Department of Computer Engineering
Academic Term: July-November 2023

Rubrics for Lab Experiments

Class : B.*E. Computer* **Subject Name :*BDA***
Semester : VII **Subject Code :**

Assignment No:	02
Title:	BDA Assignment 2
Date of Performance:	15/10/2023
Roll No:	9427
Name of the Student:	Atharva Prashant Pawar

Evaluation:

Performance Indicator	Below average	Average	Good	Excellent	Marks
On time Submission (2)	Not submitted(0)	Submitted after deadline (1)	Early or on time submission(2)	---	
Test cases and output (4)	Incorrect output (1)	The expected output is verified only a for few test cases (2)	The expected output is Verified for all test cases but is not presentable (3)	Expected output is obtained for all test cases. Presentable and easy to follow (4)	
Coding efficiency (2)	The code is not structured at all (0)	The code is structured but not efficient (1)	The code is structured and efficient. (2)	-	
Knowledge(2)	Basic concepts not clear (0)	Understood the basic concepts (1)	Could explain the concept with suitable example (1.5)	Could relate the theory with real world application(2)	
Total					

Signature of the Teacher :

BDA Assignment 2

Q1) How the business problems have been successfully solved faster, cheaper and more effectively considering NOSQL Google's Mapreduce case study. Also illustrate the business drivers and the findings in it.

Sol:

The following case studies demonstrate how business problems have successfully been solved faster, cheaper, and more effectively by thinking outside the box. Table 1. summarizes five case studies where NoSQL solutions were used to solve particular business problems. It presents the problems, the business drivers, and the ultimate findings.

Table 1. The key case studies associated with the NoSQL movement - the name of the case study/ standard, the business drivers, and the results (findings) of the selected solutions

Case study	Standard Driver	Finding
Google's Bigtable	Need to flexibly store tabular data in a distributed system.	By using a sparse matrix approach, users can think of all data as being stored in a single table with billions of rows and millions of columns without the need for up-front data modelling.
Amazon's Dynamo	Need to accept a web order 24 hours a day, 7 days a week.	A key-value store with a simple interface can be replicated even when there are large volumes of data to be processed.
MarkLogic	Need to query large collections of XML documents stored on commodity hardware using standard query languages.	By distributing queries to commodity servers that contain indexes of XML documents, each server can be responsible for processing da

Case study: Google's MapReduce - use commodity hardware to create search indexes

One of the most influential case studies in the NoSQL movement is the Google MapReduce system. In this paper, Google shared their process for transforming large volumes of web data content into search indexes using low-cost commodity CPUs. Though sharing of this information was significant, the concepts of map and reduce weren't new. Map and reduce functions are simply names for two stages of a data transformation, as described in figure 1. The initial stages of the transformation are called the map operation. They're responsible for data extraction, transformation, and filtering of data. The results of the map operation are then sent to a second layer: the reduce function. The reduce function is where the results are sorted, combined, and summarized to produce the final result. The core concepts behind the map and reduce functions are based on solid computer science work that dates back to the 1950s when programmers at MIT implemented these functions in the influential LISP system. LISP was different than other programming languages because it emphasized functions that transformed isolated lists of data. This focus is now the basis for many modern functional programming languages that have desirable properties on distributed systems. Google extended the map and reduce functions to reliably execute on billions of web pages on hundreds or thousands of low-cost commodity CPUs. Google made map and reduce work reliably on large volumes of data and did it at a low cost. It was Google's use of MapReduce that encouraged others to take another look at the power of functional programming and the ability of functional programming systems to scale over thousands of low-cost CPUs. Software packages such as Hadoop have closely modeled these functions.

