Linked List Java Problem Set

This problem set is designed to test your understanding and Java implementation skills for the following topics: - Singly Linked List (SLL) - Doubly Linked List (DLL) - Tail Pointer optimization - Fast and Slow Pointer techniques - Merging, Searching, Finding, Deleting in the middle - Reversing a linked list

Each problem also asks you to analyze the **runtime complexity**.

---

1. Singly Linked List Basics

Problem: Implement a singly linked list with the following methods: - `addFirst(int data)` - `addLast(int data)` - `removeFirst()` - `removeLast()` - `get(int index)`

Tasks: 1. Implement all methods in Java. 2. Determine the runtime of each method.

Expected Runtime: - `addFirst`: O(1) - `addLast`: O(n) - `removeFirst`: O(1) - `removeLast`: O(n) - `get`: O(n)

---

2. Tail Pointer Optimization

Problem: Modify your SLL to include a **tail pointer**. - Implement `addLast(int data)` in O(1) time. - Implement `removeLast()` in O(n) time.

Tasks: 1. Implement the optimized SLL. 2. Explain why `addLast` is now O(1) and `removeLast` is still O(n).

---

3. Doubly Linked List (DLL)

Problem: Implement a doubly linked list supporting: - `addFirst(int data)` - `addLast(int data)` - `removeFirst()` - `removeLast()` - `remove(Node node)`

Tasks: 1. Implement all methods. 2. Analyze the runtime of each method.

Expected Runtime: - `addFirst` / `addLast`: O(1) - `removeFirst` / `removeLast` / `remove(node)`: O(1)

---

4. Fast and Slow Pointer

Problem: Detect a cycle in a singly linked list using **fast and slow pointers**.

Tasks: 1. Implement a method `boolean hasCycle(Node head)`. 2. Explain why the runtime is O(n) and space is O(1). 3. Bonus: Find the starting node of the cycle.

---

5. Merging Two Sorted Linked Lists

Problem: Given two sorted singly linked lists, merge them into a single sorted linked list.

Tasks: 1. Implement `Node merge(Node a, Node b)`. 2. Explain the runtime complexity.

Expected Runtime: O(n + m), where n and m are the lengths of the two lists.

---

6. Searching and Finding in the Middle

Problem: Implement methods for a singly linked list: - `find(int data)` → returns the node if found - `delete(int data)` → deletes the first occurrence of the node

Tasks: 1. Implement the methods. 2. Explain the runtime of each operation.

Expected Runtime: O(n) for both operations.

---

7. Reversing a Linked List

Problem: Reverse a singly linked list in-place.

Tasks: 1. Implement `Node reverse(Node head)`. 2. Explain why the runtime is O(n) and space is O(1).

---

8. Bonus Challenge

Problem: Implement an LRU Cache using a doubly linked list and a hash map. - Support `get(key)` and `put(key, value)` in O(1) time.

Tasks: 1. Implement the LRU Cache. 2. Explain why the doubly linked list + hash map combination achieves O(1) operations.

Note: For all problems, write clean Java classes, include Node definitions, and comment your code to explain pointer manipulations.