Objective and Implementation
The objective of Task 2 was to implement a Queue to manage borrowing requests in a Library Management System. The BorrowRequest class was designed with attributes userId, bookId, and requestTime to uniquely identify each request. The BorrowQueue class was implemented using a Linked List to support the following methods:

- enqueue(BorrowRequest request): Adds a request to the end of the queue.
- dequeue(): Removes and returns the first request from the queue.
- size(): Returns the current number of requests in the queue.
- printQueue(): Displays all pending requests.

Test Scenario and Results
The test scenario involved:

1. Adding 10 borrowing requests to the queue.
2. Processing (dequeuing) 3 requests.
3. Printing the remaining 7 requests.
   The output confirmed the FIFO (First-In-First-Out) behavior, with requests being processed in the order they were received.

Analysis Questions

1. **Why is a Queue suitable for managing borrowing requests?**
   A Queue ensures fairness by processing requests in the order they arrive (FIFO), which is critical for a library system where users expect equitable access to resources. This structure mirrors real-world scenarios like waiting in line.

2.

What issues would arise if a Stack were used instead? Compare their Big O complexities.
A Stack follows LIFO (Last-In-First-Out), which would unfairly prioritize recent requests over earlier ones, leading to user dissatisfaction.

- **Big O Comparison**:
  - **Queue (Linked List)**:
    - enqueue(): O(1) (appending to the tail).
    - dequeue(): O(1) (removing from the head).
  - **Stack (Linked List)**:
    - push(): O(1) (adding to the head).
    - pop(): O(1) (removing from the head).
      While both have O(1) operations for insertion/deletion, their behavioral differences make the Queue more appropriate for this use case.