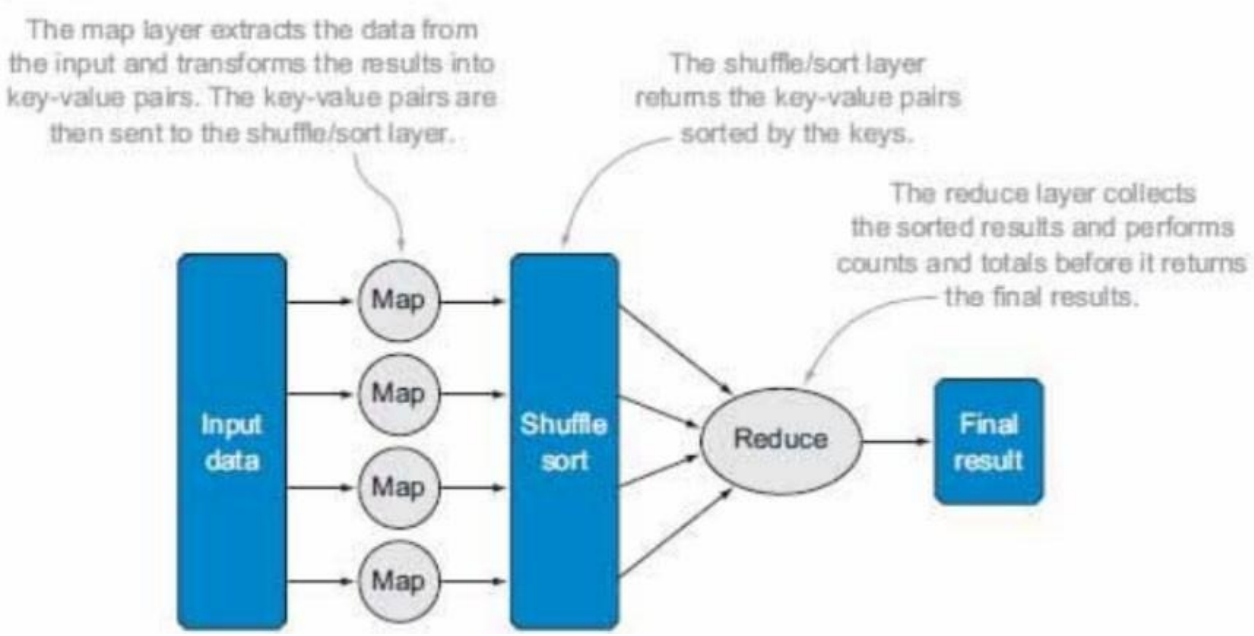


Figure 1. The map and reduce functions are ways of partitioning large datasets into smaller chunks that can be transformed on isolated and independent transformation systems. The key is isolating each function so that it can be scaled onto many servers.

The use of MapReduce inspired engineers from Yahoo! and other organizations to create open source versions of Google's MapReduce. It fostered a growing awareness of the limitations of traditional procedural programming and encouraged others to use functional programming systems.

Case study: Google's Bigtable - a table with a billion rows and a million columns

Google also influenced many software developers when they announced their Bigtable system white paper titled A Distributed Storage System for Structured Data. The motivation behind Bigtable was the need to store results from the web crawlers that extract HTML pages, images, sounds, videos, and other media from the internet. The resulting dataset was so large that it couldn't fit into a single relational database, so Google built their own storage system. Their fundamental goal was to build a system that would easily scale as their data increased without forcing them to purchase expensive hardware. The solution was neither a full relational database nor a filesystem, but what they called a "distributed storage system" that worked with structured data.

By all accounts, the Bigtable project was extremely successful. It gave Google developers a single tabular view of the data by creating one large table that stored all the data they needed. In addition, they created a system that allowed the hardware to be located in any data center, anywhere in the world, and created an environment where developers didn't need to worry about the physical location of the data they manipulated.

