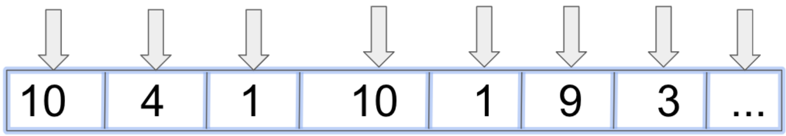
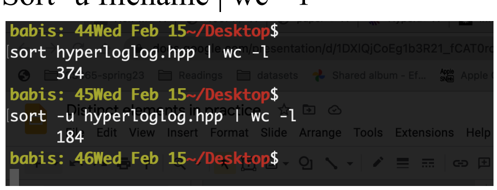
CAS CS 365
Lec 14

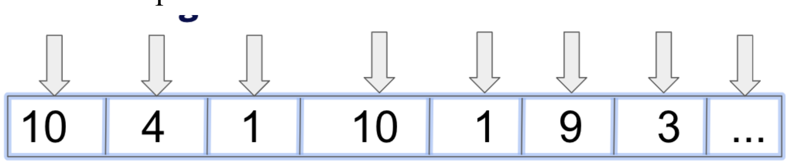
Streaming model

1. The streaming model
    a. 

    b. Input
        i. Stream of elements $\langle x_1, \dots, x_m \rangle,\ x_i \in [n]$
            1. m is the size of the stream
            2. Length of stream: $\log_2(m) + 1$
            3. As m goes huge, we even do $\log_2(\log_2(m))$
        ii. Order may be adversarial
        iii. One pass over the stream
    c. Goals
        i. Compute as accurately as possible statistics of interest
            1. Number of distinct elements appear in the stream, length of the stream, heavy hitters, etc
        ii. Key constraint: space
        iii. Ideally, also fast query and update times
2. Distinct elements
    a. A stream $\langle x_1, \dots, x_m \rangle,\ x_i \in [n]$ is received, one element from the universe[n] at a time
    b. Number of distinct elements is the number of items in the universe that appear at least once in the stream
    $|\{i : f_i > 0,\ i \in [n]\}|$ where $f_i = |\{j : x_j = i,\ j \in [m]\}|$
    c. We wish to maintain a small sketch S, whose size is independent of m, so we can return an approximate value $\tilde{F_0}$ to the true value F_0 of distinct elements
    d. There exist two broad families of algorithms for estimating F_0 in terms of the different types of guarantees they provide
3. Why not compute the distinct elements as follows?
    a. Sort -u filename | wc - 1
    b. 

c. Key constraints
   i. Limited space
   ii. One pass over the data/stream
   iii. Few operations per value (update time)
   iv. Accurate as possible
   v. Fast query time (typically at the end of the stream, but at-all-times variations exist)

4. Realistic setting
   a. Suppose we wish to court the number of distinct users who viewed "Baby shark"
      i. Say that the number of such users is 4B
      ii. If each user id is an Int64, storing those using ids in a set
          64 bits * 4 * 10^9 = 32 GBs → too big once you consider all youtube videos
      iii. Just for "baby shark", we need 32 Gbs of storage (just one video)

5. Moment Estimation problem
   a. 

   b. Distinct elements is a special case of computing p-th frequency moments
      i. $$F_p = \sum_{i=1}^{n} f_i^p$$
      ii. p=0: distinct elements (convention 0^0 = 0)
      iii. p=1: length of stream
      iv. p=2: size of self-join in DB
      v. P=inf: Here we define $F_\infty^\star = \max_i(f_i)$
      vi. In practice, we will be interested in finding the heavy hitters, elements that appear frequently in the stream

