UNIT–3 : Mining Data Streams (PYQ Solution)

AKTU · Data Analytics (BADS601) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. Differentiate between Stream Processing and Traditional Data Processing. / How is stream processing different from traditional data processing? (Jan 2025, 2022)

Stream processing handles data continuously as it arrives, in real time or near real time, usually one record or a small window at a time; the data is unbounded and each item is seen only once. Traditional (batch) processing first stores a finite data set and then processes it all together at scheduled times, with queries run on stored data. Stream processing gives low latency and standing queries; batch gives high throughput but delayed results.

Q2. What is the role of Sliding Window in Analysis of Streaming Data? (2022)

A stream is infinite, so we cannot keep or query all of it. A sliding window keeps only the most recent N elements (or the last T time units) and moves forward as new items arrive, dropping the oldest. Queries such as moving averages, counts and trends are answered over this window. It bounds memory, keeps results up to date, and gives importance to recent data.

Q3. What is DSMS? (2023)

A Data Stream Management System (DSMS) is software that manages and queries continuous, unbounded streams of data. Unlike a DBMS, where queries run once on stored data, a DSMS runs continuous (standing) queries on arriving data and produces results incrementally, using windows and approximate summaries (sketches, samples). Examples include STREAM, Aurora, Borealis and TelegraphCQ; it is used for sensor, network and stock monitoring.

SECTION B Answers (≈250 Words Each)

Q1. Discuss the Publish/Subscribe Model of Streaming Architecture. (2022)

The **publish/subscribe (pub/sub) model** is a messaging pattern in which senders of data (**publishers**) do not send it directly to receivers (**subscribers**). Instead, publishers send messages to a **broker** under a named **topic** (or channel), and the broker delivers each message to every subscriber of that topic. It is the backbone of streaming platforms such as **Apache Kafka, RabbitMQ, Google Pub/Sub and MQTT**.



Publish/Subscribe model

Components

- **Publisher (producer):** generates events – sensors, apps, logs, click-streams – and publishes them to a topic.
- **Topic:** a named category or queue of messages; may be split into **partitions** for parallelism.
- **Broker:** stores messages (often for a retention period) and routes them to subscribers.
- **Subscriber (consumer):** registers interest in topics and receives messages as they arrive, then processes them (stream-processing engine, database, dashboard).

Working

1. A subscriber subscribes to topic "temperature".
2. A sensor publishes a reading to that topic.
3. The broker pushes (or the consumer pulls) the message to all current subscribers.
4. Each subscriber processes it independently.

Types of subscription: topic-based, content-based (filters on message content) and type-based.

Advantages

- **Decoupling:** publishers and subscribers need not know each other, run at the same time, or use the same technology.
- **Scalability:** many producers and consumers can be added; partitions share the load.
- **Fault tolerance:** messages are persisted and replicated, so consumers can replay.
- **One-to-many delivery:** one event can feed analytics, storage and alerts together.

Limitations: extra broker infrastructure, possible delivery delay, and ordering/duplicate-delivery issues that need careful handling.

Conclusion: pub/sub provides a flexible, loosely coupled and scalable way to move high-volume streaming data from sources to many processing applications.

Q2. Explain Datar-Gionis-Indyk-Motwani (DGIM) algorithm for counting ones in a window. (2023)

The **DGIM algorithm** estimates the **number of 1s in the last k bits** of a binary stream using only **$O(\log^2 N)$** bits of memory, where N is the window size, instead of storing all N bits. The error in the estimate is at most **50%**.

Idea – buckets. The window is summarised by **buckets**. Each bucket records (i) the **timestamp of its most recent 1** and (ii) its **size**, the number of 1s it holds, which is always a **power of 2** (1, 2, 4, 8 …).

Rules for the bucket representation

1. The right-hand end of every bucket is a 1.
2. Every 1 belongs to exactly one bucket; buckets do not overlap.
3. There are **at most two buckets of each size**.
4. Bucket sizes do not decrease as we move to the left (older side).
5. A bucket whose end timestamp is older than N is dropped.

Updating when a new bit arrives

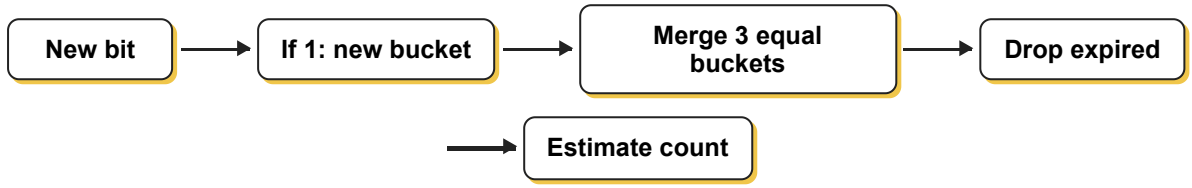
- If the bit is **0**: no change (just drop an expired bucket).
- If the bit is **1**: create a new bucket of size 1 with the current timestamp. If there are now **three** buckets of size 1, merge the **two oldest** into one bucket of size 2 (keeping the more recent timestamp). Repeat the merge for size 2, 4, … while three buckets of the same size exist.

