the Stanford GraphBase [6], using seed 314159, and calling *gb_unif_rand*(10000000) a million times.) It happens to have exactly $n = 951831$ distinct elements. Of these, 905276 occur just once; but one element occurs five times, and 41 of them occur four times.

Does Algorithm D make good estimates when given stream $A1$? It all depends, of course, on the buffer size, s. We certainly can't expect to get decent results when s has its minimum value 1; but what the heck, let's try it anyway. Nine random trials give the estimates

$$0,\ 298158,\ 577513,\ 797283,\ 1102121,\ 1577292,\ 1797809,\ 2067870,\ 2507278, \tag{8}$$

when sorted into order and truncated; so the median value, 1102121, actually isn't too far off. (Of course any value greater than 1000000 is ridiculous, because the stream has only a million elements to start with! Yet an unbiased estimate must sometimes be greater than m in order to balance cases when it is very small, if n is near m.) Another set of nine trials with $s = 1$ gives a median estimate of 523840, as well as a maximum estimate of 7242777. This experience confirms our conjecture that $s = 1$ will produce very flaky results. Nine trials with $s = 10$ exhibit some improvement:

$$436316,\ 778091,\ 785233,\ 792648,\ 836263,\ 859595,\ 1021711,\ 1190962,\ 1602061; \tag{9}$$

indeed, two of these estimates are within 10% of the truth. And with $s = 100$ there's further progress,

$$842439,\ 846607,\ 901347,\ 907046,\ 944089,\ 949036,\ 975275,\ 1027772,\ 1066863, \tag{10}$$

with two now within 1% of 951831. And of course things get even better with $s = 1000, 10000, 100000$:

$$\begin{gathered}933812,\ 940562,\ 941328,\ 941954,\ 952237,\ 952478,\ 963062,\ 970606,\ 976802;\\ 937620,\ 943866,\ 944334,\ 945349,\ 948613,\ 951159,\ 954800,\ 956146,\ 958711;\\ 947567,\ 951914,\ 952288,\ 952379,\ 953424,\ 953893,\ 954602,\ 954923,\ 957189.\end{gathered} \tag{11}$$

Consider now a second test stream $A2 = (1, 2, 3, \ldots)$, which is completely different from $A1$: Here we simply set $a_t = t \bmod 50000$, with no randomness involved. The stream therefore consists of 50000 distinct elements, repeated 20 times in the same order. Nine random trials with each of $s = 1$, 10, 100, 1000, 10000, 100000 yield the following approximations to the truth:

$$\begin{gathered}0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 327685;\\ 11315,\ 28376,\ 34651,\ 40548,\ 46217,\ 47496,\ 52164,\ 61905,\ 64441;\\ 41697,\ 44320,\ 46853,\ 47183,\ 47814,\ 47941,\ 48586,\ 48857,\ 55061;\\ 46935,\ 47869,\ 48059,\ 48329,\ 49155,\ 49318,\ 50056,\ 50893,\ 51414;\\ 49429,\ 49837,\ 49889,\ 50049,\ 50066,\ 50096,\ 50840,\ 50850,\ 50853;\\ 50000,\ 50000,\ 50000,\ 50000,\ 50000,\ 50000,\ 50000,\ 50000,\ 50000.\end{gathered} \tag{12}$$

I also tried six more streams with $m = 1000000$, in order to investigate how Algorithm D adapts itself to different kinds of input data. Stream $A3 = (0, 0, 0, \ldots, 1, 1, \ldots)$ simply has $a_t = \lfloor t/20 \rfloor$; it's sort of a dual to $A2$, and of course it's quite trivial. (Notice that $n = 50001$ in this case, not 50000, because the final element $a_{1000000} = 50000$ is unique. I've included $A3$ in this list because Algorithm D is supposed to work in all cases, whether they are trivial or not.)

Stream $A4 = (7534, 1232, 8406, \ldots)$ has $a_t = x_t + 10000\lfloor t/10000 \rfloor$, where x_t is a random 4-digit number. Thus it consists of ten thousand disjoint blocks of ten thousand numbers each. In this case the total number of distinct elements, n, is $632087 \approx (1 - 1/e)m$, as expected.

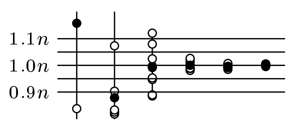
Stream $A5 = (26214270, 14654912, 16777150, \ldots)$ has $a_t = 2x_t \mid x_t$, which is the number we get by shifting the binary representation of x_t left one place and taking the bitwise OR with x_t itself, where x_t is the element from stream $A1$. For example, since $x_1 = 8777534 = (100001011110111100111110)_2$, we have $2x_1 \mid x_1 = (1100011111111111101111110)_2 = 26214270$.

Stream $A6 = (0101376, 6167318, 2103824, \ldots)$ has $a_t = x_{2t-1} \,\&\, x_{2t}$, the bitwise AND of x_{2t-1} and x_{2t}, where again each x_t comes from $A1$. For example, $8777534 \,\&\, 4951232 = (100001011110111100111110)_2 \,\&\, (010010111000110011000000)_2 = (000000011000110000000000)_2 = 0101376$. In this stream a number whose sideways sum is k is about three times as likely to occur as a number whose sideways sum is $k+1$. (The "sideways sum" of x is the number of 1s in x's binary representation.)

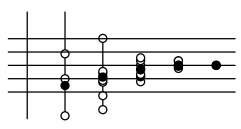
Stream $A7 = (338980, 740917, 359332, \ldots)$ has $a_t = x_t^2 + y_t^2$, where x_t and y_t are uniformly random 3-digit numbers. For example, $534^2 + 232^2 = 338980$.

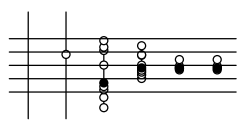
And finally, stream $A8 = (21787, 90991, 39323, \ldots)$ is perhaps the most interesting of them all. It consists of a million 5-digit numbers, for which each of the 100000 possible values occurs at least once. In fact, it was chosen to be uniformly random, among all $\left\{{1000000 \atop 100000}\right\}100000!$ such sequences. (Here $\left\{{m \atop n}\right\}$ denotes a Stirling number of the second kind. The number of ways to put m balls into n urns, leaving no urn empty, is $\left\{{m \atop n}\right\}n!$. Stream $A8$ was found on the 82nd random trial, after the first 81 trials failed to "fill every urn.")

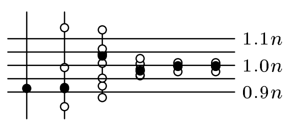
Here, in graphic form, are typical results obtained with all eight of the example streams.



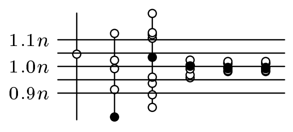
$A1$, $n = 951831$


$A2$, $n = 50000$

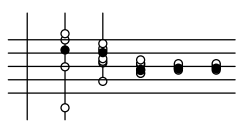

$A3$, $n = 50001$

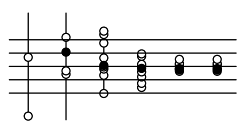


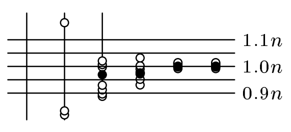
$A4$, $n = 632087$



$A5$, $n = 285039$


$A6$, $n = 405444$


$A7$, $n = 274399$



$A8$, $n = 100000$

Each stream has generated a vertical line of data for each of six buffer sizes, $s = (1, 10, 100, 1000, 10000, 100000)$, with nine independent estimates for each s. They've been "plotted" with white circles, except that the median estimate has a black circle. (Many of the circles overlap; the black one has been plotted last, hence it might cover all the others.) An estimate less than $0.8n$ or greater than $1.2n$ isn't shown. A vertical line runs from the smallest estimate to the largest one; however, that line has been truncated in cases where it would have fallen below $0.8n$ or above $1.2n$. The graphs for $A1$ and $A2$ correspond to the numeric data that was presented earlier.