6. Missing number
   a. Let o be an arbitrary permutation of {1,...n}
      i. E.g., for n=6, o = (5,2,3,6,1,4)
   b. An element j is removed from o
      i. E.g., if j = 1, then permutation o = (5,2,3,6,4)
   c. We get to see o, but we do not know what element j was removed
   d. How much space do we need to find the missing number j?
      i. ∑ i = 1 to n of i = n(n+1)/2 ~ n^2/2 ~ n^2
      ii. log(n^2) = 2log$_2$(n)
      iii. The key idea is the use small size than n → log$_2$(n) is exponentially smaller than n

iv. S ← 0

For each x

S ← S +x

Output n(n+1)/2 - S

7. Reservoir sampling
    a. How do we get a random sample (i.e., one element), from a stream of size n?
    b. What if the size n is unknown?
        i. Reservoir sampling
    c. Algorithm: When element x_j arrives, we update our sample with value x_j with probability 1/j.
8. A useful technique
    a. Theorem: Let X be an unbiased estimator of a quantity Q. Let $\{X_{ij}\}_{i\in[t],j\in[k]}$ be a collection of independent RVs with X_{ij} distributed identically to X, where $t = O\left(\log\frac{1}{\delta}\right),\ k = O\left(\frac{Var[X]}{\epsilon^2 E[X]^2}\right)$
    b. Let $Z = median_{i\in[t]}\frac{1}{k}\sum_{j=1}^{k} X_{ij}$
    c. Then, $\Pr(|Z - Q| \geq \epsilon Q) \leq \delta$
9. Naive algorithmic solutions
    a. Algorithm 1
        i. Maintains a bitmask with n bits, one per item in the universe. When an element x appears in the stream, if BITMASK[x] = 0, set it to 1
        ii. O(n) of space complexity
    b. Algorithm 2
        i. Store the whole stream
        ii. O(mlog(n))
    c. Algorithm 3
        i. D = Dict{}
        ii. For each x in stream

            If x is in dictionary d, do nothing

            Else add x to d

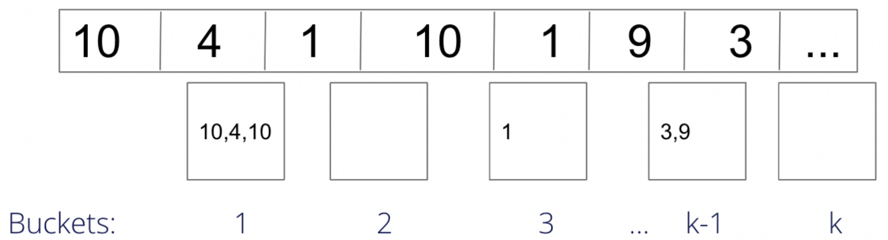
            Return size of d
        iii. Worst-case space complexity: O(min(n, mlog(n))
10. Algorithm 1: Linear counting
    a. Imagine there is a hash function such that

        H:[n] → [k]

        h(10) = h(4) = 1

        h(1) = 3

        h(3) = h(9) = k-1

b. 

c. Let Z be the number of buckets that didn't receive any item.
   i. Xi = 1 if the i-th bucket is empty
      = 0 otherwise
   ii. E[Z] = ∑E(xi)
   iii. Z = X1 + … + Xk
   iv. E(X1) = Pr(first bucket is empty)
       = (1-1/k)^F_0
   v. E[Z] = k(1-1/k)^F_0
d. Estimator: F_0 = k* ln(k/z)
e. Setting the numbers of bins k requires knowing F0, the quantity we wish to estimate
f. By first computing the variance of Z, and then applying Chebyshev, we obtain the following corollary:
   i. Setting k = F_0/12 yields a standard error of less than 1%
g. However, F_0 can be O(n), so the space can also be prohibitively large using this approach

11. Algorithm 2: Idealized F0 estimation
    a. Suppose we have access to random hash function h:U → [0,1]
    b. V ← inf
       For each x
          If h(x) < V then V ← h(x)
    c. At the end of the stream output 1/(V-1) as our estimate for the number of distinct elements
12. Minimum of n uniform random variables
    a. Let's assume that the hash function is fully random
    b. Z = min(X1,...,Xn)
    c. $X_i \sim U[0,1] \text{ for } i \in [n]$
    d. E[Z] is

$$\mathbb{E}[\mathbb{Z}] = \int_0^1 \Pr(Z > t)dt = \int_0^1 \Pr(X_1 > t)^n = \int_0^1 (1-t)^n dt = \frac{1}{n+1}$$