Design the data structure for an online book reader system

Ambiguity

- User management creation and extension
- Searching the database of books
- Reading a book
- Only one active user at a time
- Only one active book by this user

Object

- User, Library, Book

Relationship

Actions:

Class: Library

- Var: books
- Methods:
    - addBook
    - remove
    - find

Class: UserManager

- var: users
- methods
    - addUser
    - find
    - remove

Class: Display

- var: activeBook, activeUser, pageNumber
- method:
    - displayuer
    - displayBook
    - turnPageForward
    - turnPageBackward
    - refreshUsername
    - refreshTitile
    - refershPage()

Class: Book

- var: bookId, details
- methods: getId, setId, getDetails, setDetails

Class: User

- var: userId, details, accountType
- methods: getId, setId, getDeatils, setDeatils, getAccountType, setAccountType

Design a chat server

- focus on an aspect of the problem that is reasonably broad, but focused enough that you cloud accomplish it during an interview

what specific actions does it need to support?

- Discuss with your interviewer

What can we learn about these requirements?

- Users, add request status, online status, message

What are the core components of the system?

- Database, a set of clients, a set of servers
- Point out the drawbacks of using something

What are the key objects and methods?

Recursion and Dynamic Programming

How to approach

- Recursive solutions: are built off of solutions to subproblems
- There are many ways you might divide a problem into subproblems

Bottom-up approach

- Start with knowing how to solve the problem for a simple case (one element)
- Figure out how to solve the problems for two elements or three
- Key: think about how you can build the solution for one case off of the previous case

Top-Down approach

- Think about how we can divide the problem for case N into subproblems.
- Be careful of overlap. Between the cases

Half-and-half approach

- Merge sort is a half-and-half approach
- Sort each half of the array and then merge together the sorted halves

Dynamic programming and memorization

- Dynamic programming is mostly just a matter of taking a recursive algorithm and finding the overlapping subproblems
- a good way to approach DP problems is often to implement it as a normal recursive solution, and then add the caching part
- Drawing the recursive calls as a tree is a great way to figure out the runtime of a recursive algorithm
- All you really doing is changing which nodes you expand and which ones return cached values.