• **A Coarser Method.** Suppose we replace step D6 by the following substitute:

D6′. [Halve the buffer.] (At this point $u < p$ and $|B| = s$.) Remove every element (a', u') of B for which $u' \geq p/2$; then set $p \leftarrow p/2$. If B hasn't changed, and if u is still less than the new value of p, repeat this step. Otherwise insert (a, u) into B if $u < p$. Then go back to step D2. ▌

In this version, p is always 2^{-k} for some integer k. Therefore step D6′ usually has a rather drastic effect on B, reducing the number of elements to about $s/2$.

Lemma M remains true. Hence the algorithm still returns an unbiased estimate.

As before, we can implement everything conveniently by exploiting the treap structure: To perform the new step D6′, we repeatedly remove the root until the treap either becomes empty or the volatility of its root is less than the new p (the former $p/2$). (The latter condition may already hold at the beginning of step D6′, in which case the treap doesn't change.)

It's not difficult to verify that the value of p in this modification, call it $p_{\rm bin}$, is always less than or equal to the value of p in the original Algorithm D. In fact, the new value "brackets" the former one: We have

$$p_{\rm bin} \leq p < 2p_{\rm bin} \tag{13}$$

by induction on the computation, throughout the process.

This binary version of Algorithm D, which we shall call Algorithm D′, doesn't give better estimates. But we shall see that it's a lot simpler to analyze. In the first place, we can prove that step D6′ almost never has to repeat itself.

Lemma R. *The probability that step D6′ is ever performed more than once before it returns to step D2 is at most* $m/2^{s+1}$.

Proof. At each time t, for $1 \le t \le m$, step D6′ will return immediately to D2 unless $u < p/2$ and each of the s volatilities in B is also less than $p/2$. But those volatilities are $s+1$ independent and uniformly distributed numbers between 0 and p. ▌

Suppose, for example, that $s = 100$. Then the chance of a repeat at time t is 2^{-101}, which is less than $4 \cdot 10^{-31}$. It won't happen! (Unless the stream length m amounts to many octillions.)

Next we will prove that p stays reasonably large throughout the computation.

Theorem S. *Let n be the number of distinct elements in the input stream. The probability that variable p in Algorithm D′ becomes less than $(s+1)/(4n)$ is at most $me^{-(s+1)/6} + m/2^{s+1}$.*

Proof. To fix the ideas, let's suppose that $n = 10000$ and $s = 100$. Then $4n/(s+1) \approx 392.2$, so $(s+1)/(4n)$ lies between $1/256$ and $1/512$. We want to show that Algorithm D′ is unlikely to reduce p from $1/256$ to $1/512$.

Indeed, at time t each of the n distinct elements a has a current volatility, which is either 1 (if a hasn't yet been seen) or the uniform deviate u that was most recently associated with a in step D5. Step D6′ will be activated if and only if 101 of those 10000 volatilities are less than p. But when $p = 1/256$, the expected number of such small volatilities is at most $np \approx 39.1$; and that's considerably smaller than 101. Hence the probability of such an extreme event must be quite small.

A special case of the well-known "Chernoff bound," proved for example in exercise 1.2.10–22(b) of [4], states that

$$\Pr\bigl(X_{n,p} \ge np(1+\delta)\bigr) \;\le\; e^{-\delta^2 np/(2+\delta)}, \quad \text{if } \delta \ge 0, \tag{14}$$

where the random variable $X_{n,p}$ represents the number of heads that show up during n independent flips of a p-coin (a coin that comes up heads with probability p).

So we plug in the values $n = 10000$, $p = 1/256$, $np(1+\delta) = 101$, and find $\delta \approx 1.61$, $\delta^2 np/(2+\delta) \approx 28.1$, and $e^{-28.1} \approx 6 \cdot 10^{-13}$. Yes; that's quite small.

In general we let $p = 1/2^k$ when $2^k \le 4n/(s+1) < 2^{k+1}$, and define δ by the formula $np(1+\delta) = s+1$. Then, letting $\alpha = np/(s+1)$, we have $\frac{1}{4} \le \alpha < \frac{1}{2}$ and $\delta = 1/\alpha - 1$ and $\delta^2\alpha/(2+\delta) = \alpha + 4/(1+\alpha) - 3 > 1/6$. Chernoff's bound, in (14), is therefore less than $e^{-(s+1)/6}$.

By adding $m/2^{s+1}$ to that upper bound, we can assume that step D6′ is performed at most m times. ▌

We're now ready to prove that the estimates of Algorithm D′ do not stray too far from the truth.

Theorem T. *Let Y be the estimate returned by Algorithm D′, and let $\epsilon \ge 0$. The probability that $Y \ge (1+\epsilon)n$ is at most $e^{-\epsilon^2(s+1)/(8+4\epsilon)}/(1 - e^{-\epsilon^2(s+1)/(8+4\epsilon)}) + me^{-(s+1)/6} + m/2^{s+1}$. Furthermore, the probability that $Y \le (1-\epsilon)n$ is at most $e^{-\epsilon^2(s+1)/8}/(1 - e^{-\epsilon^2(s+1)/8}) + me^{-(s+1)/6} + m/2^{s+1}$.*

Proof. Suppose, as above, that $2^k \le 4n/(s+1) < 2^{k+1}$. Theorem S tells us that p will be 2^{-k} or more, except with probability $me^{-(s+1)/6} + m/2^{s+1}$. By Lemma M and (14), we have $\Pr\bigl(Y \ge (1+\epsilon)n$ and $p = 2^{q-k}\bigr) \le \Pr(X_{n,2^{q-k}} \ge (1+\epsilon)n2^{q-k}) \le \exp(-2^q\epsilon^2(s+1)/(8+4\epsilon))$. Summing on $q \ge 0$ gives the first statement. For the second statement, we need another well-known Chernoff bound, namely

$$\Pr\bigl(X_{n,p} \le np(1-\delta)\bigr) \;\le\; e^{-\delta^2 np/2}, \quad \text{if } 0 \le \delta \le 1. \tag{15}$$

The second statement is, of course, trivial when $\epsilon > 1$. ▌

Corollary T. *Theorem T applies also to the estimate returned by Algorithm D.*

Proof. We have $np \ge (s+1)/4$ whenever $np_{\text{bin}} \ge (s+1)/4$, because of (13). ▌

Exercise 10 below illustrates how Corollary T can be used in practice to choose a buffer size s compatible with m, ϵ, and δ.

• (optional) **A Closer Look.** We noticed earlier that the sequence of decisions taken by Algorithm D can be broken down into $m!$ cases, depending on the relative order of the volatilities $u_1, \ldots, u_m$ that are assigned in step D5. Each permutation of those uniform deviates determines the size of B when the algorithm terminates, as well as the rank of the final cutoff probability p. If p is the kth smallest of the m volatilities, the estimate that's computed by Algorithm D is $|B|/U_{m,k}$, where $U_{m,k}$ is the random variable defined in (4) and (5).

