Here’s a response to each of your sample midterm questions:

1. **What is the difference between a list where memory is contiguously allocated and a list where linked structures are used?**

    - In a contiguously-allocated list (e.g., an array), memory is allocated in a single contiguous block, which means elements are stored next to each other in memory. This enables fast access to elements by index but can make insertion and deletion operations costly because elements may need to be shifted.
    - In a linked list, each element (node) contains a pointer to the next node in the list. This allows for efficient insertions and deletions (especially at the beginning or middle), but accessing elements by index can be slower since the list needs to be traversed sequentially.
2. **When are linked lists faster than contiguously-allocated lists?**

    - Linked lists are faster when frequent insertions or deletions are required, especially when these operations occur at the beginning or middle of the list. This is because they don’t require shifting elements like in an array.

Add 23 to the AVL Tree below. What imbalance case is created with inserting 23?

```
    30
   / \
  25  35
 /
20
```

3.
    - When 23 is inserted, it would be placed as the right child of 20, resulting in:

```
   30
   / \
  25  35
 /
20
 \
  23
```

4.
    - This creates a left-heavy imbalance at node 25 (specifically a **Left-Right case**), which would require a **Left-Right Rotation** to balance the tree.
5. **Why is a B+ Tree better than an AVL tree when indexing a large dataset?**

    - A B+ Tree is better for large datasets because it is optimized for disk storage and minimizes the number of disk accesses. It is balanced and stores all values at the leaf level, allowing for efficient range queries and fast traversal. AVL trees, on the other hand, are in-memory structures that require more rotations and are less suitable for disk-based data.

6. **What is disk-based indexing and why is it important for database systems?**

    - Disk-based indexing involves using data structures (such as B-trees, B+ trees) to organize and quickly retrieve data from disk. It is crucial because accessing disk storage is much slower than in-memory operations, and indexing helps reduce the number of disk reads needed for queries, making data retrieval more efficient.

7. **In the context of a relational database system, what is a transaction?**

    - A transaction is a sequence of one or more database operations that are executed as a single unit. A transaction ensures that the database remains in a consistent state, either committing all changes or rolling them back if any part of the transaction fails.

8. **Succinctly describe the four components of ACID-compliant transactions.**

    - **Atomicity:** The transaction is an indivisible unit of work; either all operations are completed, or none are.
    - **Consistency:** The database transitions from one valid state to another, maintaining data integrity.
    - **Isolation:** Transactions are isolated from each other, ensuring that concurrent transactions do not interfere with each other.
    - **Durability:** Once a transaction is committed, its changes are permanent, even in the event of a system crash.

9. **Why does the CAP principle not make sense when applied to a single-node MongoDB instance?**

    - The CAP theorem applies to distributed systems where there is a trade-off between Consistency, Availability, and Partition tolerance. In a single-node MongoDB instance, there is no partitioning, so the CAP principle does not apply as there is no need to balance these three properties.

10. **Describe the differences between horizontal and vertical scaling.**

    - **Horizontal scaling** involves adding more machines (nodes) to a system to spread the load. It's often used in distributed systems and can improve availability and fault tolerance.
    - **Vertical scaling** involves adding more resources (e.g., CPU, RAM) to a single machine. It is limited by the capacity of a single machine and can be more costly and less flexible than horizontal scaling.

11. **Briefly describe how a key/value store can be used as a feature store.**

- A key/value store can store features (data attributes) where the key is a unique identifier for a particular entity (e.g., a user or product), and the value is the feature vector associated with that entity. This allows fast retrieval and efficient storage of machine learning features for real-time or batch processing.

12. **When was Redis originally released?**

    - Redis was originally released in **2009**.

13. **In Redis, what is the difference between the INC and INCR commands?**

    - **INC** is not a standard Redis command. However, **INCR** is a Redis command that increments the value of a key by 1. If the key doesn't exist, it is created with the value 1.

14. **What are the benefits of BSON over JSON in MongoDB?**

    - BSON (Binary JSON) is more efficient than JSON for several reasons:
        - It supports more data types (e.g., `ObjectId`, `Date`, `Binary`).
        - It is more compact because it stores data in binary format, which reduces storage size.
        - It allows for faster parsing since it is designed to be more efficient in MongoDB's internal processing.

Write a Mongo query based on the movies dataset that returns the titles of all movies released between 2010 and 2015 from the suspense genre.

```
 db.movies.find(
  {
    genre: "suspense",
    releaseYear: { $gte: 2010, $lte: 2015 }
  },
  { title: 1, _id: 0 }
)
```

15.
16. **What does the $nin operator mean in a Mongo query?**

The `$nin` operator in MongoDB is used to match documents where the value of a field is not in a specified array. For example:

```
 db.collection.find({ field: { $nin: [value1, value2] } })
```

- This would return all documents where the `field` value is not `value1` or `value2`.