Quiz component	Component Description
Academic CS	Academic CS covers knowledge of the CS theory covered in typical CS education programs. It includes algorithms and data structures, time and space complexity and big-O notation, and the actual mechanics by which computers operate. This is a good thing to know for the sake of precise engineering, and it's also something many employers heavily emphasize in their interview processes.
Backend web	Back-end web, for our purposes, means web development on the server side, not in a user's browser. Important topics include databases, HTTP, authentication, and API creation. This is a very broad field that encompasses a large percentage of engineering jobs, and it's a good place to start if you want to maximize your chances of finding work.
Front end	Front-end, for our purposes, means web development within a web page loaded in a user's browser. Important topics include JavaScript and its major libraries, CSS, HTTP, and event-driven programming. This field tends to attract engineers excited by the product they can create, who want to see users interacting with their creation directly.
Programmatic problem solving	Programmatic problem solving represents your comfort analyzing the structure of code to understand its flow and intended purpose. For example, if a line is missing from a function, it's often possible to recognize what is missing by looking at surrounding logic. This skill is valuable to almost everyone, especially when working with legacy code or debugging.
System architecture	System architecture is the subfield of engineering that focuses on tying together many mostly-separate components into a single application. The common database/web-server/client divide is one common architecture pattern, for example. Architecture skills are particularly important for senior engineers and tech leads, since they're expected to take charge of development at a high level.
Mobile architecture	Mobile architecture focuses on architecture patterns - common structures used by most apps unless there's some reason to deviate. Examples include the MVC (model-view-controller) pattern officially recommended by Apple for iOS development, along with other approaches like MVVM. This is a useful thing for all mobile developers to understand, but is especially important for senior engineers and tech leads.
Android knowledge	Android knowledge covers the details of the Android platform, as distinct from general-purpose software engineering. Common topics include major APIs and UI components, data persistence, and tools and approaches for performance troubleshooting. Android developers need at least intermediate knowledge of this subject, since so much of mobile development depends on understanding of the platform itself.

iOS knowledge	iOS knowledge covers the details of the iOS platform, as distinct from general-purpose software engineering. Common topics include major APIs and UI components, data persistence, and tools and approaches for performance troubleshooting. iOS developers need at least intermediate knowledge of this subject, since so much of mobile development depends on understanding of the platform itself.
Machine Learning	Machine learning understanding covers knowledge of both the theory behind ML and the pitfalls and trade-offs involved with implementing ML systems in practice. Important topics include different types of ML model (neural network, random forest, k-nearest-neighbors) and their advantages and disadvantages, methods to avoid common failure modes like overfitting, and the underlying math that makes ML possible. Both theory and practical pragmatism are important for ML engineers of any seniority.
DevOps	DevOps covers a wide and 'fuzzy' range of topics, from cloud infrastructure to performance troubleshooting to containerization. There are too many examples to cover them all here, but a few example topics include *nix command line tools; platforms like AWS, Google Cloud, or Azure; orchestration tools like Kubernetes; and infrastructure-as-code tools like Terraform. Few DevOps engineers are experts at everything suggested by that title; engineers looking to transition into DevOps may benefit from picking one or two of these topics and focusing heavily on them.
Low-level systems	Low-level systems covers programming at the operating system level. It involves technical details that are often handled automatically in high-level languages, like pointer handling, concurrency, and explicit memory management. C and C++ are the traditional languages for most low-level development, though alternatives like Rust have become more popular. This is a specialized discipline and is a common area of weakness for everyday application developers; typical low-level programming jobs include low-latency financial trading software, game engines, and some forms of security engineering.