For example, it turns out that the nine empirical estimates for stream $A1$, which were obtained in (10) for the case $s = 100$, were respectively

$$\frac{100}{U_{m,110}},\ \frac{100}{U_{m,107}},\ \frac{100}{U_{m,105}},\ \frac{100}{U_{m,107}},\ \frac{99}{U_{m,105}},\ \frac{100}{U_{m,104}},\ \frac{100}{U_{m,112}},\ \frac{100}{U_{m,102}},\ \frac{99}{U_{m,105}}, \tag{16}$$

where $m = 1000000$, with respect to the nine permutations of volatilities that were generated in each case.

[We observed in (6) that the random variable $1/U_{1000000,k}$ has the mean value $1000000/(k-1)$. Hence the mean values of the estimates in (16) are respectively

$$917431,\ 943396,\ 961538,\ 943396,\ 951923,\ 970873,\ 900900,\ 990099,\ 951923, \tag{17}$$

when truncated to integers. The difference between (17) and (10) comes from the fact that, for example, the random variable $100/U_{1000000,110}$ has a standard deviation of ≈ 88793.2; see exercise 6 below.]

Pursuing this line of thought leads us to a "meta-algorithm" that is equivalent to carrying out Algorithm D simultaneously for all buffer sizes at once.

We can assume without loss of generality that the input stream A is a *restricted growth string*—namely that $a_1 = 0$, that $a_2 = 0$ or 1, and in general that $a_{t+1} \le \max\{a_1, \ldots, a_t\} + 1$. (See [7, §7.2.1.5].) For if we take any stream A, there's a restricted growth string $(a'_1, \ldots, a'_m)$ such that $a'_i = a'_j$ if and only if $a_i = a_j$. (Let $a'_t = a'_j$ if $a_t = a_j$ for some $j < t$; otherwise let $a'_t = |\{a_1, \ldots, a_{t-1}\}|$.)

Algorithm M (*Meta-estimation of distinct elements*). Given a restricted growth string $A = (a_1, a_2, \ldots, a_m)$, this algorithm computes unbiased estimates $X_1, X_2, \ldots, X_{m-1}$ of the quantity $|A| = |\{a_1, a_2, \ldots, a_m\}|$, where X_s is the estimate for buffer size s that is returned by Algorithm D. It uses an auxiliary vector $(v_0, \ldots, v_{m-1})$ of volatilities, an auxiliary vector $(w_0, \ldots, w_{m-1})$ of sorted volatilities, and an auxiliary vector $(p_0, \ldots, p_{m-1})$ of cutoff probabilities. It does not try to be efficient.

M1. [Initialize.] Set $p_s \leftarrow v_s \leftarrow 1$ for $0 \le s < m$.

M2. [Loop on t.] Do step M3 for $t = 1, 2, \ldots, m$; then proceed to step M4.

M3. [Process a_t.] Set $v_{a_t} \leftarrow u_t$, where u_t is an independent uniform deviate. Let $w_0 \le \cdots \le w_{m-1}$ be the result of sorting $\{v_0, \ldots, v_{m-1}\}$ into nondecreasing order. Set $p_s \leftarrow \min\{p_s, w_s\}$ for $0 \le s < m$.

M4. [Finish.] Set $X_s = \bigl(\sum_{a=0}^{m-1} [v_a < p_s]\bigr)/p_s$, for $1 \le s < m$. ▌

Proceeding cautiously as we did earlier, we can readily become familiar with this algorithm by watching what it does with the simplest nontrivial stream, $A = (0, 1, 0)$. There are six equiprobable cases: Either

- $u_1 < u_2 < u_3$ and $p_1 = u_2$ and $X_1 = 0/p_1 = 0/u_2$; or
- $u_1 < u_3 < u_2$ and $p_1 = u_2$ and $X_1 = 1/p_1 = 1/u_2$; or
- $u_2 < u_1 < u_3$ and $p_1 = u_1$ and $X_1 = 1/p_1 = 1/u_1$; or
- $u_2 < u_3 < u_1$ and $p_1 = u_3$ and $X_1 = 1/p_1 = 1/u_3$; or
- $u_3 < u_1 < u_2$ and $p_1 = u_2$ and $X_1 = 1/p_1 = 1/u_2$; or
- $u_3 < u_2 < u_1$ and $p_1 = u_2$ and $X_1 = 1/p_1 = 1/u_2$.

(And $p_2 = 1$, $X_2 = 2$ in all cases.) Indeed, it's not difficult to verify that Algorithm M always ends with $p_0 < p_1 < \cdots < p_n = \cdots = p_{m-1} = 1$, when $|A| = n$.

Algorithm M is correct because of a simple consequence of Lemma M:

Corollary M. *Let $v_{t,a}$ be the current volatility of element a in step M3 at time t.* (Thus $v_{t,a} = u_j$ in the notation of Lemma M.) *Then, in step D2 at time t when Algorithm D has buffer size s, we have $a \in B$ if and only if $v_{t,a} < p_s$ in step M3 at time t.*

Proof. By step M3, the value of p_s is the smallest value of w_s that has occurred up to time t, where w_s is the $(s+1)$st smallest current volatility of any element. Therefore the value of p_s decreases in step M3 precisely at those times t when Algorithm D, given buffer size s, would decrease p. In fact, Algorithm D's variable p is the same as Algorithm M's variable p_s, at every time t. ▌

Returning to our observation that these algorithms essentially depend on the relative order of the volatilities, rather than on their actual values, we can now convert Algorithm M into an all-integer procedure that determines the *exact distribution* of every estimate X_s.

Algorithm M′ (*Discrete meta-estimation of distinct elements*). This reformulation of Algorithm M generates a random permutation $u_1u_2\ldots u_m$ of the integers $\{0,1,\ldots,m-1\}$ instead of generating uniform deviates. We now have $u_t=k$ if and only if u_t was the $(k+1)$st smallest of $\{u_1,\ldots,u_m\}$ in Algorithm M.

M1′. [Initialize.] Set $p_s\leftarrow v_s\leftarrow m$ for $0\le s<m$, and $u_t\leftarrow t-1$ for $1\le t\le m$.

M2′. [Loop on t.] Do step M3′ for $t=1,\,2,\,\ldots,\,m$; then proceed to step M4′.

M3′. [Process a_t.] Let k be a uniformly random integer in the $(m+1-t)$-element set $\{t,\ldots,m\}$, and swap $u_t\leftrightarrow u_k$. Then set $v_{a_t}\leftarrow u_t$. Let $w_0\le\cdots\le w_{m-1}$ be the result of sorting $\{v_0,\ldots,v_{m-1}\}$ into nondecreasing order. Set $p_s\leftarrow\min\{p_s,w_s\}$ for $0\le s<m$.

M4′. [Finish.] Set $X_s=\bigl(\sum_{a=0}^{m-1}[v_a<p_s]\bigr)/U_{m,p_s+1}$, for $1\le s<m$, where $U_{m,k}$ is defined in (4) and (5). ▮

For example, the six equiprobable cases for $A=(0,1,0)$ listed above have $u_1u_2u_3=(012,\,021,\,102,\,201,\,120,\,210)$; they yield the respective estimates $0/U_{3,2}$, $1/U_{3,3}$, $1/U_{3,2}$, $1/U_{3,2}$, $1/U_{3,3}$, $1/U_{3,2}$, as in (3).