Counting the 1s in the last k bits

- Add the sizes of all buckets completely inside the last k bits.
- For the oldest bucket that is partly inside, add **half of its size**.

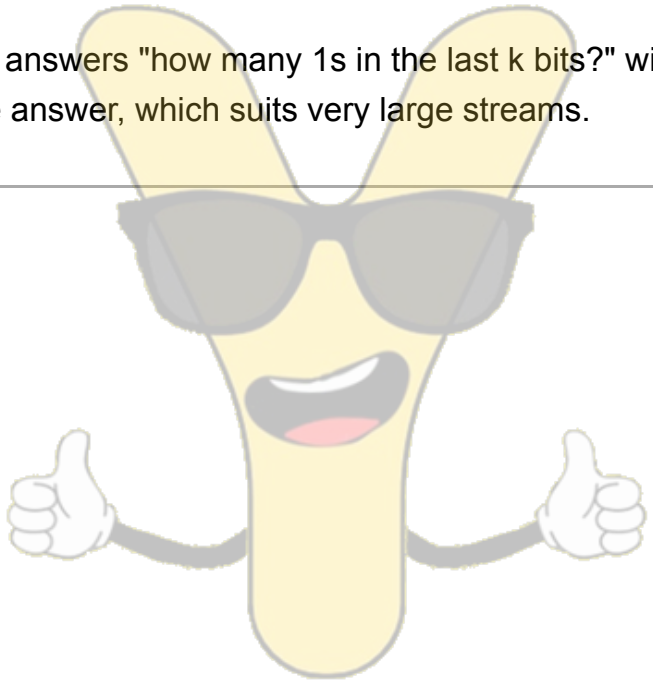
Example: buckets (sizes, newest to oldest) 1, 1, 2, 4, 4. Estimate = 1 + 1 + 2 + 4 + (4/2) = 10 ones.

Error: only the oldest bucket is uncertain, so the true count is within 50% of the estimate; using up to r buckets per size reduces the error to 1/(r − 1).



DGIM update cycle

Conclusion: DGIM answers "how many 1s in the last k bits?" with small memory and an approximate answer, which suits very large streams.



SECTION C Answers (≈500 Words Each)

Q1. Explain and apply Flajolet-Martin algorithm on the following stream of data to identify unique elements in the stream. S = {1,3,2,1,2,3,1,2,3,1} (Jan 2025) and S = 1,3,2,1,2,3,4,3,1,2,3,1 (2022). Given: h(x) = (6x+1) mod 5. (Jan 2025, 2022)

The **Flajolet-Martin (FM) algorithm** estimates the number of **distinct elements** in a stream using very little memory, without storing the elements.

Algorithm

1. Choose a hash function h that maps each element to a binary number of fixed length.
2. For every stream element a, compute h(a) and write it in binary.
3. Find the **tail length** r(a) = the number of **trailing zeros** in that binary number.
4. Keep R = the **maximum** tail length seen over the whole stream.
5. Estimate the number of distinct elements as **2^R**.

(Intuition: the chance that a hash value ends in at least r zeros is 1/2^r, so if the longest tail is R, about 2^R different values were probably hashed.)

Step 1 – Hash each distinct value with h(x) = (6x + 1) mod 5

x	6x + 1	mod 5 = h(x)	Binary	Trailing zeros
1	7	2	10	1
2	13	3	11	0
3	19	4	100	2
4	25	0	00000 (all zero)	taken as 0 (value 0 is ignored)

The same x always gives the same hash, so repeated elements do not change R.

Part 1 – S = {1,3,2,1,2,3,1,2,3,1} (Jan 2025)
Elements appearing: 1, 3, 2 only.

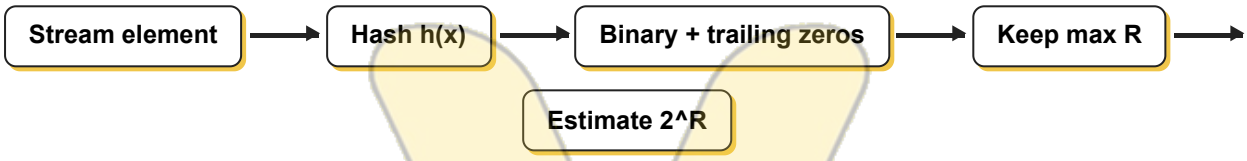
- Tail lengths seen: 1 (for 1), 2 (for 3), 0 (for 2).
- R = max(1, 2, 0) = **2**.
- Estimate of distinct elements = 2^2 = **4**.

