Question Bank:

1. Explain applications of data streams with examples.

   Applications of data streams include:

   a. Fraud detection: Data streams can be used to detect fraudulent transactions in real-time. For example, a credit card company can use streaming data from all transactions to identify suspicious transactions, such as those that are made from unusual locations or that exceed a certain spending limit.
   b. Real-time monitoring: Data streams can be used to monitor systems and processes in real-time. For example, a company can use streaming data from its IT systems to identify and respond to outages and performance problems.
   c. Predictive analytics: Data streams can be used to predict future events based on historical data. For example, a retailer can use streaming data from its sales transactions to predict demand for products and optimize inventory levels.
   d. Personalized recommendations: Data streams can be used to provide personalized recommendations to users in real-time. For example, a streaming music service can use streaming data from a user's listening history to recommend new songs.
   e. Real-time optimization: Data streams can be used to optimize systems and processes in real-time. For example, a traffic management system can use streaming data from traffic sensors to adjust traffic lights and optimize traffic flow.

2. Elaborate issues in data stream query processing.

   Data stream query processing is the process of continuously processing an unbounded stream of data to answer queries in real time. This presents a number of challenges, including:

   - Unbounded memory requirements: Data streams are potentially unbounded in size, so the amount of storage required to compute an exact answer to a data stream query may also grow without bound. This is in contrast to batch processing, where the dataset is known in advance and can be stored in memory.
   - Limited processing time: Data stream query processing algorithms need to be able to process each data item in real

time, even if the stream is very high-volume. This means that algorithms need to be efficient and avoid unnecessary computation.

- Approximate query answering: In some cases, it may be impossible to produce an exact answer to a data stream query due to the constraints of memory and processing time. However, it may be possible to produce an approximate answer that is still useful.
- Fault tolerance: Data stream processing systems need to be fault-tolerant, meaning that they should be able to continue operating even if some components fail. This is important because data streams are often critical to business operations.
- Handling out-of-order data: Data streams may arrive out of order, which can make it difficult to process them accurately.
- Dealing with noise and outliers: Data streams may contain noise and outliers, which can also make it difficult to process them accurately.
- Supporting sliding window queries: Sliding window queries are common in data stream processing, but they can be challenging to implement efficiently. Sliding window queries operate on a subset of the data stream that is constantly being updated as new data arrives and old data is discarded.
- Supporting continuous queries: Continuous queries are evaluated over the entire data stream, not just over a single window of data. This can be challenging to implement efficiently, especially for complex queries.

3. Explain the sliding window problem with the help of an example.

The sliding window technique is a data processing approach that analyzes a stream of data over a fixed period of time. The window is then shifted forward to analyze the next period of data, and so on. This allows for real-time or near-real-time analysis of streaming data.

The sliding window technique is particularly useful for big data analytics, as it allows for efficient processing of large datasets. For example, a sliding window can be used to track the number of active users on a website over the past hour, or to identify the most popular products in an online store over the past week.

Example

Consider the following example:

A company wants to track the number of active users on its website over the past hour. The company has a stream of data that shows the time at which each user logs in and logs out.

To solve this problem using the sliding window technique, the company can create a sliding window of 60 minutes. The window will initially contain the data for the first 60 minutes of the day. As new data arrives, the window will be slid forward by one minute, and the oldest data will be removed from the window.

At any given time, the window will contain the data for the most recent 60 minutes. The company can then calculate the number of active users by counting the number of users who are logged in during that time period.

4. Explain DGIM algorithm for counting ones in stream with given problem N=24 and data set is 101011000101111011001011011

DGIM algorithm (*Datar-Gionis-Indyk-Motwani Algorithm)*

Designed to find the number 1's in a data set. This algorithm uses $O(\log^2 N)$ bits to represent a window of N bit, allows to estimate the number of 1's in the window with and error of no more than 50%.

So this algorithm gives a 50% precise answer.

In DGIM algorithm, each bit that arrives has a timestamp, for the position at which it arrives. if the first bit has a timestamp 1, the second bit has a timestamp 2 and so on.. the positions are recognized with the window size N (the window sizes are usually taken as a multiple of 2).The windows are divided into buckets consisting of 1's and 0's.