I applied Algorithm M′ to stream $A1$ with five different sources of pseudorandom numbers, obtaining the following results:

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	X_{100000}	
$1/U_{m,2}$	$10/U_{m,12}$	$100/U_{m,107}$	$1000/U_{m,1049}$	$10000/U_{m,10498}$	$100000/U_{m,104982}$	
$1/U_{m,2}$	$10/U_{m,13}$	$100/U_{m,108}$	$1000/U_{m,1043}$	$9999/U_{m,10497}$	$100000/U_{m,104970}$	
$1/U_{m,2}$	$10/U_{m,11}$	$100/U_{m,108}$	$1000/U_{m,1050}$	$10000/U_{m,10521}$	$100000/U_{m,105160}$	$(A1)$
$1/U_{m,2}$	$10/U_{m,11}$	$100/U_{m,104}$	$999/U_{m,1075}$	$10000/U_{m,10545}$	$100000/U_{m,105210}$	
$1/U_{m,2}$	$10/U_{m,11}$	$100/U_{m,107}$	$1000/U_{m,1055}$	$10000/U_{m,10506}$	$100000/U_{m,105137}$	

We can now see the "structure" that underlies empirical results such as (8), (9), (10), and (11).

Similarly, Algorithm M′ reveals Algorithm D's typical behavior on the other seven example streams:

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	
$1/U_{m,9}$	$3/U_{m,103}$	$75/U_{m,1633}$	$892/U_{m,18170}$	$9918/U_{m,195556}$	
$0/U_{m,5}$	$3/U_{m,73}$	$66/U_{m,1360}$	$925/U_{m,18127}$	$9772/U_{m,196186}$	$(A2)$
$0/U_{m,4}$	$5/U_{m,110}$	$65/U_{m,1440}$	$944/U_{m,18379}$	$9760/U_{m,195041}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	
$0/U_{m,49}$	$9/U_{m,195}$	$99/U_{m,1811}$	$999/U_{m,19673}$	$10000/U_{m,201119}$	
$0/U_{m,6}$	$9/U_{m,199}$	$99/U_{m,2020}$	$999/U_{m,20244}$	$10000/U_{m,200652}$	$(A3)$
$0/U_{m,3}$	$9/U_{m,204}$	$99/U_{m,2041}$	$999/U_{m,20165}$	$9999/U_{m,198719}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	X_{100000}	
$1/U_{m,2}$	$10/U_{m,16}$	$100/U_{m,140}$	$998/U_{m,1561}$	$9998/U_{m,15533}$	$100000/U_{m,157720}$	
$0/U_{m,4}$	$9/U_{m,17}$	$100/U_{m,138}$	$1000/U_{m,1550}$	$9999/U_{m,15814}$	$100000/U_{m,158181}$	$(A4)$
$0/U_{m,4}$	$10/U_{m,28}$	$100/U_{m,166}$	$1000/U_{m,1582}$	$10000/U_{m,15758}$	$100000/U_{m,158350}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	X_{100000}	
$1/U_{m,2}$	$9/U_{m,33}$	$98/U_{m,371}$	$1000/U_{m,3635}$	$9997/U_{m,35313}$	$100000/U_{m,351336}$	
$1/U_{m,5}$	$8/U_{m,31}$	$100/U_{m,324}$	$999/U_{m,3491}$	$10000/U_{m,34993}$	$100000/U_{m,352070}$	$(A5)$
$1/U_{m,2}$	$8/U_{m,39}$	$100/U_{m,426}$	$999/U_{m,3451}$	$9991/U_{m,35051}$	$99996/U_{m,351861}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	X_{100000}	
$1/U_{m,4}$	$6/U_{m,23}$	$96/U_{m,279}$	$1000/U_{m,2494}$	$9999/U_{m,24391}$	$100000/U_{m,246772}$	
$0/U_{m,5}$	$9/U_{m,23}$	$100/U_{m,243}$	$995/U_{m,2502}$	$10000/U_{m,24543}$	$99999/U_{m,247015}$	$(A6)$
$0/U_{m,2}$	$8/U_{m,30}$	$100/U_{m,229}$	$999/U_{m,2412}$	$9999/U_{m,24629}$	$99998/U_{m,246946}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	X_{100000}	
$1/U_{m,2}$	$8/U_{m,34}$	$87/U_{m,312}$	$990/U_{m,3629}$	$9993/U_{m,36628}$	$99994/U_{m,364984}$	
$0/U_{m,3}$	$7/U_{m,23}$	$99/U_{m,385}$	$992/U_{m,3764}$	$9969/U_{m,36545}$	$99984/U_{m,365507}$	$(A7)$
$0/U_{m,3}$	$10/U_{m,35}$	$94/U_{m,323}$	$1000/U_{m,3608}$	$10000/U_{m,36541}$	$99996/U_{m,364837}$	

X_1	X_{10}	X_{100}	X_{1000}	X_{10000}	
$1/U_{m,4}$	$6/U_{m,76}$	$79/U_{m,876}$	$948/U_{m,9562}$	$9949/U_{m,98298}$	
$1/U_{m,5}$	$9/U_{m,54}$	$77/U_{m,791}$	$934/U_{m,9034}$	$9836/U_{m,98234}$	$(A8)$
$0/U_{m,3}$	$3/U_{m,36}$	$73/U_{m,769}$	$971/U_{m,9322}$	$9870/U_{m,98689}$	

In these experiments the buffer tends to remain nearly full, except in streams $A2$ and $A8$, which have the most "diversity." The value of p_s tends to be roughly sm/n.

We can summarize what we've just learned as follows:

Theorem D. *For every stream $A = (a_1, a_2, \ldots, a_m)$, the estimate X_s returned by Algorithm D when $s < |A|$ is a mixture of $m!$ random variables*

$$X_{\pi,s} \;=\; q_{\pi,s}/U_{m,p_{\pi,s}+1}, \tag{18}$$

one for each permutation π of $\{0, 1, \ldots, m-1\}$, where each permutation occurs with probability $1/m!$. Here $q_{\pi,s}$ and $p_{\pi,s}$ are integers, depending on A, with $q_{\pi,s} \le s$ and $p_{\pi,s} \ge s$. We have

$$\mathrm{E}\,X_s \;=\; \frac{1}{m!}\sum_{\pi} q_{\pi,s}\,m/p_{\pi,s} \;=\; |A|. \tag{19}$$

Proof. This result is an immediate consequence of Algorithm M′, with $p_{\pi,s}$ the value of p_s for the permutation $\pi = u_1 \ldots u_m$ in step M4′, and with $q_{\pi,s} = \sum_{a=0}^{m-1}[v_a < p_{\pi,s}]$. Equation (19), in connection with (6), simply states that the estimate is unbiased; and we know from Lemma M that it is. ▌

• (optional) **Asymptotic Analysis.** To conclude this study, I had hoped to use Theorem D to sharpen the results of Corollary T, because the actual performance of Algorithm D in practice is noticeably better than the comparatively weak theoretical guarantees that are derivable from the coarser Algorithm D′. Algorithm D is quite simple, so I believed that I'd be able to analyze its behavior without great difficulty.

Alas, after several weeks of trying, I must confess that I've failed to get much further. Indeed, I think Algorithm D may well be the simplest algorithm that lies beyond my current ability to carry out a sharp analysis! However, I shall record here some of the things that I tried, and some of the questions that have me stumped at the moment, in hopes that interested readers may be able to resolve them.

Going back to basics, we're given a stream A of length m, and a buffer size s. The stream contains exactly $n = |A|$ distinct elements, but we don't know n. Theorem D tells us that Algorithm D's estimate X_s is a random variable whose *mean* value, n, is correct.

The value of X_s is rarely an integer, except when it's 0, or when $s \ge n$ and $X_s = n$. So we certainly can't expect absolute perfection. But we'd like to know that X_s is almost always a pretty good approximation to the truth. Algorithm M′ reduces the analysis to an interesting problem about permutations.

