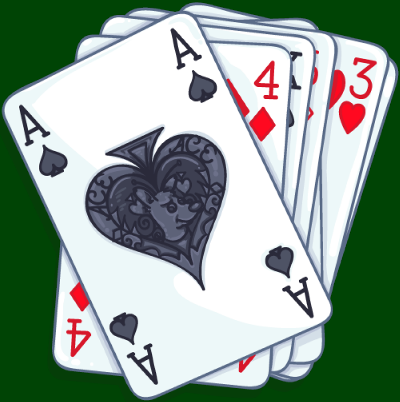
CSE 312

Foundations of Computing II

Sampling & Shuffling



Imagine you have a large data set with N items, and you'd like to get a representative sample of the data with $n \ll N$ items.

Imagine you have a large data set with N items, and you'd like to get a representative sample of the data with $n \ll N$ items.

This problem comes up a lot due to the prevalence of "big data":

- Physical measurements (from science)
- Medical data (genome sequences, time series)
- Activity data (social network activity)
- Web-server logs
- Financial data

Imagine you have a large data set with N items, and you'd like to get a representative sample of the data with $n \ll N$ items.

This problem comes up a lot due to the prevalence of "big data":

- Physical measurements (from science)
- Medical data (genome sequences, time series)
- Activity data (social network activity)
- Web-server logs
- Financial data

Some immediate concerns:

- We might have too much data to store in memory

Random Sampling

Imagine you have a large data set with N items, and you'd like to get a representative sample of the data with $n \ll N$ items.

This problem comes up a lot due to the prevalence of "big data":

- Physical measurements (from science)
- Medical data (genome sequences, time series)
- Activity data (social network activity)
- Web-server logs
- Financial data

Some immediate concerns:

- We might have too much data to store in memory
- We might not know what N is

Starting Out Samp-ly

Let's try the simplest algorithm we can think of:

In

- N – number of total records
- n – number of records we want in the sample
- S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Some questions:

- Are there any issues with SAMPLE1?

Let's try the simplest algorithm we can think of:

IN
- N – number of total records
- n – number of records we want in the sample
- S – a **stream** of the records in the data set

OUT
A sample of the N records, ideally of size n

ALG
For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Some questions:

- Are there any issues with SAMPLE1?
- Let M be the number of elements we choose. What is $\mathbb{E}[M]$?

Starting Out Samp-ly

Let's try the simplest algorithm we can think of:

In

N – number of total records
n – number of records we want in the sample
S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

```
Sample1(N, n, S):
   result = {}
   while HasNext(S):
      record = Next(S)
      if FlipCoin(n/N) == HEADS:
         result.add(record)
   return result
```

Some questions:

- Are there any issues with Sample1?
- Let M be the number of elements we choose. What is $\mathbb{E}[M]$?
- Let M be the number of elements we choose. What is $\text{Var}(M)$?

In

N – number of total records
n – number of records we want in the sample
S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Issue with SAMPLE1

Starting Out Samp-ly

In

- N – number of total records
- n – number of records we want in the sample
- S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Issue with SAMPLE1

The algorithm doesn't guarantee us a sample of exactly n elements. How can we figure out exactly how bad it does?

In

N – number of total records
n – number of records we want in the sample
S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Let M be a r.v. for the number of records in the sample. Let M_i for $1 \le i \le N$ be i.r.v.'s for the events "the ith record is selected".