RULES FOR FORMING THE BUCKETS:

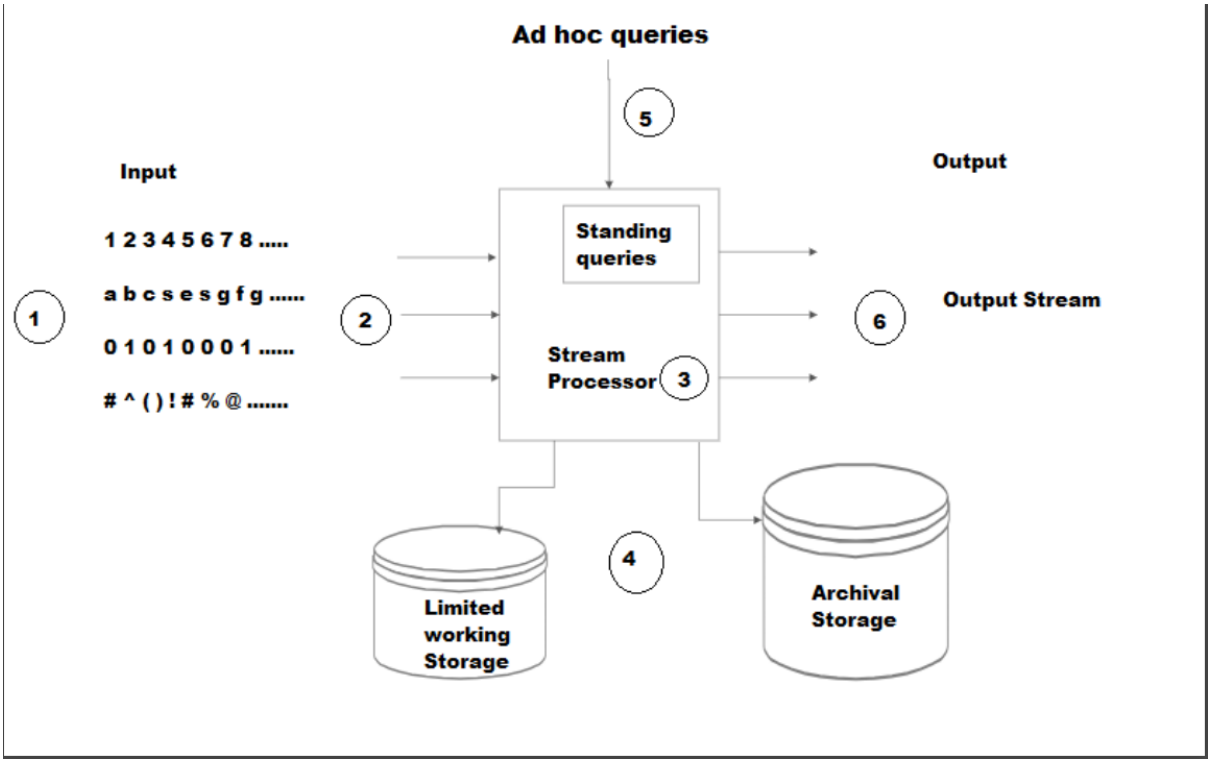
1. The right side of the bucket should always start with 1. (if it starts with a 0,it is to be neglected) E.g. · 1001011 → a bucket of size 4 ,having four 1's and starting with 1 on it's right end.
2. Every bucket should have at least one 1, else no bucket can be formed.
3. All buckets should be in powers of 2.
4. The buckets cannot decrease in size as we move to the left. (move in increasing order towards left)

10101100010111 (8) - 1100101 (4) - 101 (2) - 1 (1)