I began by introducing some further notation. We may assume that $A = (a_1, \ldots, a_m)$ is one of the $\left\{ {m \atop n} \right\}$ restricted growth strings whose maximum element is $n-1$. Let l be the smallest index for which $a_l = n-1$. And for $l \le t \le m$, let B_t be the set $\{b_{t,0}, b_{t,1}, \ldots, b_{t,n-1}\}$, where $b_{t,k}$ is the largest index $j \le t$ such that $a_j = k$. Notice, for example, that $b_{t,a_t} = t$, and that b_{t+1} is obtained from B_t by removing $b_{t,a_{t+1}}$ and replacing it by $t+1$. For all $k \ne a_{t+1}$, we have $b_{t+1,k} = b_{t,k}$. The elements of array w in step M3′ are $\{u_{b_{t,k}} \mid 0 \le k < n\}$. The value of $q_{\pi,s}$ in (18) is the number of elements of B_m whose volatility is less than $p_{\pi,s}$.

My first goal was to study the value of $p_s = p_{\pi,s}$ in step M4′, and to show that it's fairly well concentrated about the value sm/n. I took a leisurely path to the analysis, using the concrete parameters $m = 1000000$, $n = 10000$, and $s = 100$, before attempting to resolve the general case. For these particular parameters we have $sm/n = 10000$; so I tried first to prove that p_s is unlikely to be ≤ 5000.

Let $f(p)$ be the number of ways to choose 10000 nonnegative integers less than 1000000 so that the 101st smallest of those integers is p. There are $\binom{p}{100}$ ways to choose the integers less than p, and $\binom{999999-p}{9899}$ ways to choose the integers greater than p. Hence we have

$$f(p) = \binom{p}{100}\binom{999999-p}{9899}; \qquad \text{and} \qquad g(p) = \frac{f(p)}{f(p-1)} = \frac{p}{p-100}\cdot\frac{990101-p}{1000000-p}. \tag{20}$$

Since $g(p) = 1$ when $p = p^* = 100000000/9999 \approx 10001$, we deduce that $f(p-1) \le f(p)$ when $p \le p^*$ and $f(p-1) \ge f(p)$ when $p > p^*$. In particular, the maximum $f(p)$ occurs when $p = 10001$.

Now we can show that it's quite unlikely to have $p \le 5000$. For the values decrease faster than geometrically: $f(5000-k) = f(5000)/(g(5000)g(4999)\ldots g(5001-k)) \le f(5000)/g(5000)^k$. It follows that

$$\frac{\Pr(p \le 5000)}{\Pr(p = 5000)} \le \sum_{k=0}^{5000} \frac{1}{g(5000)^k} < \frac{g(5000)}{g(5000)-1} = \frac{985101}{10001} < 100. \tag{21}$$

And $\Pr(p = 5000)$ is quite small, namely $f(5000)/\binom{1000000}{10000} \approx 10^{-12}$. So $\Pr(p \le 5000) < 10^{-10}$.

If p_s in step M4$'$ is 5000 or less, at least one of the $1000000 - l$ sets B_t will lead to an array w with 10000 nonnegative integers less than 1000000, whose 101st smallest element is at most 5000. The probability that this happens is less than 1000000 times $\Pr(p \le 5000)$, which is approximately 10^{-4}. Thus I've accomplished my objective when $s = 100$, $m = 1000000$, and $n = 10000$.

Suppose we keep those values of s and m unchanged, but reduce n from 10000 to 1000; is p_s then unlikely to be at most 50000? The answer, by essentially the same argument, is yes; it's now even less likely than it was before. And similarly we're OK when $n = 100000$ and we consider $\Pr(p_s \le 500)$.

So I examined all values of n, and found that the worst upper bound for $p_s \le \frac{1}{2}sm/n$ by this method occurs when n is approximately 7000. Good news: This extremal upper bound was only about 2% higher than the bound that worked for $n = 10000$. I was glad to see that the method had therefore turned out to be satisfactory when $s = 100$ and $m = 1000000$, and I had high hopes that it would work well also in the general case.

The probability distribution that replaces (20) in general is

$$f(p) = \binom{p}{s}\binom{m-1-p}{n-1-s}; \qquad \text{and} \qquad g(p) = \frac{f(p)}{f(p-1)} = \frac{p}{p-s} \cdot \frac{m-n+s+1-p}{m-p}. \tag{22}$$

Furthermore the "mode" is $p^* = sm/(n-1)$.

I showed this to Persi Diaconis, who immediately recognized that the probability distribution in (22) is essentially a "negative hypergeometric distribution," except that its values are shifted by s. Furthermore, the negative hypergeometric distribution is a special case of the "beta-binomial distribution," which is very pleasant. Namely, we can generate a random variable $X_{m,n,s}$ with distribution (22) in the following simple and intuitive way: (i) Set $p \leftarrow U_{n,s+1}$; (ii) flip n coins whose probability of heads is p; (iii) Set $X_{m,n,s} \leftarrow s$ plus the number of heads that turned up in step (ii).

Persi also mentioned another nice property of distribution (22): In slight disguise, it already appears in my book [8], where Pólya's famous "urn model" is described on pages 6 and 7. If we start Pólya's process with $s+1$ red balls and $n-s$ black balls, the probability that have have exactly p red balls after $m-n$ steps is precisely $f(p)/\binom{m}{n}$. Furthermore, the number of red balls is a martingale with respect to the red density ratio $r/(r+b)$; it's another way to see that the distribution has nice concentration properties.

Unfortunately I don't see how to use any of those beautiful facts to prove a good upper bound on $m \cdot \Pr(X_{m,n,s} \le (1-\epsilon)sm/n)$ that is uniform in the unknown quantity n.

Furthermore, I have only rudimentary ideas about how to account for the observed fact that the quantity $q_{\pi,s}$ in (18) tends to be very near s. Algorithm D gets into a very interesting "steady state" after the last time it changes the current value of p.

I did succeed in analyzing Algorithm D's exact behavior with respect to one interesting class of streams with $n = 2$ and $s = 1$. (See exercise 11 below.) But I must now put these questions aside and go back to work on what I *should* have been doing during the past month, namely the preparation of a sequel to [7] and [8]. I fervently hope that somebody else will be able to come up with a better analysis of Algorithm D.

I'm extremely grateful to Persi Diaconis, Svante Janson, and Kuldeep Meel for informative discussions of these issues and for correcting errors in my early drafts.

• **Exercises.** The following exercises have "rating numbers" as in *The Art of Computer Programming*.

1. [*18*] Find the exact distribution of Algorithm D's output when it's applied with $s = 1$ to the streams (a_1, a_2, a_3) for which (i) $a_1 = a_2 \neq a_3$ and (ii) $a_1 \neq a_2 = a_3$. What formulas correspond to (3) and (7) in those cases?

2. [*20*] Is it possible for Algorithm D to return the estimate 0, when $m = 5$ and $s = 2$?

3. [*HM25*] Let the random variable X be the output of Algorithm D when all m elements of the stream are distinct. We know that $\mathrm{E}\,X = m$. What is the *variance* of X, as a function of s, when $1 \le s \le m$?

4. [*20*] Would Algorithm D still be valid if step D1 said '$p \leftarrow 1/2$' instead of '$p \leftarrow 1$'?

5. [*21*] Suppose every element a_t of the input stream is an integer in the range $1 \le a_t \le n$, and let w_1, $\ldots$, w_n be any weights. Extend Algorithm D so that it returns an unbiased estimate of $\sum_{k\in A} w_k$.

6. [*HM20*] Show that the variance of $1/U_{m,\alpha m}$ is $(1-\alpha)/(\alpha^3 m) + O(1/m^2)$, for fixed $\alpha < 1$ as $m \to \infty$.

