B1 - MARQUEE BATCH

Model Assessment (Google-Level DSA Problems - Java)

1. **ARRAYS - Fuel-Efficient Route Detection (Google Maps)**

Problem:
You are given an array representing fuel cost per kilo-meter for a long trip.
Find the **most fuel-efficient contiguous route segment** of length ≥ **k**.

Requirements:

- Must run in **O(n)** using sliding window or prefix sums.
- Return: **start index, end index, and minimum total cost**.
- If two segments have same cost → choose **shorter** one.
- If still tie → choose the one that starts **earlier**.

2. **STRINGS - Gmail Autocorrect Suggestion Engine**

Problem:
Given a typed incorrect word and a dictionary of valid words,
find the **closest valid suggestions** using minimum edit distance.

Requirements:

- Dictionary size can be **up to 100,000 words** → must optimize.
- Return the **top 3 closest words**, not just one.
- Ties: return lexicographically smallest.
- Total computation must be < **100 ms per query**.

3. **RECURSION - File Explorer Folder Size Computation**

Problem:
Compute the **total size** of a folder that contains nested folders and files.

Requirements:

- Folder structure is a tree but may contain **symbolic links** → detect cycles.
- Use recursion + memoization to avoid recomputing sizes.
- Also return the **largest 3 files inside** the folder.
- No global static variables allowed.

4. **LINKED LIST - Music Playlist Engine (Spotify-like)**

Problem:
Build a playlist system with:

- playNext()
- playPrev()
- addSong()
- removeSong()
- prevent duplicate songs

Requirements:

- All operations must run in **O(1)** using doubly linked list + HashSet.
- Maintain a **current pointer** that moves dynamically.
- Support exporting playlist both **forward and backward**.

5. **STACK - Google Docs Undo/Redo Manager**

Problem:
Design the undo/redo engine for Google Docs supporting text editing.

Requirements:

- Handle **10,000+ operations**.
- Undo stack can store only **latest 50 actions** (drop oldest automatically).
- Any new action clears the redo stack.
- Support two actions:
  - INSERT(x)
  - DELETE(k characters)
- Return:
  - **final document**
  - **total operations performed**

6. **QUEUE - Hospital Emergency Priority System**

Problem:
Patients arrive with name, severity, and arrival time.
Create a system to pick patients in correct priority order.

Requirements:

- Maintain 2 queues:
  - **Critical Queue (8–10 severity)**
  - **Normal Queue (1–7 severity)**
- At least **1 critical patient every 5 minutes** must be treated.
- If severity is same → earlier arrival first.
- Must support real-time updates and retrieval.

7. **TREE - Lowest Common Directory in a File System**

Problem:
Given a Unix-like directory tree and two file paths,
find the **lowest common directory**.

Requirements:

- Tree may have **100,000+ nodes** → LCA must be **O(log n)** using binary lifting.
- Convert file path /a/b/c.txt → node ID before solving.
- Return the **full directory path** of the LCA.

8. **HEAP - YouTube Real-Time Trending Videos**

Problem:
Each video has a dynamic "trending score" updated in real-time.
Return **top K trending videos**.

Requirements:

- updateScore(videoID, score) must run in **O(log n)**.
- Fetch top-K in **O(k log n)**.
- Videos not watched for 24 hours → lose **10% score automatically**.
- If two videos have same score → newest timestamp is higher priority.

9. HASHING - Real-Time Fraud Detection System

Problem:
Given a continuous stream of transactions, detect duplicates within the last **k seconds**.

Requirements:

- A duplicate = same **transaction ID + amount**.
- Must process **10 million transactions/day**.
- Implement using **HashMap + Doubly Linked List** sliding window.
- Return:
    - Whether fraud occurred
    - All suspicious transaction IDs

10. GRAPH + DIJKSTRA - Minimum Toll Path (Google Maps)

Problem:
Given a road network with tolls and traffic weights,
find the **cheapest path** from source to destination.

Requirements:

- Edge cost formula:
  totalCost = toll + (trafficWeight × 0.5)
- Must output:
    - cheapest path
    - total cost
    - list of intersections
    - **second-best alternative path**
- Graph may contain cycles and one-way edges.
- Use **Dijkstra + parent path reconstruction**.