5. How bloom filters are useful for big data analytics explain with example.
   a. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set.
   b. It works by using a bit array and a set of hash functions to map elements to the bit array.
   c. When an element is added to the Bloom filter, all of the corresponding bits in the bit array are set to 1. To test whether an element is a member of the Bloom filter, all of the corresponding bits in the bit array are checked.
   d. If any of the bits are 0, then the element is definitely not a member of the Bloom filter. If all of the bits are 1, then the element may or may not be a member of the Bloom filter.
   e. Bloom filters are probabilistic because there is a small chance of false positives. A false positive occurs when the Bloom filter indicates that an element is a member of the set, even though it is not.
   f. The probability of a false positive increases as the number of elements in the set increases. However, the probability of a false negative is always 0. A false negative occurs when the Bloom filter indicates that an element is not a member of the set, even though it is.
   g. Example : A company that sells products online could use a bloom filter to store the IDs of all of the products that have been purchased by a particular customer. Then, when the customer visits the company's website, the company could use the bloom filter to quickly identify the products that the customer is most likely to be interested in. This would allow the company to personalize the customer's experience and recommend products that the customer is more likely to buy.

6. With the help of a diagram explain the data stream management system(DSMS).

DSMS is nothing but a software application just like DBMS (database management system) but it involves processing and management of continuously flowing data streams rather than static data like excel or pdf or other files. It is generally used to deal with data streams from various sources which includes sensor data, social media field, financial report etc.
DSMS processes 2 types of queries: standard queries and ad hoc queries.



DSMS consists of various layer which are dedicated to perform particular operation which are as follows:

1. Data source Layer

The first layer of DSMS is data source layer and as the name suggests, it comprises of all data sources which includes sensors, social media feeds, financial market, stock markets etc. In this layer , capturing and parsing of data takes place. Basically it is the layer which collects the data.

2. Data Ingestion Layer

This layer acts as a bridge between the data source layer and processing layer. The main purpose of this layer is to handle the flow of data i.e., data flow control, data buffering and data routing.

3. Processing Layer

This layer can be considered as the heart of DSMS architecture; it is a functional layer of DSMS applications. It processes the data streams in real time. To perform processing it uses processing engines like Apache flink or Apache storm etc. The main function of this layer is to filter, transform, aggregate and enrich the data stream. This can be done by deriving insights and detecting patterns in the data.

4. Storage Layer

Once data is processed we need to store the processed data in any storage unit. Storage layer consists of various storage like NoSQL database, distributed database etc., It helps to ensure data durability and availability of data in case of system failure.

5. Querying Layer

This layer supports 2 types of queries: ad hoc query and standard query. This layer provides the tools which can be used for querying and analyzing the stored data stream. It also has SQL like query languages or programming API.

6. Visualization and Reporting Layer

This layer provides tools to perform visualization like bar-chart, pie-chart, histogram etc., On the basis of this visualization , it also helps to generate the report for analysis.

7. Integration Layer

This layer is responsible for integrating DSMS applications with traditional systems, business intelligence tools, data warehouses, ML applications, NLP applications.

7. What are the challenges of querying on large data stream?

   Data stream query processing is the process of continuously processing an unbounded stream of data to answer queries in real time. This presents a number of challenges, including:

   a. Unbounded memory requirements: Data streams are potentially unbounded in size, so the amount of storage required to compute an exact answer to a data stream query may also grow without bound. This is in contrast to batch processing, where the dataset is known in advance and can be stored in memory.
   b. Limited processing time: Data stream query processing algorithms need to be able to process each data item in real time, even if the stream is very high-volume. This means that algorithms need to be efficient and avoid unnecessary computation.
   c. Approximate query answering: In some cases, it may be impossible to produce an exact answer to a data stream query due to the constraints of memory and processing time. However, it may be possible to produce an approximate answer that is still useful.
   d. Fault tolerance: Data stream processing systems need to be fault-tolerant, meaning that they should be able to continue operating even if some components fail. This is important because data streams are often critical to business operations.
   e. Handling out-of-order data: Data streams may arrive out of order, which can make it difficult to process them accurately.
   f. Dealing with noise and outliers: Data streams may contain noise and outliers, which can also make it difficult to process them accurately.
   g. Supporting sliding window queries: Sliding window queries are common in data stream processing, but they can be challenging to implement efficiently. Sliding window queries operate on a subset of the data stream that is constantly being updated as new data arrives and old data is discarded.
   h. Supporting continuous queries: Continuous queries are evaluated over the entire data stream, not just over a single

window of data. This can be challenging to implement efficiently, especially for complex queries.

