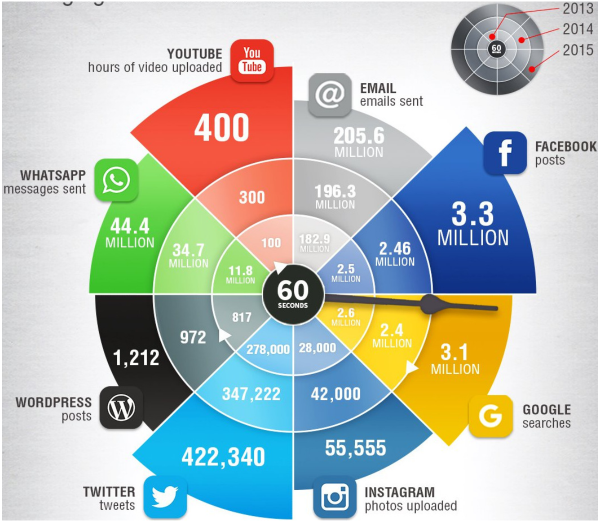
# What happens online in a single minute?




Source: Smart Insights

# Basics

Big data refers to the data sets that are so large or complex that traditional data processing applications are inadequate. Challenges in big data analysis include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy.

# Basics

Big data refers to the data sets that are so large or complex that traditional data processing applications are inadequate. Challenges in big data analysis include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy.

The four V's of Big data:

- **Volume:** Volume refers to the size of the data to be analyzed.
- **Variety:** Variety refers to the different forms of the data.
- **Veracity:** Veracity refers to the trustworthiness of the data.
- **Velocity:** Velocity is the frequency of incoming data that needs to be processed.

# Basics

Big data refers to the data sets that are so large or complex that traditional data processing applications are inadequate. Challenges in big data analysis include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy.

The four V's of Big data:

- **Volume:** Volume refers to the size of the data to be analyzed.
- **Variety:** Variety refers to the different forms of the data.
- **Veracity:** Veracity refers to the trustworthiness of the data.
- **Velocity:** Velocity is the frequency of incoming data that needs to be processed.

**Note**: The **Value** of data is also an important issue.

# Real-life scenarios

1. Mining query streams
   - Google desires to know the queries that are more frequent today in comparison with yesterday.

# Real-life scenarios

1. Mining query streams
   - Google desires to know the queries that are more frequent today in comparison with yesterday.
2. Mining click streams
   - Yahoo! wishes to know the pages receiving an unusual number of clicks in the past hour.

# *Real-life scenarios*

1. Mining query streams
   - Google desires to know the queries that are more frequent today in comparison with yesterday.
2. Mining click streams
   - Yahoo! wishes to know the pages receiving an unusual number of clicks in the past hour.
3. Monitoring IP packets
   - A switch targets to detect the denial-of-service attacks.
   - A switch wants to gather information about IP packets for optimal routing.

# *Real-life scenarios*

1. Mining query streams
   - Google desires to know the queries that are more frequent today in comparison with yesterday.
2. Mining click streams
   - Yahoo! wishes to know the pages receiving an unusual number of clicks in the past hour.
3. Monitoring IP packets
   - A switch targets to detect the denial-of-service attacks.
   - A switch wants to gather information about IP packets for optimal routing.
4. Studying dynamically changing health conditions
   - A wearable device aims to keep track of all the body parameters (that keep changing) in real-time.

# Data stream models

## Definition (Data stream model)

A *data stream model* defines an input stream $\mathcal{S} =< s_1, s_2, \cdots >$ arriving sequentially, item by item, and describes an underlying signal $S$, where $S : [1 \ldots N] \rightarrow \mathbb{R}$ is a one-dimensional function.

# Data stream models

**Definition (Data stream model)**

A *data stream model* defines an input stream $\mathcal{S} =< s_1, s_2, \cdots >$ arriving sequentially, item by item, and describes an underlying signal $S$, where $S : [1 \ldots N] \rightarrow \mathbb{R}$ is a one-dimensional function.

The data stream models can be of the following three types [Muthukrishnan, 2005]:

- Time Series Model
- Cash Register Model
- Turnstile Model

# Different data stream models

The models are classified based on the information about how the input data elements stream in.