Case study: Amazon's Dynamo - accept an order 24 hours a day, 7 days a week

Google's work focused on ways to make distributed batch processing and reporting easier, but wasn't intended to support the need for highly scalable web storefronts that ran 24/7. This development came from Amazon. Amazon published another significant NoSQL paper: Amazon's 2007 Dynamo: A Highly Available Key-Value Store. The business motivation behind Dynamo was Amazon's need to create a highly reliable web storefront that supported transactions from around the world 24 hours a day, 7 days a week, without interruption. Traditional brick-and-mortar retailers that operate in a few locations have the luxury of having their cash registers and point-of-sale equipment operating only during business hours. When not open for business, they run daily reports, and perform backups and software upgrades. The Amazon model is different. Not only are their customers from all corners of the world, but they shop at all hours of the day, every day. Any downtime in the purchasing cycle could result in the loss of millions of dollars. Amazon's systems need to be ironclad reliable and scalable without a loss in service. In its initial offerings, Amazon used a relational database to support its shopping cart and checkout system.

They had unlimited licenses for RDBMS software and a consulting budget that allowed them to atract the best and brightest consultants for their projects. In spite of all that power and money, they eventually realized that a relational model wouldn't meet their future business needs. Many in the NoSQL community cite Amazon's Dynamo paper as a significant turning point in the movement. At a time when relational models were still used, it challenged the status quo and current best practices. Amazon found that because key-value stores had a simple interface, it was easier to replicate the data and more reliable. In the end, Amazon used a key-value store to build a turnkey system that was reliable, extensible, and able to support their 24/7 business model, making them one of the most successful online retailers in the world.

Case study: MarkLogic

In 2001 a group of engineers in the San Francisco Bay Area with experience in document search formed a company that focused on managing large collections of XML documents. Because XML documents contained markup, they named the company MarkLogic. MarkLogic defined two types of nodes in a cluster: query and document nodes. Query nodes receive query requests and coordinate all activities associated with executing a query. Document nodes contain XML documents and are responsible for executing queries on the documents in the local filesystem. Query requests are sent to a query node, which distributes queries to each remote server that contains indexed XML documents. All document matches are returned to the query node. When all document nodes have responded, the query result is then returned. The MarkLogic architecture, moving queries to documents rather than moving documents to the query server, allowed them to achieve linear scalability with petabytes of documents.MarkLogic found a demand for their products in US federal government systems that stored terabytes of intelligence information and large publishing entities that wanted to store and search their XML documents. Since 2001, MarkLogic has matured into a general-purpose highly scalable document store with support for ACID transactions and fine-grained, role-based access control. Initially, the primary language of MarkLogic developers was XQuery paired with REST; newer versions support Java as well as other language interfaces.

MarkLogic is a commercial product that requires a software license for any datasets over 40 GB. NoSQL is associated with commercial as well as open source products that provide innovative solutions to business problems.

Q2) Explain the concept of bloom filter with an example.

1. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set.
2. It works by using a bit array and a set of hash functions to map elements to the bit array.
3. 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.
4. 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.

5. 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.
6. 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.
7. **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.

Q3) Can you explain the difference between a local and distributed file system in the context of MapReduce?

1. Local File System (LFS) :
The basic file system of Linux operating system is termed as Local file system. It stores any data file as it is in single copy.
It stores data files in Tree format. Here, any user can access data files directly. LFS does not Replicate the data blocks. It always used for storing and processing personal data(small data).

2. Distributed File System (DFS) :
When we need to store and process a large data file (approx 1 TB size file at least), the Local file system of Operating system is not appropriate. In such cases we use Distributed File system. It can be created on any Linux operating system with Hadoop. DFS stores any data file by dividing it into several blocks.
This file system works on Master-Slave format where Master is NameNode and DataNodes are the slaves. All the blocks of a Data file is stored into different DataNodes and the location is only known by NameNode. Every Data Block is replicated into different Datanodes to avoid data loss when any datanode fails. In DFS Data files are directly not accessible to any user because only NameNode knows where the Data blocks of Data file are stored.

MapReduce Context:

Local File System in MapReduce:

MapReduce jobs can run on data stored in local file systems, but they lack the advantages of parallelism and scalability inherent in distributed file systems.