- Actual distinct elements = 3 (namely 1, 2, 3). The estimate 4 is close; FM gives an approximation.

Part 2 – S = 1,3,2,1,2,3,4,3,1,2,3,1 (2022)
Elements appearing: 1, 3, 2, 4.

- Tail lengths: 1 → 1, 3 → 2, 2 → 0, 4 → h = 0, ignored (0 trailing zeros counted).
- R = max = **2**.
- Estimate = 2^2 = **4**.
- Actual distinct elements = 4 (1, 2, 3, 4). The estimate matches exactly.

Improving accuracy: a single hash function can be off by a factor of 2. Use many hash functions, split them into groups, take the **average within a group** and the **median across groups** of the estimates 2^R, which removes the effect of outliers.



Flajolet-Martin algorithm

Memory: only R (a small integer, O(log log n) bits) is stored per hash function.

Conclusion: for both streams, R = 2 and the FM estimate is 4 distinct elements; the algorithm counts unique items in one pass with tiny space.

Q2. Discuss the concept of filtering in Data Stream Processing. Explain Bloom Filtering in detail. (Jan 2025)

Filtering in stream processing means **selecting only those stream elements that satisfy a condition** (a property or membership in a set) and discarding the rest, so that later stages deal with less data. Examples: pass only e-mails from known addresses, keep only tweets with certain keywords, accept only URLs not already crawled. When the set of allowed items is very large (millions), storing it in memory and checking each element is costly, so an approximate technique such as the **Bloom filter** is used.

Bloom filter

A Bloom filter is a **space-efficient probabilistic data structure** that tests whether an element is a **member of a set**. It may give **false positives** (says "present" when absent) but **never false negatives** (if it says "absent" it is certainly absent).

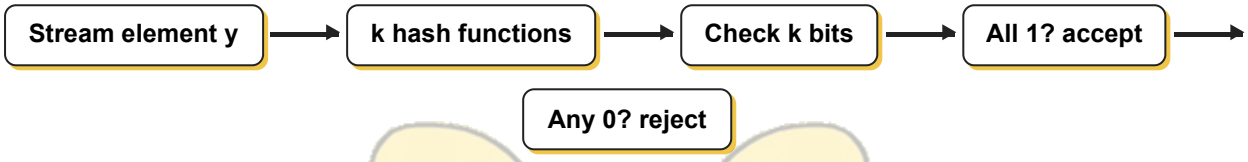
Components

- A bit array of **n bits**, all initially 0.
- **k independent hash functions** h_1 … h_k, each mapping an element to a position 0 … n − 1.
- A set S of **m** keys to be accepted.

Building (insertion): for each key x in S, set the bits at positions $h_1(x)$, …, h_k(x) to 1.

Lookup (filtering): for a stream element y, compute $h_1(y)$, …, h_k(y).

- If **all** those bits are 1 → accept (probably in S).
- If **any** bit is 0 → reject (definitely not in S).



Bloom filter lookup

Example: n = 10, k = 2. Insert "cat": h_1 = 1, h_2 = 4 → bits 1, 4 set. Insert "dog": h_1 = 4, h_2 = 7 → bits 4, 7 set. Array = 0100100100 (positions 1, 4, 7). Check "cow": h_1 = 1, h_2 = 7 → both 1 → accepted, a **false positive**. Check "pig": h_1 = 2 → bit is 0 → rejected correctly.

False-positive probability: the probability that a given bit is still 0 after inserting m keys is (1 − 1/n)^(km) ≈ e^(−km/n). So

P(false positive) ≈ (1 − e^(−km/n))^k

The best number of hash functions is k ≈ (n/m) ln 2, which gives about 10 bits per key for a 1% error.

Advantages: very fast O(k) operations, small memory, no false negatives, easy to build in parallel.

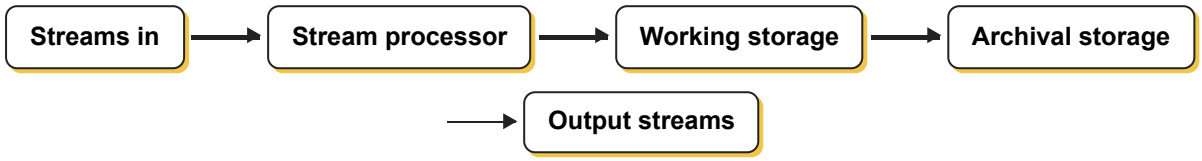
Limitations: false positives, elements cannot be deleted (use a counting Bloom filter), and the size must be planned in advance.