1. Time Series Model
   - Each $s_i$ equals $S[i]$ and they appear in increasing order of $i$.
   - Observing the traffic at an IP link for each 5 min, or volume estimation of share trading in every 10 min, etc.
2. Cash Register Model
   - Perhaps the most popular data model.
   - Here $s_i$'s are increments to $S[j]$'s.
   - Monitoring IP addresses that access a web server.
3. Turnstile Model
   - This is the most general model.
   - Here $s_i$'s are updates to $S[j]$'s.
   - Monitoring the stock values of a company.

# *Processing streaming data*

A useful model for processing streaming data is to work on a window of the most recent $N$ elements received. Alternatively, we consider the elements received within a fixed time interval $T$.

# Processing streaming data

A useful model for processing streaming data is to work on a window of the most recent $N$ elements received. Alternatively, we consider the elements received within a fixed time interval $T$.

*Consider a stream of integers. Further suppose, we want to find out the average of the integers in a window of size N.*

For the first $N$ inputs, we can sum and count the integers and calculate the average. After that, for each of the new input $i$ received, add $\frac{(i-j)}{N}$ to the previous average value, where $j$ is the oldest integer in the window.

# *Streaming algorithms*

In streaming algorithms, the data is available as a stream and we would like –

- the per-item processing time
- storage and
- overall computing time

to be simultaneously $O(N, t)$, preferably $O(polylog(N, t))$, at any time instant $t$ in the data stream.

**Note:** $O(polylog(N, t))$, often written as $polylog(N, t)$, means $O((\log n)^k)$ or $O(\log^k n)$ for some $k$.

# Developments on streaming algorithms

- The limited earlier contributions before 2005 have been well reviewed in [Muthukrishnan, 2005].
- Diverse efforts were made to revisit and solve a number of problems in a streaming setting. There were studies on matrix approximation, matrix decomposition, low rank approximation, $\ell_p$ regression, etc. [Halko, 2010, Kannan, 2009, Mahoney, 2011].
- There has been an influential line of work on computing a low-rank approximation of a given matrix, starting with the works of [Frieze, 2004, Papadimitriou, 1998].
- Very recently, the $\ell_1$ and $\ell_2$ heavy eigen-hitter problems have been estimated in the streaming model in a lower dimension [Andoni, 2013]. Andoni and Huy achieved a success probability of $\frac{5}{9}$ [Andoni, 2013]. They also estimated the residual error with the same probabilistic accuracy.

# Streaming graph

**Definition (Streaming graph)**

A *streaming graph* is a simple graph on $n$ vertices $V = \{v_1, v_2, \ldots, v_n\}$ with edges $E = \{(v_i, v_j) : s_k = (i, j) \text{ for some } k \in [m]\}$, where the data items $s_k \in [n] \times [n]$ are available as an input stream $\mathcal{S} =< s_1, s_2, \ldots, s_m >$.

# Streaming graph

## *Definition (Streaming graph)*

A *streaming graph* is a simple graph on $n$ vertices $V = \{v_1, v_2, \ldots, v_n\}$ with edges $E = \{(v_i, v_j) : s_k = (i, j)$ for some $k \in [m]\}$, where the data items $s_k \in [n] \times [n]$ are available as an input stream $\mathcal{S} =< s_1, s_2, \ldots, s_m >$.

| Input stream | Interpretation in terms of a graph |
|---|---|
| (1, 2) | Two new vertices and a new edge are included |
| (1, 3) | A new vertex and a new edge are included |
| (2, 3) | A new edge is included |
| ... | ... |

# Basics

Sampling in a data stream refers to retaining only a subset of items (at most polylogarithmic sized), even though every input/update is seen.

The sampling methods can be of different types [Muthukrishnan, 2005]:

- Domain sampling
- Universe sampling
- Reservoir sampling
- Priority sampling
- Distinct sampling
- etc.

# *Finding missing numbers*

**Problem statement:** Let $\pi$ be a permutation of $\{1, \ldots, n\}$. Further, let $\pi_{-1}$ be $\pi$ with one element missing. Given the stream of elements $\pi_{-1}[i]$ in increasing order $i$, one by one, determine the missing integer.

# Finding missing numbers

**Problem statement:** Let $\pi$ be a permutation of $\{1, \ldots, n\}$. Further, let $\pi_{-1}$ be $\pi$ with one element missing. Given the stream of elements $\pi_{-1}[i]$ in increasing order $i$, one by one, determine the missing integer.

Naive approach: Store all the $n$ integers and explore. It will consume $O(n)$ time and $O(n)$ space.