7. [*20*] True or false? Algorithm D always gives a perfectly correct answer, with absolutely no error regardless of the random numbers it uses, if and only if $s \ge |A|$.

8. [*M22*] Analyze the exact behavior of Algorithm D$'$ when it is applied with $s = 1$ to the stream of distinct elements (a, b, c). What is the probability that it outputs 0? What is the probability that it outputs 2^q?

9. [*24*] Experiment with Algorithm D$'$ on stream $A1$. What results do you get for $s = 1$, $s = 10$, $s = 100$, $s = 1000$, $s = 10000$, and $s = 100000$, corresponding to Algorithm D's (8), (9), (10), and (11)?

10. [*M22*] Given $\epsilon > 0$, and $\delta > 0$, and a stream length m, explain how to choose s so that Algorithm D's estimate is guaranteed to lie between $(1-\epsilon)|A|$ and $(1+\epsilon)|A|$, except with probability δ. Illustrate your method when $m = 1000000$ and $\delta = \epsilon = 1/10$.

11. [*M36*] Suppose the input stream $(a_1, \ldots, a_m)$ is the alternating sequence $(0, 1, 0, 1, \ldots, (1+(-1)^m)/2)$, for $m > 1$. When Algorithm M$'$ terminates, we always have $p_0 = 0$ and $p_2 = \cdots = p_{m-1} = m$, because the stream has only two distinct elements. The other value, $r = p_1$, can have any value in the range $0 < r < m$. Three different scenarios can arise at the end:

- $v_0 \ge r$ and $v_1 \ge r$. (The buffer is empty and the estimate is 0.)
- $v_{a_m} = q < r$ and $v_{1-a_m} \ge r$. (The buffer contains a_m, and the estimate is $1/U_{m,r+1}$.)
- $v_{a_m} \ge r$ and $v_{1-a_m} = q < r$. (The buffer contains a_{m-1}, and the estimate is $1/U_{m,r+1}$.)

Let $e_m(r)$, $a_m(q,r)$, and $b_m(q,r)$ be the number of permutations of $\{0, 1, \ldots, m-1\}$ that lead to each of those respective scenarios. For example, the text's analysis for $m = 3$ shows that $e_3(1) = 1$, $e_3(2) = 0$, $a_3(0,1) = a_3(0,2) = a_3(1,2) = 1$, $a_3(0,1) = 2$, and $b_3(0,2) = b_3(1,2) = 0$.

a) Let $a_m(r) = a_m(0,r)$, $b_m(r) = b_m(0,r)$. Prove that $a_m(q,r) = a_m(r)$ and $b_m(q,r) = b_m(r)$ for $0 \le q < r$. [Consequently we have $\sum_{r=1}^{m-1}\bigl(e_m(r) + r a_m(r) + r b_m(r)\bigr) = m!$.]

b) Tabulate $e_m(r)$, $a_m(r)$, and $b_m(r)$ for $1 \le r < m \le 8$. Look for patterns!

c) Explain why $a_m(1) + \cdots + a_m(m-1) = b_m(1) + \cdots + b_m(m-1) = (m-1)!$.

d) Find "simple" formulas for $e_m(r)$, $a_m(r)$, and $b_m(r)$, given m and r. *Hint:* Consider $\sum_{k=r}^{m-1} a_m(k)$.

12. [*HM41*] Continuing exercise 11, what's the asymptotic behavior of $\sum_{r=1}^{m-1} e_m(r)/m!$? (This is the probability that the estimate returned by Algorithm D for the alternating stream is zero.)

13. [*M46*] There are $\left\{{m \atop 2}\right\} = 2^{m-1} - 1$ restricted growth sequences whose maximum element is 1; hence there are $\left\{{m \atop 2}\right\}$ essentially different streams of length m that have only two distinct elements. Algorithm D does a much better job on some of these streams than on others. (For example, the result of the trivial stream $(0, 0, \ldots, 0, 1)$ is $1/U_{m,r+1}$ with probability $r/\binom{m}{2}$, for $1 \le r < m$.)

Does the stream of exercise 11 lead to the worst estimates, in some sense, of all 2-element streams?

14. [*HM30*] What is the generating function $\sum_p f(p) z^p$ for the shifted negative hypergeometric probability distribution (22)?

15. [*M20*] Suppose n is a multiple of 25. Alice makes n tosses of a $\frac12$-coin (a "fair coin"); she's unlucky if she gets fewer than $.48n$ heads. Bob makes n tosses of a $\frac14$-coin; he's unlucky if he gets fewer than $.24n$ heads. Prove that Pr(Bob is unlucky) $>$ Pr(Alice is unlucky)$/2$.

16. [*M46*] Can the result of exercise 15 be improved to Pr(Bob is unlucky) $>$ Pr(Alice is unlucky)? More generally, if $p > q$, is it more probable to have fewer than $(1-\epsilon)nq$ heads with a q-coin than $(1-\epsilon)np$ heads with a p-coin?

• **Appendix.** The procedure originally presented in [2] was the following.

Algorithm X (*Distinct element estimation*). Given an arbitrary data stream $(a_1, a_2, \ldots, a_m)$ and a buffer size $s \geq 2$ as described above, this algorithm either returns an estimate of $|A| = |\{a_1, a_2, \ldots, a_m\}|$ or reports failure. It uses a buffer B that's capable of holding up to s elements of the stream.

X1. [Initialize.] Set $t \leftarrow 0$, $p \leftarrow 1$, and $B \leftarrow \emptyset$.

X2. [Done?] Terminate if $t = m$, returning the estimate $|B|/p$.

X3. [Input a.] Set $t \leftarrow t + 1$ and $a \leftarrow a_t$, the next element of the stream.

X4. [Remove a from B.] If B contains the element a, delete it.

X5. [Maybe put a in B.] Let u be a uniform deviate, independent of all others (namely a random real number in the range $0 \leq u < 1$). If $u < p$, insert a into B.

X6. [Ensure $|B| < s$.] If $|B| = s$, do the following: Throw away each element of B with probability $1/2$; set $p \leftarrow p/2$; and terminate with failure if we still have $|B| = s$. Then go back to step X2. ▌

This algorithm is appealing at first sight; but unfortunately it's biased!

Consider, for example, the case when $s = 2$ and the input stream (a, b, c) has just $m = 3$ distinct elements. After we've input the first two elements we get to step X6 with $B = \{a, b\}$; thus $|B| = s$. The "remedy" for a full buffer has four possible outcomes, each of which will hold with probability $1/4$: Either

- $B = \emptyset$, $p = 1/2$; or
- $B = \{a\}$, $p = 1/2$; or
- $B = \{b\}$, $p = 1/2$; or
- failure has occurred.

Now consider the scenarios that are possible after Algorithm X has either input the third element, c, and terminated normally, or failed.

- $B = \emptyset$, $p = 1/2$, estimate 0; this holds with probability $1/8$.
- $B = \{a\}$, $p = 1/2$, estimate 2; this holds with probability $1/8$.
- $B = \{b\}$, $p = 1/2$, estimate 2; this holds with probability $1/8$.
- $B = \{c\}$, $p = 1/2$, estimate 2: this holds with probability $1/8$.
- $B = \emptyset$, $p = 1/4$, estimate 0; this holds with probability $1/16$.
- $B = \{a\}$, $p = 1/4$, estimate 4; this holds with probability $1/32$.
- $B = \{b\}$, $p = 1/4$, estimate 4; this holds with probability $1/32$.
- $B = \{c\}$, $p = 1/4$, estimate 4; this holds with probability $1/16$.
- Failure has occurred; this holds with probability $1/4 + 1/16$.

