🎮 Interview Task: Design and Implement a Real-Time Player Ladder System

Objective

Design and implement a **real-time player ladder service** for an online game where players' **scores change over time**, and their **ranks update live**.

Scenario

You are building the backend service for a multiplayer game.
Each player has:

- A unique ID (e.g., `player_id`).
- A score that changes frequently (e.g., after each game or event).
- A current level or rank in the global leaderboard.

Your goal is to design and implement a **real-time ladder system** that keeps track of all players, updates their positions dynamically, and allows clients to fetch the **top players** or a player's **current rank** instantly.

Requirements

Functional Requirements

1. **Add / Update Player Score**
    - The system must allow adding new players and updating existing player scores in real time.
    - When a player's score changes, their rank in the leaderboard must adjust immediately.

2. **Get Top Players**
    - Provide an API or function to retrieve the **top N players** (e.g., top 10 or top 100).
3. **Get Player Rank**
    - Provide an API or function to retrieve a **specific player's rank and score** at any time.
4. **Real-Time Ranking**
    - The leaderboard should update ranks **dynamically** when any player's score changes.
5. **Scalability**
    - The system should handle at least **100 score updates per second** efficiently.

Non-Functional Requirements

- **Low Latency:** Rank queries and updates should be fast (ideally <100ms).
- **Persistence:** The leaderboard state should survive restarts (use in-memory + persistent storage hybrid if possible).
- **Accuracy:** Rankings must always reflect the latest scores correctly.
- **Concurrency:** Support multiple players updating their scores at once without race conditions.

Implementation Guidelines

- You can use any programming language or framework you prefer.
- You can build:
    - A **standalone command-line app**, or
      A **REST API service** (preferred).

- Use any data structure or technology that fits (e.g., **Redis sorted sets**, **priority queues**, **binary search trees**, or a **custom ranking algorithm**).
- Focus on correctness, performance, and clarity of design rather than production polish.

Deliverables

1. **Source Code** in a GitHub/GitLab repo (or ZIP).
2. **README.md** including:
    - System design explanation (data structures, scaling plan, etc.)
    - How to run and test the system.
    - Example API calls or test scripts.
3. **Demonstration / Simulation Script** that:
    - Adds multiple players.
    - Updates their scores randomly.
    - Shows that rankings change in real-time.