Week 02 - Lab Practice Report

I. Code Explanation

The linked list implementation consists of two main classes: Item and Linkedlist.

- **Item class**: Represents the data stored in each node. It contains a name and an id to uniquely identify the item. It includes a method printInfo() to display the item's details.
- **Node struct**: Each node holds an item and a pointer to the next node in the list.
- **LinkedList class**: Manages the nodes and provides methods to manipulate the list:
    - insertFront(Item) : Inserts a new node at the front of the list (O(1) operation).
    - insertEnd(Item) : Inserts a new node at the end (O(n) operation).
    - insertAfter(int id, Item) : Inserts after a node with a given ID.
    - deleteFront() : Deletes the node at the front (O(1))
    - deleteEnd() : Deletes the node at the end (O(n)).
    - deleteById(ind id) : Deletes the node with a specific ID.
    - traverse() : Prints the entire list.
    - swapNodes(int id1, int id2) : Swaps two nodes identified by their IDs without swapping their data.
    - Search (int id) : Finds and returns a pointer to an `Item` by its ID.
    - Destructor : Frees all allocated memory to avoid leaks.

All operations are designed to respect the time complexity constraints of linked lists and handle edge cases such as empty lists or single-node lists.

II. Answer

Challenge 1 — Insert at the Front

- Implemented insertFront() which inserts a node in O(1) time by adjusting the head pointer.

Challenge 2 — Insert at the End

- Implemented insertEnd() which traverses the list to append at the end (O(n)).

Challenge 3 — Insert After a Specific ID

- Implemented insertAfter(int id , Item) to insert a node after a node matching a given ID.

Challenge 4 — Delete from the Front

- Implemented deleteFront() which removes the head node efficiently (O(1)).

Challenge 5 — Delete from the End

- Implemented deleteEnd() which traverses the list to remove the last node (O(n)).

Challenge 6 — Delete by ID

- Implemented deleteById(int id) to remove a node by matching its ID.

Challenge 7 — Traverse the List

- Implemented traverse() to print all nodes in the list sequentially.

Challenge 8 — Swap Two Nodes (by ID)

- Implemented swapNodes(int id1, int id2) to swap two nodes' positions in the list, handling all edge cases.

Challenge 9 — Search by ID

- Implemented Search (int id) to find a node and return a pointer to its data.

Challenge 10 — Compare with array

- linked lists are better suited for applications where data is frequently added or removed from various positions, and the size is unpredictable. Arrays are better when the number of elements is stable, and fast random access by index is important.
- In the context of this assignment, where the focus is on practicing insertions, deletions, and node manipulation, using a linked list is clearly the better choice. It allows these operations to be done more efficiently and with greater flexibility compared to an array. Arrays would make these tasks more complex and slower due to the need for shifting and resizing.