Uses: spam filtering, web crawlers, database look-ups (Cassandra, HBase), safe-browsing checks, cache filtering.

Conclusion: Bloom filtering lets a stream processor check membership in a huge set in constant time and tiny memory, at the cost of a small, controllable false-positive rate.

Q3. Discuss the Components of a General Stream Processing Model. List few sources of Streaming Data. (2022, 2023)

A **stream processing model** describes how continuous, unbounded data is received, processed and answered. Because the stream is fast and cannot be stored completely, the system works with **limited memory** and answers queries in one pass.



General stream processing model

Components

1. **Input streams:** one or more streams of tuples arriving at different rates and in no controlled order; the system cannot slow them down.
2. **Stream processor:** the engine that runs **standing (continuous) queries** and ad-hoc queries on the arriving data. It applies filtering, windowing, sampling, aggregation and mining algorithms.
3. **Limited working storage:** fast main memory (or disk) holding summaries – windows, counters, sketches, samples – needed to answer queries. It is too small to hold the entire stream.
4. **Archival storage:** large, slow store (disk, tape, data lake) where the raw stream may be archived; it cannot be used to answer queries in real time.
5. **Query / standing-query layer:** users register queries that are evaluated continuously (e.g. "report the maximum temperature in the last 10 minutes").
6. **Output streams / results:** the answers, alerts or aggregated streams passed to dashboards, applications or other processors.

Key properties: continuous arrival, one-pass processing, real-time response, approximate answers and bounded memory.

Sources of streaming data

Source	Example data
Sensors and IoT	Temperature, GPS, smart meters, wearables
Web click-streams	Page views, search queries, e-commerce clicks
Social media	Tweets, posts, likes, comments
Financial markets	Stock tickers, trades, currency rates
Network traffic	Packets, IP logs, intrusion alerts
Server and application logs	Error logs, transaction logs
Satellite and image streams	Weather, surveillance video
Telecom	Call records, mobile data usage

Conclusion: a general stream processing model consists of input streams, a processor with limited memory, an archive, and output streams, and it is built to give fast, approximate answers on never-ending data.

Q4. Explain Bernoulli sampling with its algorithm. (2023)

Sampling selects a small, representative subset of a stream so that queries can be answered approximately with limited memory. **Bernoulli sampling** is the simplest scheme: every element of the stream is **included in the sample independently with the same probability p** ($0 < p < 1$), like tossing a biased coin for each element – heads (probability p) means keep, tails (probability $1 - p$) means discard.

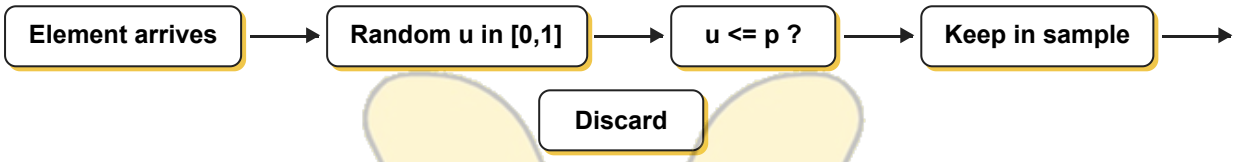
Properties

- Each element is decided on its own, so the method works in **one pass** and needs no knowledge of the stream length.
- The sample size is **random**: for a stream of N elements it follows a Binomial(N, p) distribution, with expected size **N × p** and variance $N\,p\,(1 - p)$.
- The probability that a particular element is in the sample is exactly p; the elements are independent.
- Estimates are scaled up: an aggregate computed on the sample is divided by p. For example, if the sample count of a value is c, the estimated count in the stream is c / p.

Algorithm

```
Input : stream x1, x2, x3, ...   and probability p
Output: sample S
S = empty
for each arriving element xi:
    generate a random number u uniformly in [0, 1]
    if u <= p:
        add xi to S      // keep
    else:
        discard xi       // skip
return S
```

Example: p = 0.1 on a stream of 1,000 searches. About 100 are kept. If 7 sampled searches contain the word "AI", the estimated total in the whole stream is 7 / 0.1 = 70.



Bernoulli sampling

Advantages: very simple, constant work per element, unbiased estimates, easy to run in parallel and on distributed streams.

Limitations: the sample size is not fixed and may grow without bound for long streams; to keep a fixed size, **reservoir sampling** (replace a random item with probability k/i for the i-th element) is used instead. Bernoulli sampling also does not guarantee that rare items are present.

Comparison:

Basis	Bernoulli sampling	Reservoir sampling
Sample size	Random, about N·p	Fixed k
Need N in advance	No	No
Memory	Grows with stream	Constant

Conclusion: Bernoulli sampling keeps each stream element with a fixed probability p, giving a simple and unbiased sample whose size is about p times the stream length.