8. Suppose the stream is 1,3,2,1,2,3,4,3,1,2,3,1 let h(x)=6x+1 mod 5 show how the Flajolet-Martin algorithm will estimate the number of distinct elements in this stream.

Flajolet-Martin algorithm approximates the number of unique objects in a stream or a database in one pass. If the stream contains n elements with m of them unique, this algorithm runs in O(n) time and needs O(log(m)) memory.

The Flajolet-Martin algorithm for estimating the cardinality of a multiset is as follows:

1. Initialize a bit vector BITMAP to be of length L, such that 2^L>n, the number of elements in the stream. Usually a 64-bit vector is sufficient since 2^64 is quite large for most purposes.
2. The i-th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in at least i trailing zeroes. So initialize each bit to 0.
3. For each element x in the stream:
    1. Calculate the index r(x) of the longest trailing run of zeroes in the binary representation of the hash function value of x.
    2. Set the r(x)-th bit of BITMAP to 1.
4. Let R be the maximum value of r(x) for all x in the stream.
5. Estimate the cardinality of the stream as 2^R.

The Flajolet-Martin algorithm is a probabilistic algorithm, meaning that it does not give an exact answer. However, it is very efficient and can be used to estimate the cardinality of very large streams.

Data	h(x) = 6x+1 mod 5	Reminder	Binary bit-String	Tail Length
1	h(x) = 6 (1) +1 mod 5	2	010	1
3	h(x) = 6 (3) +1 mod 5	4	100	2
2	h(x) = 6 (2) +1 mod 5	3	011	0
1	h(x) = 6 (1) +1 mod 5	2	010	1
2	h(x) = 6 (2) +1 mod 5	3	011	0
3	h(x) = 6 (3) +1 mod 5	4	100	2
4	h(x) = 6 (4) +1 mod 5	0	000	0
3	h(x) = 6 (3) +1 mod 5	4	100	2
1	h(x) = 6 (1) +1 mod 5	2	010	1
2	h(x) = 6 (2) +1 mod 5	3	011	0
3	h(x) = 6 (3) +1 mod 5	4	100	2
1	h(x) = 6 (1) +1 mod 5	2	010	1

Tail Length = {1,2,0,1,0,2,0,2,1,0,2,1}
R = max (Tail Length) = 2
Estimation of m = 2 ^ R = 2 ^ 2 = 4
Hence we have 4 distinct elements 1,2,3,4

9. How recommendation is done based on properties of the product?Elaborate with a suitable example.
    a. Product recommendation systems based on properties work by comparing the properties of products that a user has previously interacted with (e.g., purchased, viewed, rated) to the properties of other products in the catalog. The system then recommends products that are similar to the products that the user has previously interacted with.
    b. For example, a product recommendation system for an online retailer might consider the following product properties:
    c. Category (e.g., books, music, clothes, electronics)
        i. Brand
        ii. Price
        iii. Rating
        iv. Number of reviews
        v. Features
        vi. Tags

d. The system might also consider user demographics, such as age, gender, and location.
e. To generate recommendations, the system would first identify the products that the user has previously interacted with. Then, it would compare the properties of those products to the properties of other products in the catalog. The system would then recommend products that are similar to the products that the user has previously interacted with.
f. For example, if a user has previously purchased books in the science fiction genre, the system might recommend other science fiction books. Or, if a user has previously purchased a specific brand of smartphone, the system might recommend other smartphones from the same brand.
g. Product recommendation systems based on properties are often used in conjunction with other recommendation techniques, such as collaborative filtering. Collaborative filtering works by recommending products to users based on the ratings and purchase history of other users with similar tastes.
h. Different algorithms such as Cosine similarity, Pearson Correlation, Jaccard index, K-Nearest Neighbor are used for recommendation of product based on properties.