In

N – number of total records

n – number of records we want in the sample

S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
    record = Next(S)
    if FlipCoin(n/N) == HEADS:
        result.add(record)
return result
```

Let M be a r.v. for the number of records in the sample. Let M_i for $1 \leq i \leq N$ be i.r.v.'s for the events "the ith record is selected". Note that

$$\mathbb{E}[M] = \mathbb{E}\left[\sum_{i=1}^{N} M_i\right] = \sum_{i=1}^{N} \mathbb{E}[M_i] = \sum_{i=1}^{N} \frac{n}{N} = n.$$

In

- N – number of total records
- n – number of records we want in the sample
- S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

Sample1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Let M be a r.v. for the number of records in the sample. Let M_i for $1 \le i \le N$ be i.r.v.'s for the events "the ith record is selected".

In

N – number of total records
n – number of records we want in the sample
S – a **stream** of the records in the data set

Out

A sample of the N records, ideally of size n

Alg

For each record, independently include in the sample with probability n/N.

SAMPLE1(N, n, S):

```
result = {}
while HasNext(S):
   record = Next(S)
   if FlipCoin(n/N) == HEADS:
      result.add(record)
return result
```

Let M be a r.v. for the number of records in the sample. Let M_i for $1 \le i \le N$ be i.r.v.'s for the events "the ith record is selected". Note that the M_i's are independent.

$$\text{Var}(M) = \text{Var}\left(\sum_{i=1}^{N} M_i\right) = \sum_{i=1}^{N} \text{Var}(M_i) = \sum_{i=1}^{N} \frac{n}{N}\left(1 - \frac{n}{N}\right) = n\left(1 - \frac{n}{N}\right).$$

Another Idea

What if we used a different probability for each record based on how many are **already selected**? That is:

Another Idea

What if we used a different probability for each record based on how many are **already selected**? That is:

> *What probability should the* $(t+1)$*st record be selected with if* m *records are already selected?*

What if we used a different probability for each record based on how many are **already selected**? That is:

> *What probability should the $(t+1)$st record be selected with if m records are already selected?*

$$\frac{\text{ways to choose remaining records including record } t+1}{\text{ways to choose remaining records}} =$$

What if we used a different probability for each record based on how many are **already selected**? That is:

> *What probability should the $(t+1)$st record be selected with if m records are already selected?*

$$\frac{\text{ways to choose remaining records including record } t+1}{\text{ways to choose remaining records}} = \frac{\binom{N-t-1}{n-m-1}}{\binom{N-t}{n-m}} = \frac{n-m}{N-t}$$

Another Idea

In

- N – number of total records
- n – number of records we want in the sample
- S – a **stream** of the records in the data set

Out

A sample of the N records of **exactly** size n

Alg

For each record, include in the sample with probability proportional to how many more records we need.

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
   record = Next(S)
   if FlipCoin((n-m)/(N-t)) == HEADS:
      result.add(record)
      m += 1 # we selected a new record
   t += 1 # we processed a new record
return result
```

Another Idea

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
   record = Next(S)
   if FlipCoin((n−m)/(N−t)) == HEADS:
      result.add(record)
      m += 1 # we selected a new record
   t += 1 # we processed a new record
return result
```

- Does the algorithm always terminate?
- What are $\mathbb{E}[m]$ and $\mathrm{Var}(m)$?
- Does the algorithm guarantee us an unbiased sample?

```
SAMPLE2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

To show SAMPLE2 terminates, we prove:

Theorem (Termination of SAMPLE2)

As long as $N \geq n$, whenever $t = N - (n - m)$, we select all remaining m records which makes $m = n$.

Proof.

Termination of Sample2

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
    record = Next(S)
    if FlipCoin((n−m)/(N−t)) == HEADS:
        result.add(record)
        m += 1 # we selected a new record
    t += 1 # we processed a new record
return result
```

To show SAMPLE2 terminates, we prove:

Theorem (Termination of SAMPLE2)

As long as $N \geq n$, whenever $t = N-(n-m)$, we select all remaining m records which makes $m = n$.

Proof.

Since $t = N-(n-m)$, $N-t = n-m$. So, $\frac{n-m}{N-t} = \frac{n-m}{n-m} = 1$ □