# *Finding missing numbers*

**Problem statement:** Let $\pi$ be a permutation of $\{1, \ldots, n\}$. Further, let $\pi_{-1}$ be $\pi$ with one element missing. Given the stream of elements $\pi_{-1}[i]$ in increasing order $i$, one by one, determine the missing integer.

Naive approach: Store all the $n$ integers and explore. It will consume $O(n)$ time and $O(n)$ space.

Smart approach: Store the value $\frac{n(n+1)}{2} - \sum_{j \leq i} \pi_{-1}[j]$ while receiving the elements in each pass. It will consume $O(\log n)$ bits of space.

# Finding missing numbers

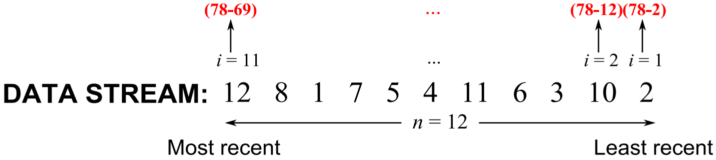
**Problem statement:** Let $\pi$ be a permutation of $\{1, \ldots, n\}$. Further, let $\pi_{-1}$ be $\pi$ with one element missing. Given the stream of elements $\pi_{-1}[i]$ in increasing order $i$, one by one, determine the missing integer.

Naive approach: Store all the $n$ integers and explore. It will consume $O(n)$ time and $O(n)$ space.

Smart approach: Store the value $\frac{n(n+1)}{2} - \sum_{j \leq i} \pi_{-1}[j]$ while receiving the elements in each pass. It will consume $O(\log n)$ bits of space.

**Note:** This was actually posed as a puzzle by Paul Erdos.

# *Finding missing numbers*



# *Finding missing numbers*

(78-69) ... (78-12)(78-2)

$i = 11$ ... $i = 2$ $i = 1$

**DATA STREAM:** 12 8 1 7 5 4 11 6 3 10 2

$\longleftarrow n = 12 \longrightarrow$

Most recent Least recent

**Note:** This consumes exactly $2 \log n$ bits of space in the worst case.

# *Finding missing numbers*

Smarter approach: For each $i$, store the parity sum of the $i^{th}$ bits of all numbers seen thus far. The final parity sum bits are the bits of the missing number. It will consume $\log n$ bits of space in the worst case.

# *Finding missing numbers*

Smarter approach: For each $i$, store the parity sum of the $i^{th}$ bits of all numbers seen thus far. The final parity sum bits are the bits of the missing number. It will consume $\log n$ bits of space in the worst case.

**Note:** A similar solution will work even if $n$ is unknown, for example by letting $n = \max_{j \leq i} \pi_{-1}[j]$ in each pass.

# Counting 1's

**Problem statement:** Given a bitstream, count the number of 1's in the last $k$ bits, where $k \leq N$ (the window size).

# Counting 1's

**Problem statement:** Given a bitstream, count the number of 1's in the last $k$ bits, where $k \leq N$ (the window size).

Naive approach: Store the most recent $N$ bits (of the stream) and count. It will consume $O(k)$ time and $O(N)$ space.

# *Counting 1's*

**Problem statement:** Given a bitstream, count the number of 1's in the last $k$ bits, where $k \leq N$ (the window size).

Naive approach: Store the most recent $N$ bits (of the stream) and count. It will consume $O(k)$ time and $O(N)$ space.

Smart approach: Store the count of 1's in blocks (of the stream) with exponentially increasing number of elements (size), in the reverse order. It will consume $O(\log^2 N)$ space.

# Counting 1's

**Problem statement:** Given a bitstream, count the number of 1's in the last $k$ bits, where $k \leq N$ (the window size).

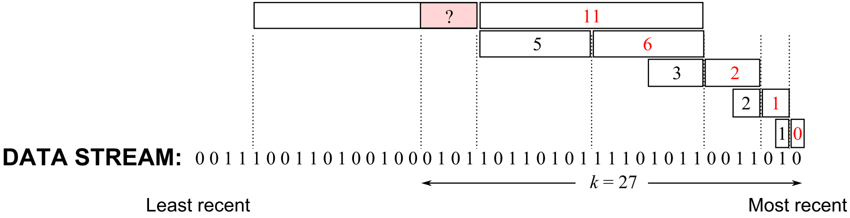
Naive approach: Store the most recent $N$ bits (of the stream) and count. It will consume $O(k)$ time and $O(N)$ space.

