Use Case Based Problem

Basic

1. What will be your design in the scenario where 250 MB of memory is available to you and 10 file of 100 MB coming from upstream the requirement is to sort these 10 files and save it in database. Which **sorting algorithm** will you use?
    1. sort each word in those 10 files in **alpha-numeric order**.
2. How will you implement **producer/consumer problem** where there are **ten producers** and **ten consumers**.
3. Implement **producer/consumer problem**
    1. Using **custom blocking queue, join, semaphore** and **wait/notify.**
4. Implement Merge Sort and Quick Sort. Learn to calculate time and space complexity.
5. Print **Fibonacci series** based on provided number.
    1. Implement both **recursive** and **non-recursive** approaches. consider using **dynamic programming**.
6. Design a **Chess Game.**
    1. Share **design diagram** or **class diagram.**
7. **Design a student management system. Identify which design patterns will be used here.**
8. Sort **HashMap** by **values**in descending order using **Java 8 Stream APIs.**
9. Write program to find **largest** and **second largest** element in an **unsorted array**.
    1. Program should be scalable enough to **find nth largest element**.
    2. Implement using Java 8 Stream APIs.
10. Design a **Call Center.**
    1. Imagine you have a call center with three levels of employees: fresher, technical lead (TL), product manager (PM). There can be multiple employees, but only one TL or PM. An incoming telephone call must be allocated to a fresher who is free. If a fresher can't handle the call, he or she must escalate the call to technical lead. If the TL is not free or not able to handle it, then the call should be escalated to PM. Design the classes and data structures for this problem.

Advanced

1. Implement **custom thread pool**.
    1. Handle **exceptions**, **monitorexecuting threads** and implement **shutdown** mechanism.
2. Implement **custom cyclic brier** and **custom countdown latch**.
3. Implement Merge Sort using ForkJoin. Learn to calculate **time** and **space complexity**.
4. Design a **Custom Concurrent Counter**. Don't use AtomicInteger.
    1. Implement custom **ReentrantLock**.
5. Design a **generic object pool**. Identify which **design patterns** will be used here.
    1. How do you **limit number of objects** being created in a pool?
6. Design an multi-threaded **Download Manager** which can show the progress of different downloads
7. Design a **custom concurrent HashMap** using **lock splitting** concept.

Note: Each use case should have adequate JUnit test cases. Share all use case code with mentor as maven project.