```
SAMPLE2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

$\mathbb{E}[m]$ and Var(m) in Sample2


```
SAMPLE2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

Note that $m = n$ at the end of the algorithm because of the while loop condition. So, $\mathbb{E}[m] = n$. Furthermore, $\mathbb{E}\left[m^2\right] = n^2$. So, Var$(m) = \mathbb{E}\left[m^2\right] - (\mathbb{E}[m])^2 = n^2 - n^2 = 0$.

Does Sample2 give an unbiased sample?

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
    record = Next(S)
    if FlipCoin((n-m)/(N-t)) == HEADS:
        result.add(record)
        m += 1 # we selected a new record
    t += 1 # we processed a new record
return result
```

Simple Case: Select First Element

The first element is selected with probability

Does Sample2 give an unbiased sample?

```
SAMPLE2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

Simple Case: Select First Element

The first element is selected with probability $\frac{n-0}{N-0} = \frac{n}{N}$, because at that point in time, the number of processed and selected records are both 0.

Does Sample2 give an unbiased sample?


```
SAMPLE2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

Simple Case: Second Element

The second element is selected with probability:

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
    record = Next(S)
    if FlipCoin((n-m)/(N-t)) == HEADS:
        result.add(record)
        m += 1 # we selected a new record
    t += 1 # we processed a new record
return result
```

Simple Case: Second Element

The second element is selected with probability:

$$\frac{n}{N}\left(\frac{n-1}{N-1}\right)+\left(1-\frac{n}{N}\right)\left(\frac{n}{N-1}\right)=\frac{n}{N}$$

- $\frac{n}{N}$: select 1st element
- $\frac{n-1}{N-1}$: select 2nd element
- $1-\frac{n}{N}$: do not select 1st element
- $\frac{n}{N-1}$: select 2nd element

Does Sample2 give an unbiased sample?

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
   record = Next(S)
   if FlipCoin((n−m)/(N−t)) == HEADS:
      result.add(record)
      m += 1 # we selected a new record
   t += 1 # we processed a new record
return result
```

Generalizing The Idea

Define $p(m,t)$ as the probability that **exactly** m records are selected from the first t.

$$p(m,t) =$$

Does Sample2 give an unbiased sample?

SAMPLE2(N, n, S):

```
result = {}
t = 0 # number of processed records
m = 0 # number of selected records
while m < n:
   record = Next(S)
   if FlipCoin((n−m)/(N−t)) == HEADS:
      result.add(record)
      m += 1 # we selected a new record
   t += 1 # we processed a new record
return result
```

Generalizing The Idea

Define $p(m,t)$ as the probability that **exactly** m records are selected from the first t.

choose m of first t

choose remaining

$$p(m,t) = \frac{\binom{t}{m}\binom{N-t}{n-m}}{\binom{N}{n}}$$

choose all

Does Sample2 give an unbiased sample?


```
Sample2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n-m)/(N-t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

Generalizing The Idea

Define $p(m,t)$ as the probability that **exactly** m records are selected from the first t.
Then:

$$\Pr(\text{element } t+1 \text{ is selected}) = \sum_{m=0}^{t} \frac{n-m}{N-t} p(m,t)$$

Does Sample2 give an unbiased sample?

```
Sample2(N, n, S):
    result = {}
    t = 0 # number of processed records
    m = 0 # number of selected records
    while m < n:
        record = Next(S)
        if FlipCoin((n−m)/(N−t)) == HEADS:
            result.add(record)
            m += 1 # we selected a new record
        t += 1 # we processed a new record
    return result
```

Generalizing The Idea

Define $p(m,t)$ as the probability that **exactly** m records are selected from the first t.
Then:

$$\Pr(\text{element } t+1 \text{ is selected}) = \sum_{m=0}^{t} \frac{n-m}{N-t} p(m,t) = \frac{\frac{(N-1)!}{(n-1)!(N-n)!}}{\binom{N}{n}} = \frac{n}{N}$$

Now What?

Our algorithm works great if...

- we know N in advance
- the records actually fit in memory

Now What?

Our algorithm works great if...

- we know N in advance
- the records actually fit in memory

How about this?

In: n – number of records we want in the sample
S – a **stream** of the records in the data set

Out: A sample of the stream of records of **exactly** size n

Alg: Keep track of a "current" total sample and repeatedly update the sample with new records based on how many we've seen.

Now What?


