Core Data Structures Scenari

(correction of the course exercise)

1. **Storing Student Grades**

   Data Structure: Array/Hash Tables
   Justification: Arrays are suitable for storing student grades for a single course if the number of students is known and constant, as arrays provide fast access to elements at any position. Hash tables could also be used along with arrays to store the grades of multiple students.

2. **Keeping Track of Patients in a Doctor Waiting Room**

   Data Structure: Queue
   Justification: A queue is ideal for this scenario as it follows the First In First Out (FIFO) principle. Patients are "served" in the order they arrive, and a queue efficiently manages this kind of sequential processing.

3. **Implementing Undo Functionality in a Text Editor**

   Data Structure: Stack
   Justification: A stack is appropriate for undo functionality because it operates on the Last In First Out (LIFO) principle. The most recent actions (like text edits) are stored at the top and can be reversed in the exact opposite order they were made.

4. **Organizing a Company's Organizational Structure**

   Data Structure: Tree
   Justification: Trees are perfect for representing hierarchical relationships, such as those in an organizational chart. Each node (employee) has a parent node (supervisor), except for the root node (CEO or top-level manager).

5. **Mapping Routes and Connections in a Public Transportation System**

   Data Structure: Graph
   Justification: Graphs are ideal for modeling networks with multiple connections, like public transportation systems. Nodes represent stops or

stations, while edges represent routes between them. Graphs can handle complex relationships and paths, including one-way routes (directed graphs) or bidirectional connections (undirected graphs).

6. **Managing a Library Catalog System**

   Data Structure: Hash Table
   Justification: Hash tables are efficient for scenarios requiring quick data retrieval, such as looking up a book in a library catalog. By storing book information in key-value pairs (e.g., book ID as the key and book details as the value), the system can quickly retrieve book details using the key, ensuring fast and efficient access.

7. **Implementing a Playlist for a Music Streaming Service**

   Data Structure: Linked List
   Justification: A linked list is ideal for a music playlist where songs can be added, removed, or skipped over. The dynamic nature of linked lists allows for easy insertion and deletion without reallocating the entire list, which is beneficial for frequently changing playlists.

8. **Organizing Books in a Bookshelf Based on Categories**

   Data Structure: Tree
   Justification: A tree is effective for organizing books where each node can represent a book category, and books within each category can be organized in a specific order.

9. **Analyzing Social Networks**

   Data Structure: Graph
   Justification: Graphs are ideal for representing social networks where individuals are nodes and relationships or interactions are edges. They can easily depict complex interconnected relationships and support both directed and undirected connections.

10. **Implementing a Username Lookup System in an Application**

    Data Structure: Hash Table
    Justification: A hash table is perfect for a username lookup system due to its efficient search capabilities. Usernames can be stored as keys, providing quick access to user data. Hash tables are excellent for operations with a high frequency of lookups and minimal insertions or deletions.

11. **Tracking the Order of Documents Printed by a Printer**

    Data Structure: Queue
    Justification: Queues are effective for managing print jobs in a printer. As documents are sent to the printer, they are enqueued and processed in the order they are received, adhering to the FIFO principle.

12. **Building a Browser's History Feature**

    Data Structure: Stack
    Justification: A stack is suitable for a browser's history where the most recently visited pages are stored at the top. It allows users to backtrack through their browsing history in a LIFO manner, reflecting the exact order in which pages were visited.