Distributed File System in MapReduce:

Hadoop MapReduce is designed to work seamlessly with distributed file systems like HDFS. MapReduce tasks can operate on data distributed across the cluster, taking advantage of parallel processing on different nodes.

Q4) What are the benefits of using MapReduce for distributed data processing.

Sol:

MapReduce is a programming model and processing engine designed for distributed data processing. Here are some of the benefits of using MapReduce for distributed data processing:

1. **Scalability:** MapReduce allows the processing of large-scale data sets by distributing the workload across a cluster of machines. The system scales horizontally, enabling the addition of more nodes to handle increasing data volumes.
2. **Parallel Processing**: The model enables parallel processing by dividing the input data into smaller chunks processed independently on different nodes. Mapper and Reducer tasks can operate concurrently, optimizing performance.
3. **Fault Tolerance:** MapReduce frameworks, like Hadoop, provide fault tolerance by replicating data across nodes. In case of node failures, tasks are automatically reassigned to healthy nodes, ensuring job completion.
4. **Flexibility:** MapReduce is versatile and can handle a variety of data processing tasks, from simple data filtering and aggregation to complex machine learning algorithms. Users can implement custom Mapper and Reducer functions tailored to specific processing requirements.
5. **Distributed Storage Integration:** Integrates seamlessly with distributed file systems like Hadoop Distributed File System (HDFS). Allows data to be stored across the cluster, enabling efficient data locality for processing.
6. **Cost-Effective:** Leverages commodity hardware in a cluster, making it a cost-effective solution for large-scale data processing compared to specialized hardware solutions.
7. **Ease of Programming:** Abstracts the complexities of distributed computing, making it easier for developers to write scalable and parallelizable code. The

programming model is inspired by functional programming concepts, simplifying the development of distributed applications.

8. **Community Support:** MapReduce frameworks, especially Apache Hadoop, have large and active communities. This community support leads to ongoing development, improvements, and a wealth of resources for users.
9. **Data Locality:** Optimizes performance by processing data on nodes where it is stored (data locality). Reduces the need for extensive data movement across the network, improving efficiency.
10. **Diversity of Applications:** While initially designed for batch processing, MapReduce frameworks have evolved to support various types of data processing tasks, including batch, interactive, and real-time processing.
11. **Big Data Analytics:** Well-suited for big data analytics tasks, enabling organizations to derive insights from massive datasets efficiently.
12. **Ecosystem and Tool Integration:** MapReduce frameworks are often part of larger ecosystems that include additional tools for data processing, storage, and analysis (e.g., Hadoop ecosystem).

Q5) Explain applications of data streams with examples.

1. **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.
2. **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.
3. **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.
4. **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.
5. **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.

Q6) 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:

1. **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.
2. **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.
3. **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.
4. **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.
5. **Handling out-of-order data:** Data streams may arrive out of order, which can make it difficult to process them accurately.
6. **Dealing with noise and outliers:** Data streams may contain noise and outliers, which can also make it difficult to process them accurately.
7. **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.
8. **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.

Q7) Explain the sliding window problem with the help of an example.

1. 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.
2. 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.
3. **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.

Q8) Explain DGIM algorithm for counting ones in stream with given problem N=24 and data set is 101011000101110110010110 11

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)