```
ReservoirSample(n, S):
    reservoir = [] # pool of records on disk
    chosen = [] # mapping to reservoir records
    for i = 1 to n:
        record = Next(S)
        reservoir.add(record)
        chosen[i] = i
    t = n  # number of records processed
    m = n  # size of reservoir

    while HasNext(S):
        record = Next(S)
        t += 1
        M = RollDie(t)
        if M ≤ n:
            reservoir.add(record)
            m += 1
            chosen[M] = m

    sample = {}
    for i to m:
        record = Next(reservoir)
        if i ∈ chosen:
            sample.add(record)
    return sample
```

Now What?

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

- Does each item end up in the sample with equal probability?
- What is the expected number of records in the reservoir?
- What about the variance of the size of the reservoir?

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

Equal Probabilities?

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

First n are always added at the beginning; so, to be selected, they just need to be never evicted.
That is:

$$\left(\frac{n}{n}\right)\left(\frac{n}{n+1}\right)\cdots\left(\frac{N-1}{N}\right)$$

For the remaining records, we must (1) select them, and (2) never evict them: That is:

$$\left(\frac{n}{t}\right)\left(\frac{t}{t+1}\right)\cdots\left(\frac{N-1}{N}\right)$$

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

The first n will definitely be added to the reservoir. For any remaining record, t, it will be added with probability $\frac{n}{t}$.

So, $\mathbb{E}[m] = n + \sum_{t=n+1}^{N} \frac{n}{t} = n + n(H_N - H_n) \approx n + n \ln\left(\frac{N}{n}\right)$

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

Reservoir Sampling in Three Steps

- Initialize by choosing the first n records automatically.
- For the t-th record after the first n, evict record i with probability $\frac{1}{t}$.
- Recover the final set of records from the reservoir.

Since each inclusion is independent, we can add together the individual variances. Let m_i be the i.r.v. for if the ith record is chosen for the reservoir. Note that $\text{Var}(m_i) = \frac{n}{t} - \frac{n^2}{t^2}$.

So, $\text{Var}(m) = \sum \text{Var}(m_i) = \sum_{i=n+1}^{N} \frac{n}{i} - \sum_{i=n+1}^{N} \frac{n^2}{i^2} = n(H_N - H_n) - n^2\left(\sum_{i=n+1}^{N} \frac{1}{i^2}\right)$.

Shuffling An Array

```
NaiveShuffle(A):
   for i = 1 to |A|:
      swap(A[i], A[RollDie(|A|)])
```

Shuffling An Array

```
NaiveShuffle(A):
   for i = 1 to |A|:
      swap(A[i], A[RollDie(|A|)])
```

Each choice of random numbers is equally likely. There are n^n choices for each string of numbers. However, there are only $n!$ permutations of n numbers. These numbers are not equal, and often $\frac{n^n}{n!} \notin \mathbb{Z}$. In particular, $\frac{3^3}{3!} = \frac{27}{6} \notin \mathbb{Z}$.

NaiveShuffle(A):

```
for i = 1 to |A|:
   swap(A[i], A[RollDie(|A|)])
```

FischerYatesShuffle(A):

```
for i = 1 to |A|:
   swap(A[i], A[i + RollDie(|A| − i)])
```

```
NaiveShuffle(A):
   for i = 1 to |A|:
      swap(A[i], A[RollDie(|A|)])
```

```
FischerYatesShuffle(A):
   for i = 1 to |A|:
      swap(A[i], A[i + RollDie(|A| − i)])
```

Argument for why FY works: This is just the algorithmic version of why $n!$ counts permutations! Choose the first element from all of them, the second from the remaining, etc!