Smart approach: Store the count of 1's in blocks (of the stream) with exponentially increasing number of elements (size), in the reverse order. It will consume $O(\log^2 N)$ space.

**Note:** Counting the number of 1's in a stream of binary digits (bitstream) helps to compute the entropy of the stream.

# *Counting 1's*

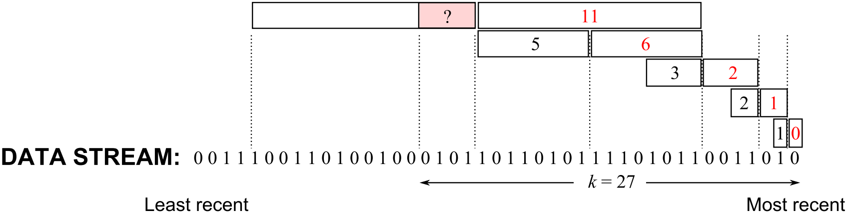
We need no more than two blocks of any size as explained below.




The count of 1's is: $(0 + 1 + 2 + 6 + 11) + \frac{7}{16}.4 \simeq 22.$

# *Counting 1's*

We need no more than two blocks of any size as explained below.




The count of 1's is: $(0 + 1 + 2 + 6 + 11) + \frac{7}{16}.4 \simeq 22$.

**<u>Note</u>:** Error in count is due to the unknown area in the most recent block. Hence, it is unbounded.

# *Counting 1's*

Smarter approach (DGIM) [Datar, 2002]: Store the count of 1's in *buckets* (of the stream) with variable number of elements but having exponentially increasing number of 1's (size), in the reverse order. It will consume $O(\log^2 N)$ space.

# Counting 1's

Smarter approach (DGIM) [Datar, 2002]: Store the count of 1's in *buckets* (of the stream) with variable number of elements but having exponentially increasing number of 1's (size), in the reverse order. It will consume $O(\log^2 N)$ space.

A *bucket* is a segment of the window having the following properties:

- The size of a bucket (number of 1's in it) is in the form of $2^i$.
- Each bucket contains the timestamp of its end bit (requires $O(\log N)$ bits) and its size (requires $O(\log \log N)$ bits).

# *Counting 1's*

Smarter approach (DGIM) [Datar, 2002]: Store the count of 1's in *buckets* (of the stream) with variable number of elements but having exponentially increasing number of 1's (size), in the reverse order. It will consume $O(\log^2 N)$ space.

A *bucket* is a segment of the window having the following properties:

- The size of a bucket (number of 1's in it) is in the form of $2^i$.
- Each bucket contains the timestamp of its end bit (requires $O(\log N)$ bits) and its size (requires $O(\log\log N)$ bits).

**Note:** Each bit in the stream has a *timestamp*, which is defined with a $(\mod N)$ function (to map everything within the window).

# *Counting 1's*

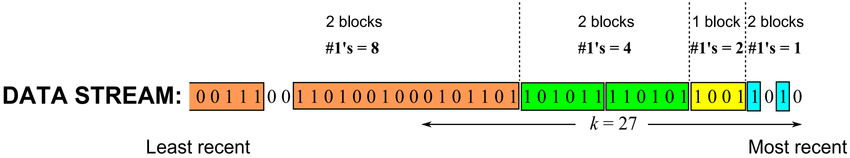
The bitstream is represented with a collection of buckets as follows:

- There can be either one or two buckets having the same size.
- Buckets are sorted by size.
- Buckets do not overlap.
- Buckets are removed as and when their end-time is more than $N$ time units in the past.

# *Counting 1's*

The bitstream is represented with a collection of buckets as follows:

- There can be either one or two buckets having the same size.
- Buckets are sorted by size.
- Buckets do not overlap.
- Buckets are removed as and when their end-time is more than $N$ time units in the past.



# Counting 1's

The buckets are updated as follows:

- When a new bit comes in, drop the oldest bucket if its end-time is prior to $N$ time units before the current time.
- No changes are required if the new bit is 0. Otherwise, do the following:
    1. Create a new bucket of size 1 containing the new bit.
    2. Define the timestamp of new bucket with the current time.
    3. Starting from $i = 0$, recursively apply the following principle: If there are now three buckets of size $2^i$, combine the oldest two to create a new bucket of size $2^{i+1}$.