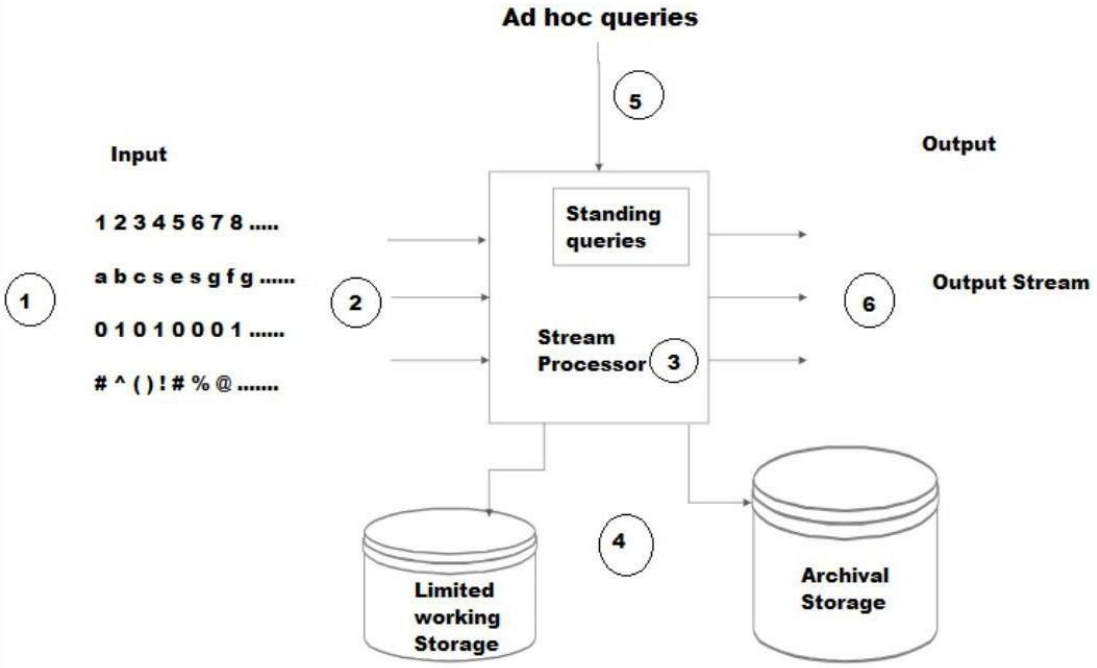
10101100010111 - 1100101 - 101 - 1
2^3 (8) 2^2 (4) 2^1 (2) 2^0 (1)

Q9) How bloom filters are useful for big data analytics explain with example.

1. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set.
2. It works by using a bit array and a set of hash functions to map elements to the bit array.
3. 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.
4. 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.
5. 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.
6. 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.
7. **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.

Q10) 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 paterns 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.

Q11) 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:

1) **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.

2) **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.
3) **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.
4) **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.
5) **Handling out-of-order data: Data** streams may arrive out of order, which can make it difficult to process them accurately.
6) **Dealing with noise and outliers:** Data streams may contain noise and outliers, which can also make it difficult to process them accurately.
7) **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.
8) **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.

Q12) Suppose the stream is 12,31,21,11,12,13,14,3,9,14,20,19 let h(x)=2x+1mod 6 show how the Flajolet-Martin algorithm will estimate the number of distinct elements in this stream.

Sol:

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:
    a. Calculate the index r(x) of the longest trailing run of zeroes in the binary representation of the hash function value of x.
    b. 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

Q13) Explain Girvan -Newman algorithm with the help of given example.

Sol:

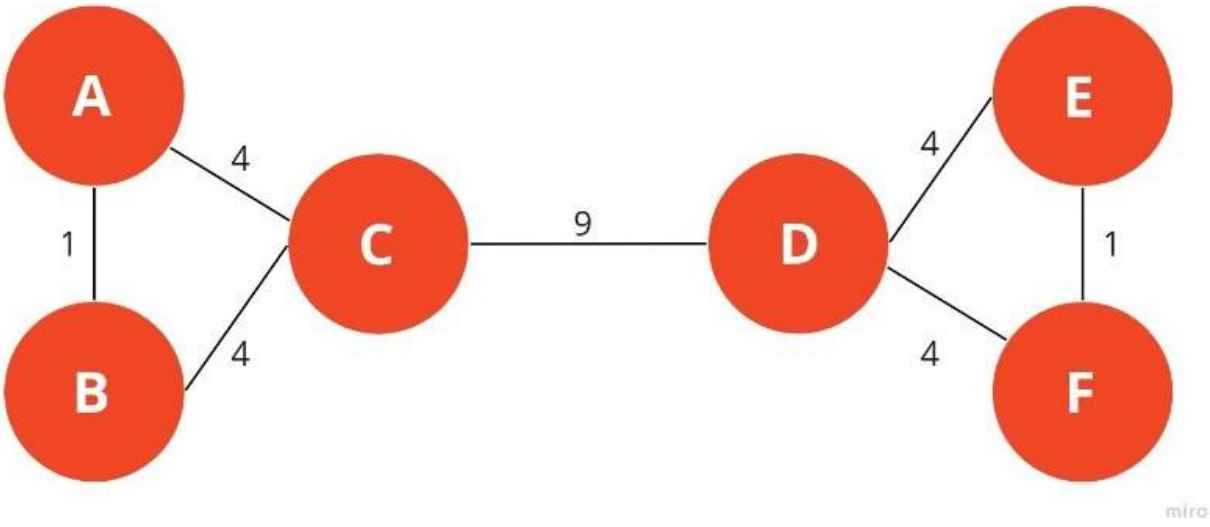
The Girvan-Newman algorithm for the detection and analysis of community structure relies on the iterative elimination of edges that have the highest number of shortest paths between nodes passing through them. By removing edges from the graph one-by-one, the network breaks down into smaller pieces, so-called communities. The algorithm was introduced by Michelle Girvan and Mark Newman.

Working:

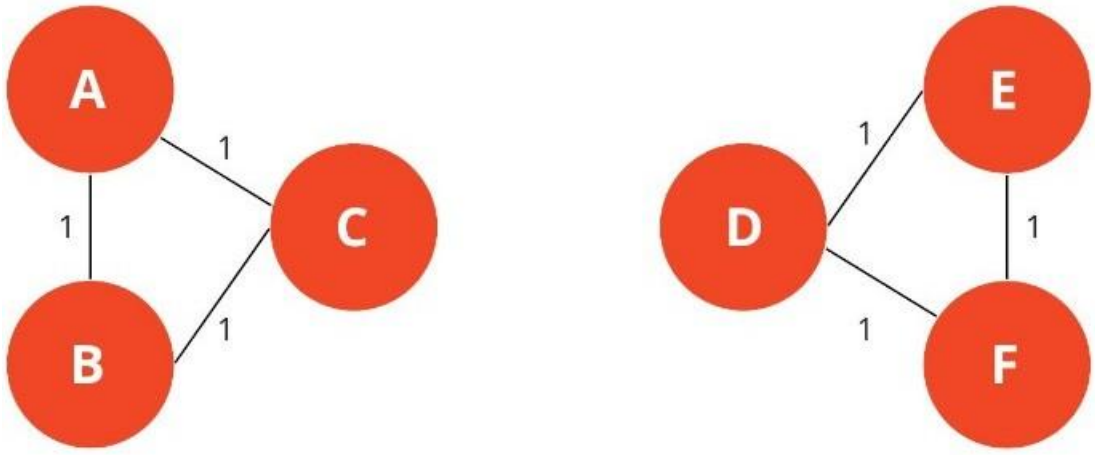
The idea was to find which edges in a network occur most frequently between other pairs of nodes by finding edges betweenness centrality. The edges joining communities are then expected to have a high edge betweenness. The underlying community structure of the network will be much more fine-grained once the edges with the highest betweenness are eliminated which means that communities will be much easier to spot.

The Girvan-Newman algorithm can be divided into four main steps:

1. For every edge in a graph, calculate the edge betweenness centrality.
2. Remove the edge with the highest betweenness centrality.
3. Calculate the betweenness centrality for every remaining edge.
4. Repeat steps 2-4 until there are no more edges left.



In this example, you can see how a typical graph looks like when edges are assigned weights based on the number of shortest paths passing through them. To keep things simple, we only calculated the number of undirected shortest paths that pass through an edge. The edge between nodes A and B has a strength of 1 because we don't count A->B and B->A as two different paths.