10. What is jaccard distance and cosine distance in collaborative filtering?

Jaccard distance and cosine distance are two similarity measures that can be used in collaborative filtering to identify users with similar tastes.

Jaccard distance is a measure of the similarity between two sets. It is calculated by dividing the number of elements that are in both sets by the total number of elements in both sets. The Jaccard distance between two users is calculated by comparing the sets of items that the two users have rated. A Jaccard distance of 0 indicates that the two users have no items in common, while a Jaccard distance of 1 indicates that the two users have rated all of the same items.

Formula for Jaccard distance:

```
Jaccard distance = (A ∩ B) / (A ∪ B)
```

where A and B are the sets of items that the two users have rated.

Cosine distance is a measure of the similarity between two vectors. It is calculated by taking the dot product of the two vectors and dividing by the product of their magnitudes. The cosine distance between two users is calculated by comparing the vectors of ratings that the two users have given to items. A cosine distance of 0 indicates that the two users have identical rating vectors, while a cosine distance of 1 indicates that the two users have completely opposite rating vectors.

Formula for cosine distance:

```
Cosine distance = 1 - (A · B) / (|A| |B|)
```

where A and B are the vectors of ratings that the two users have given to items.

Both Jaccard distance and cosine distance can be used to identify users with similar tastes in collaborative filtering. However, they have different strengths and weaknesses.

Jaccard distance is a good measure of similarity for users who have rated many of the same items. However, it is not as good of a measure of similarity for users who have rated different sets of items.

Cosine distance is a good measure of similarity for users who have rated different sets of items. However, it is not as good of a measure of similarity for users who have given very different ratings to the items that they have rated.

Collaborative filtering systems often use a weighted combination of Jaccard distance and cosine distance to identify users with similar tastes. The weights are chosen to optimize the performance of the system on a particular dataset.

11. A bloom filter with m=1000 cells is used to store information about n=100 items,using k=4 hash functions.Calculate the false positive probability of this instance.will the performance improve by increasing the number of hash function from 4 to 5.Explain your answer.

    The false positive probability of a Bloom filter can be calculated using

the following formula:

```
P_f = (1 - e^(-kn/m))^k
```

where:

- Pf is the false positive probability
- k is the number of hash functions
- n is the number of items stored in the Bloom filter
- m is the size of the Bloom filter

For the given instance, we have:

- m=1000
- n=100
- k=4

Plugging these values into the formula, we get:

```
P_f = (1 - e^(-4*100/1000))^4 = 0.3298
```

Therefore, the false positive probability of this Bloom filter is 0.3298.

Increasing the number of hash functions from 4 to 5 will reduce the false positive probability. This is because each additional hash function makes it less likely that a false positive will occur.

The following table shows the false positive probability for different values of k:

k	Pf
4	0.3298
5	0.1274
6	0.0503

7	0.0199
8	0.0079

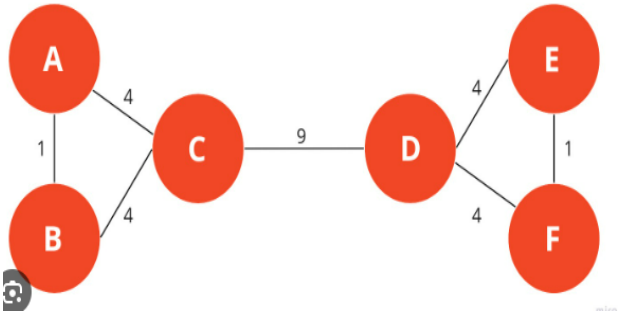
As you can see, the false positive probability decreases significantly as the number of hash functions increases.

However, it is important to note that increasing the number of hash functions also increases the overhead of the Bloom filter. This is because each additional hash function requires more computation.

Therefore, it is important to choose a value of k that balances the need for a low false positive probability with the need for a low overhead.

In general, it is recommended to use at least 5 hash functions for Bloom filters. This will provide a good balance between false positive probability and overhead.