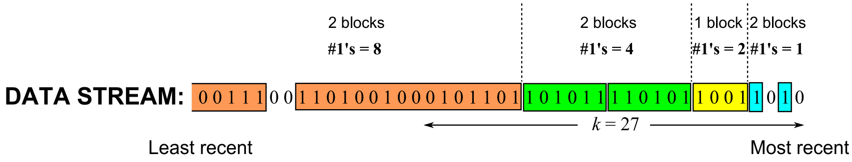
# *Counting 1's*

The buckets are updated as follows:

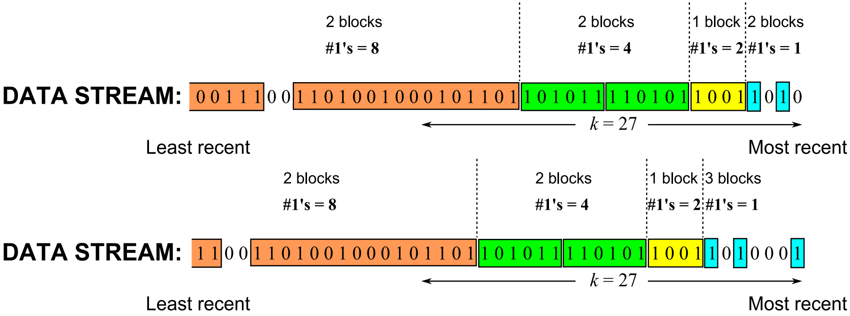
- When a new bit comes in, drop the oldest bucket if its end-time is prior to $N$ time units before the current time.
- No changes are required if the new bit is 0. Otherwise, do the following:
  1. Create a new bucket of size 1 containing the new bit.
  2. Define the timestamp of new bucket with the current time.
  3. Starting from $i = 0$, recursively apply the following principle: If there are now three buckets of size $2^i$, combine the oldest two to create a new bucket of size $2^{i+1}$.

**Note:** While combining two buckets into a new one, timestamp of the newest bucket becomes the timestamp of the new bucket.

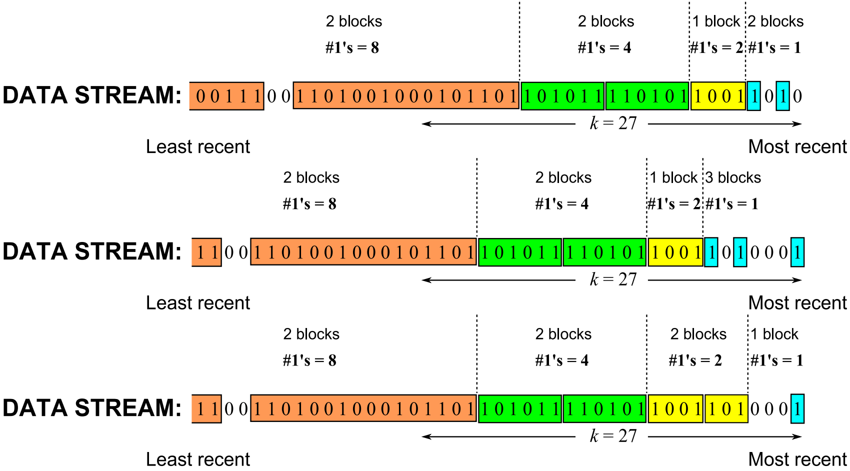
# *Counting 1's*


2 blocks
#1's = 8
2 blocks
#1's = 4
1 block
#1's = 2
2 blocks
#1's = 1
DATA STREAM:
0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
k = 27
Least recent
Most recent

# *Counting 1's*


2 blocks
#1's = 8
2 blocks
#1's = 4
1 block
#1's = 2
2 blocks
#1's = 1
DATA STREAM:
0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
k = 27
Least recent
Most recent
2 blocks
#1's = 8
2 blocks
#1's = 4
1 block
#1's = 2
3 blocks
#1's = 1
DATA STREAM:
1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1
k = 27
Least recent
Most recent

# Counting 1's


2 blocks
#1's = 8
2 blocks
#1's = 4
1 block
#1's = 2
2 blocks
#1's = 1
DATA STREAM:
0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0
k = 27
Least recent
Most recent
2 blocks
#1's = 8
2 blocks
#1's = 4
1 block
#1's = 2
3 blocks
#1's = 1
DATA STREAM:
1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1
k = 27
Least recent
Most recent
2 blocks
#1's = 8
2 blocks
#1's = 4
2 blocks
#1's = 2
1 block
#1's = 1
DATA STREAM:
1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1
k = 27
Least recent
Most recent