The Girvan-Newman algorithm would remove the edge between nodes C and D because it is the one with the highest strength. As you can see intuitively, this means that the edge is located between communities. After removing an edge, the betweenness centrality has to be recalculated for every remaining edge. In this example, we have come to the point where every edge has the same betweenness centrality.

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

Sol:

There are many different functions used for manipulating and processing data in R. Here is a list of some of the most common functions:

a. dplyr: The dplyr package provides a set of functions for manipulating data frames. Some of the most common dplyr functions include:

i. select(): Selects columns from a data frame.

ii. filter(): Filters rows from a data frame based on a condition.

iii. mutate(): Adds new columns to a data frame.

iv. arrange(): Reorders rows in a data frame.

v. group_by(): Groups rows in a data frame by a variable or set of variables.

vi. summarize(): Summarizes data in a data frame by group or by the entire data frame.

b. tidyr: The tidyr package provides a set of functions for cleaning and reshaping data frames. Some of the most common tidyr functions include:

i. spread(): Spreads data from a column in a data frame into multiple columns.

ii. gather(): Gathers data from multiple columns in a data frame into a single column.

iii. complete(): Completes missing values in a data frame.
iv. separate(): Separates a column in a data frame into multiple columns based on a delimiter.

c. ggplot2: The ggplot2 package provides a set of functions for creating data visualizations. Some of the most common ggplot2 functions include:

i. geom_point(): Creates a scater plot.
ii. geom_line(): Creates a line plot.
iii. geom_bar(): Creates a bar chart.
iv. geom_histogram(): Creates a histogram.
v. facet_wrap(): Creates a facet wrap, which is a grid of plots.

Q15) 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.

Sol:

```
# 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")

Q16) Name and explain the operators used to form data subsets in R.
Sol:
Method 1: Subsetting in R Using [ ] Operator
Using the '[ ]' operator, elements of vectors and observations from data frames can be accessed. To neglect some indexes, '-' is used to access all other indexes of vector or data frame.
eg: In this example, let us create a vector and perform subsetting using the [ ] operator.
Create vector x <- 1:15
Subsetting vector
cat("First 5 values of vector: ", x[1:5], "\n") Output - First 5 values of vector: 1 2 3 4 5

Method 2: Subsetting in R Using [[ ]] Operator
[[ ]] operator is used for subsetting of list-objects. This operator is the same as [ ] operator but the only difference is that [[ ]] selects only one element whereas [ ] operator can select more than 1 element in a single command.
Example 1: In this example, let us create a list and select the elements using [[]] operator.
# Create list
ls <- list(a = 1, b = 2, c = 10, d = 20) # Select first element of list cat("First element of list: ", ls[[1]], "\n") Output - First element of list: 1

Method 3: Subsetting in R Using $ Operator

$ operator can be used for lists and data frames in R. Unlike [ ] operator, it selects only a single observation at a time. It can be used to access an element in named list or a column in data frame. $ operator is only applicable for recursive objects or list-like objects.

Example 1: In this example, let us create a named list and access the elements using $ operator

```
# Create list
ls <- list(a = 1, b = 2, c = "Hello", d = "GM") # Print "GM" using $ operator
cat("Using $ operator:\n") print(ls$d)
```

Output - Using $ operator:

[1] "GM"

Method 4: Subsetting in R Using subset() Function

subset() function in R programming is used to create a subset of vectors, matrices, or data frames based on the conditions provided in the parameters.

Example 2: In this example, let us use mtcars data frame present in R base package and selects the car with 5 gears and hp > 200.

```
# Subsetting
mtc <- subset(mtcars, gear == 5 & hp > 200,
select = c(gear, hp)) # Print subset
print(mtc) Output -
gear  hp
Ford Pantera L  5  264
Maserati Bora  5  335
```