The expected value of the estimate, given that failure has not occurred, is therefore $\frac{5}{4}/\frac{11}{16} = \frac{20}{11}$, not 3.

Indeed, Algorithm X thinks that c is more likely to be in the stream than a or b. In general when $m = s + 1$ and the stream has distinct elements, the set B is more likely to contain the last element than any of the others, because we know that at least one of the others is missing.

However, if we change step X6 by saying "While $|B| = s$" instead of "If $|B| = s$," and if we never terminate with failure, it's not difficult to see that the resulting algorithm has become unbiased. Indeed, it is then equivalent to Algorithm D′, but with s changed to $s + 1$.

The minor slip noted here does not invalidate the proof of the main theorem in [2]. Indeed, I have essentially used the theory that was pioneered in that paper as the basis for the proof of Theorem T above.

• **References.**

[1] Cecilia R. Aragon and Raimund G. Seidel, "Randomized search trees," *IEEE Symposium on Foundations of Computer Science* **30** (1989), 540–545. Subsequently expanded into a journal article by R. Seidel and C. R. Aragon, "Randomized search trees," *Algorithmica* **16** (1996), 464–497.

[2] Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel, "Distinct elements in streams: An algorithm for the (text) book," *30th Annual European Symposium on Algorithms* (ESA 2022), 39:1–39:6. Published by Leibniz International Proceedings in Informatics, at Schloss Dagstuhl. See arXiv:2301.10191 [cs.DS] for subsequent versions.

[3] Norman L. Johnson, Adrienne W. Kemp, and Samuel Kotz, *Univariate Discrete Distributions*. Third edition (Hoboken, New Jersey: Wiley Interscience, 2005), xix + 646 pp.

[4] Donald E. Knuth, *Fundamental Algorithms*, Volume 1 of *The Art of Computer Programming*. Third edition (Reading, Massachusetts: Addison–Wesley, 1997), originally xx + 650 pp.; xx + 652 pp. since 2011.

[5] Donald E. Knuth, *Sorting and Searching*, Volume 3 of *The Art of Computer Programming*. First edition (Reading, Massachusetts: Addison–Wesley, 1973), xii + 722 pp. + foldout illustration. Second edition (Reading, Massachusetts: Addison–Wesley, 1998), originally xiv + 780 pp.; xiv + 782 pp. since 2011.

[6] Donald E. Knuth, *The Stanford GraphBase*: A Platform for Combinatorial Computing. (New York: ACM Press, 1994), viii + 576 pp.

[7] Donald E. Knuth, *Combinatorial Algorithms, Part 1*, Volume 4A of *The Art of Computer Programming* (Upper Saddle River, N. J.: Addison–Wesley, 2011), xvi + 883 pp.

[8] Donald E. Knuth, *Combinatorial Algorithms, Part 2*, Volume 4B of *The Art of Computer Programming* (Boston, Massachusetts: Addison–Wesley, 2023), xviii + 714 pp.

[9] Donald E. Knuth, `http://cs.stanford.edu/~knuth/programs/cvm-estimates.w`. (A literate program written in CWEB, May 2023.)

[10] Jean Vuillemin, "A unifying look at data structures," *Communications of the ACM* **23** (1980), 229–239.

• **Answers to the Exercises.**

1. (i) $1/U_{3,3}$ or $1/U_{3,3}$ or $1/U_{3,3}$ or $1/U_{3,2}$ or $1/U_{3,3}$ or $1/U_{3,2}$. Expected estimate $\frac{1}{6}\bigl(\frac{3}{2}+\frac{3}{2}+\frac{3}{2}+3+\frac{3}{2}+3\bigr)$. (ii) $1/U_{3,2}$ or $1/U_{3,2}$ or $0/U_{3,2}$ or $1/U_{3,3}$ or $1/U_{3,2}$ or $1/U_{3,3}$. Expected estimate $\frac{1}{6}\bigl(3+3+0+\frac{3}{2}+3+\frac{3}{2}\bigr)$.

2. Yes. For instance, assume that a_1, a_2, and a_3 are distinct but $a_4 = a_1$ and $a_5 = a_2$. Then if $u_1 < u_2 < u_3 < u_4 < u_5$, the buffer doesn't receive (a_3, u_3) at time 3; it loses (a_1, u_1) at time 4, and (a_2, u_2) at time 5.

(There are many other instances. For example, let $a_4 = a_2$, $a_5 = a_3$, and $u_2 < u_3 < u_1 < u_5 < u_4$.)

3. The final value of $|B|$ will always be s, and the final value of p will always be the $(s+1)$st smallest of the uniform deviates $\{u_1, \ldots, u_m\}$ used in step D5. Therefore X is the random variable $s/U_{m,s+1}$.

Of course $\mathrm{E}\,X = m$, by (6). We also have $\mathrm{E}\,(1/U_{m,k}^2) = k\binom{m}{k}\int_0^1 t^{k-3}(1-t)^{m-k}dt = m(m-1)/((k-1)(k-2))$. Hence $\mathrm{var}(X) = \mathrm{E}\,X^2 - (\mathrm{E}\,X)^2 = s^2m(m-1)/(s(s-1)) - m^2 = m(m-s)/(s-1)$.

4. Yes, because the proof of Lemma M would still be correct. But the estimates might be substantially less accurate. For example, if $m = 1$, it would estimate $|A| = 0$ or $|A| = 2$, each with probability 1/2.

On the other hand, if the value of p in the unmodified algorithm ever becomes $\le 1/2$, this modification to step D1 has absolutely no effect on the final result that's returned in step D2 (again because of Lemma M)! Furthermore, the final value of p will almost surely be $\le 1/2$ when $s/|A|$ is less than 1/2 and m is large,

5. When $t = m$ in step D2, return the estimate $\sum\{w_k \mid (k,u) \in B \text{ for some } u\}/p$. (This extension is valid because of Lemma M: Any given key k is in B with probability p.)

6. The exact variance of $1/U_{m,k}$ (see answer 3) is $m(m-1)/((k-1)(k-2)) - (m/(k-1))^2 = m(m-k+1)/((k-1)^2(k-2))$. Replace k by αm.

7. Let $n = |A|$. The estimate is clearly exact when $s \ge n$. But if $s < n$, the value of p_s computed by Algorithm M′ is at most the maximum of n volatilities v_a, each of which is less than m. Hence $p_s < m$, and X_s is not always equal to n.

8. Infinitely many scenarios are possible, depending on how many halvings k were performed when element b was processed and how many halvings l were performed when element c was processed: case Z_k, $p = 1/2^k$, buffer empty; case A_k, $p = 1/2^k$, buffer contains a; case B_k, $p = 1/2^k$, buffer contains b; case C_k, $p = 1/2^k$, buffer contains c. The first three occur with probability $2^{-2k} - 2^{-3k}$; the latter with probability 2^{-3k}. There also are six cases that each have $p = 1/2^{k+l}$ and occur with probability 2^{-3k-2l}: case A_kZ_l, buffer empty; case A_kA_l, buffer contains a; case A_kC_l, buffer contains c; case B_kZ_l, buffer empty; case B_kB_l, buffer contains b; case B_kC_l, buffer contains c.

So the estimate is 0 with probability $\sum_{k>0}\bigl(2^{-2k} - 2^{-3k} + 2^{1-3k}\sum_{l>0}2^{-2k}\bigr) = \frac{2}{7}$. And it's 2^q with probability $2(2^{-2q} - 2^{-3q}) + 4\sum_{k=1}^{q-1}2^{-3k-2(q-k)} = 2^{1-2q} - 2^{-3q} + 2^{2-2q} - 2^{3-3q}$.