12. Explain Girvan -Newman algorithm with the help of given example.



Girvan-Newman algorithm

The Girvan-Newman algorithm is a hierarchical method for detecting communities in complex networks. It works by iteratively removing the edges with the highest betweenness centrality. Betweenness centrality is a measure of how important a node or edge is in a network, and it is calculated by counting the number of shortest paths between pairs of nodes that pass through the node or edge.

Steps of the Girvan-Newman algorithm

1. Calculate the betweenness centrality of all edges in the network.
2. Remove the edge with the highest betweenness centrality.
3. Recalculate the betweenness centrality of all remaining edges.
4. Repeat steps 2 and 3 until the network is disconnected.

5. The connected components of the disconnected network are the communities.

https://networkx.guide/algorithms/communit-detection/girvan-newman/
y

13. Enlist and explain different functions used for manipulating and processing data in R.

Here is a list of some of the most commonly used functions for manipulating and processing data in R, along with examples:

Function	Description	Example
select()	Selects a subset of columns from a data frame.	

Code snippet

```
# Create a data frame
df <- data.frame(x = c(1, 2, 3, 4, 5), y = c(6, 7, 8, 9,
10), z = c(11, 12, 13, 14, 15))

# Select the x and y columns
df_selected <- select(df, x, y)

# Print the selected data frame
print(df_selected)

Output:
  x y
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
```

filter() | Filters a data frame based on a condition. |

Code snippet

```
# Filter the data frame to only include rows where x is
greater than 2
df_filtered <- filter(df, x > 2)

# Print the filtered data frame
```

```
print(df_filtered)

Output:
  x y z
2  2  7 12
3  3  8 13
4  4  9 14
```

mutate() | Adds new columns to a data frame. |

Code snippet

```
# Add a new column to the data frame called "sum" which is
the sum of the x and y columns
df <- mutate(df, sum = x + y)

# Print the data frame with the new column
print(df)

Output:
  x y z sum
1  1  6 11  7
2  2  7 12  9
3  3  8 13 11
4  4  9 14 13
5  5 10 15 15
```

transmute() | Creates a new data frame with only the specified columns. |

Code snippet

```
# Create a new data frame with only the x and y columns
df_transmuted <- transmute(df, x, y)

# Print the new data frame
print(df_transmuted)

Output:
  x y
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
```

join() | Joins two data frames together based on a common column. |

Code snippet

```
# Create a new data frame with additional information
df_new <- data.frame(x = c(1, 2, 3), age = c(25, 30, 35))

# Join the two data frames together based on the x column
df_joined <- join(df, df_new, by = "x")

# Print the joined data frame
print(df_joined)

Output:
   x y z age
1  1  6 11  25
2  2  7 12  30
3  3  8 13  35
```

14. Write the script to sort the values contained in the following vector in ascending order and descending order(23,45,10,34,89,20,67,99).Demonstrate the output.

```
# Given vector
values <- c(23, 45, 10, 34, 89, 20, 67, 99)

# Sort in ascending order
ascending_order <- sort(values)
cat("Ascending order:", ascending_order, "\n")

# Sort in descending order
descending_order <- sort(values, decreasing = TRUE)
cat("Descending order:", descending_order, "\n")
```

Without in-built function

```
bubble_sort <- function(vector) {
  n <- length(vector)
  for (i in 1:(n - 1)) {
    for (j in 1:(n - i)) {
      if (vector[j] > vector[j + 1]) {
        temp <- vector[j]
        vector[j] <- vector[j + 1]
        vector[j + 1] <- temp
      }
    }
  }
  return(vector)
}
```

```
# Given vector
values <- c(23, 45, 10, 34, 89, 20, 67, 99)

# Sort in ascending order using bubble sort
ascending_order <- bubble_sort(values)
cat("Ascending order:", ascending_order, "\n")

# Sort in descending order using bubble sort
descending_order <- rev(bubble_sort(values))
cat("Descending order:", descending_order, "\n")
```

15. Name and explain the operators used to form data subsets in R.
    Subsetting in R Programming - GeeksforGeeks