[This exercise corresponds to the situation studied in the Appendix, after step X6 has been modified to make Algorithm X unbiased.]

9.
0, 0, 524288, 1048576, 1048576, 2097152, 2097152, 2097152, 4194304;
589824, 786432, 786432, 786432, 917504, 1048576, 1179648, 1179648, 1835008;
720896, 819200, 851968, 884736, 884736, 933888, 983040, 999424, 999424;
921600, 941056, 942080, 945152, 948224, 953344, 962560, 965632, 970752;
944000, 944256, 944640, 945280, 945792, 948480, 948736, 951936, 954112;
945376, 951568, 951600, 952048, 952496, 953392, 955456, 955872, 956096.

10. Suppose $\delta = \delta_1 + \delta_2 + \delta_3$. Corollary T tells us that it suffices to have

$$s + 1 \ge \max\Bigl\{\frac{8+4\epsilon}{\epsilon^2}\ln\Bigl(1 + \frac{1}{\delta_1}\Bigr),\ 6\ln\frac{m}{\delta_2},\ \frac{1}{\ln 2}\ln\frac{m}{\delta_3}\Bigr\}.$$

We want to choose α_1, α_2, α_3 so that the maximum is small when $\delta_1 = \alpha_1\delta$, $\delta_2 = \alpha_2\delta$, $\delta_3 = \alpha_3\delta$, and $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

For example, with the suggested parameters we want to minimize

$$\begin{aligned}&\max\{840\ln(1 + 10/\alpha_1),\ 6(16.12 - \ln\alpha_2),\ 1.443(16.12 - \ln\alpha_3)\}\\ &\quad = \max\{840\ln(1 + 10/\alpha_1),\ 96.7 - 6\ln\alpha_2,\ 23.25 - 1.443\ln\alpha_3\}\end{aligned}$$

subject to $\alpha_1 + \alpha_2 + \alpha_3 = 1$. It's pretty clear that we want α_1 to be quite near 1. Indeed, there's a good chance we can make the maximum come out to be $\lceil 840 \ln 11 \rceil = 2015$. Sure enough, if we choose $\ln(1/\alpha_2) \approx \frac{1}{6}(2015 - 96.7) \approx 319.7$ and $\ln(1/\alpha_3) \approx \ln 2(2015 - 23.25) \approx 1380.6$, so that the second and third components are 2015, we get $\alpha_1 = 1 - \alpha_2 - \alpha_3 > 1 - e^{-319}$; hence $840 \ln(1 + 10/\alpha_1) \approx 2014.23$. We therefore choose $s = 2014$.

11. (a) This follows by induction on m, considering u_m's relative order among the other u_k.

(b)

	$e_m(r)$				$a_m(r)$					$b_m(r)$			
	$r{=}1$	$r{=}2$	$r{=}3$	$r{=}4$	$r{=}1$	$r{=}2$	$r{=}3$	$r{=}4$	$r{=}5$	$r{=}1$	$r{=}2$	$r{=}3$	$r{=}4$
$m{=}2$	0	0	0	0	1	0	0	0	0	1	0	0	0
$m{=}3$	1	0	0	0	1	1	0	0	0	2	0	0	0
$m{=}4$	6	0	0	0	2	4	0	0	0	4	2	0	0
$m{=}5$	30	8	0	0	6	14	4	0	0	12	12	0	0
$m{=}6$	168	96	0	0	24	60	36	0	0	48	60	12	0
$m{=}7$	1080	864	108	0	120	312	252	36	0	240	336	144	0
$m{=}8$	7920	7680	2160	0	720	1920	1824	576	0	1440	2160	1296	144
$m{=}9$	65520	72000	30240	2304	5040	13680	14400	6624	576	10080	15840	11520	2880

Notice that $e_m(r) = 0 \Leftrightarrow r \geq \lceil m/2 \rceil$; $a_m(r) = 0 \Leftrightarrow r > \lceil m/2 \rceil$; $b_m(r) = 0 \Leftrightarrow r > \lfloor m/2 \rfloor$.

(c) The expected contribution of element a_m to the final estimate is

$$\mathrm{E}(a_m(1)/U_{m,2} + 2a_m(2)/U_{m,3} + \cdots)/m! = (a_m(1)m/1 + 2a_m(2)m/2 + \cdots)/m! = \sum_{r=1}^{m-1} a_m(r)/(m-1)!,$$

and this is 1 because the estimate is unbiased. The same argument works for $\sum b_m(r)$, which accounts for the expected contribution of the other element, $a_{m-1} = 1 - a_m$.

(d) From $\sum_{k=r}^{m-1} a_m(r) = (m-r)!(m-r)^{\underline{r-1}}$ and $\sum_{k=r}^{m-1} b_m(r) = b_m(r) = (m-r)!(m-r-1)^{\underline{r-1}}$, we know $a_m(r)$ and $b_m(r)$. Also $e_m(r) + ra_m(r) + rb_m(r) = b_{m+1}(r)$ tells us the value of $e_m(r)$.

12. (The probability is 82.8% when $m = 100$ and 87.7% when $m = 200$.) (Incidentally, the expected value of r can be shown to be $\frac{1}{2}\sqrt{\pi m} + O(1)$.)

13. (Every nonconstant stream with $m > 1$ and $s = 1$ has positive probability of outputting the estimate $1/U_{m,2}$, because there are permutations in Algorithm M$'$ with $u_t = 0$ and $u_{t+1} = 1$, for any t with $a_t \neq a_{t+1}$. And the variance of $1/U_{m,2}$ is infinite. So we can't just rank streams by the variance of their estimates. But it seems reasonable to say that one stream is better than another if it has lower probability of outputting $1/U_{m,2}$; or, if equal in that sense, lower probability of outputting $1/U_{m,3}$; and so on.)

(It may also be true that the alternating stream maximizes the probability of a zero estimate.)

14. It's $z^s F(n-m, s+1; n+1; 1-z)$, where $F(a, b; c; w)$ is the hypergeometric function $\sum_{k \geq 0} a^{\overline{k}} b^{\overline{k}} w^k/(k!\, c^{\overline{k}})$. (See formula (6.11) in [3]. Section 6.2 of that book begins with an excellent history of the beta-binomial distribution.)

15. Instead of flipping a $\frac{1}{4}$-coin, Bob can flip two fair coins, and say that he got heads if and only if they both came up heads. And we can couple Bob's first coin to Alice's.

For concreteness, condition on the number of heads, h, obtained by Alice. Suppose $n = 1000$. If $h \geq 480$, Alice is lucky. So Pr(Bob is unlucky) $\geq 0 =$ Pr(Alice is unlucky). If $h = 479$, Alice is unlucky. Bob's number of heads is the result of flipping 479 fair coins; he's unlucky if and only if this number is 239 or less. And that happens with probability 1/2.

If $h < 479$, Bob's chance of getting at most 239 coins with h fair flips is greater than 1/2.

Now sum on h. A similar argument works when $n = 25q$ for any q.

16. The answer to the first question is certainly yes, because computation shows that the actual ratios of the quantity Pr(Bob is unlucky)/ Pr(Alice is unlucky) are respectively 1.0964, 1.13609, 1.17049, 1.20227, 1.23248, 1.26169, … for $n = 25, 50, 75, 100, 125, 150, \ldots$; and they continue to increase. But is there a simple way to prove such an intuitively obvious fact rigorously?