Problem 1: Compare and contrast the Android and iOS operating systems from the following viewpoints:

(1) customizability

Android :A lot. Can change almost anything.
iOS :Limited unless jailbroken

(2) performance

- iOS: Apps are often optimized for specific iOS devices, taking advantage of the hardware and software capabilities, leading to better performance. and Apple's strict control over hardware and software allows for efficient resource management, with background processes and apps being well-managed to preserve performance and battery life.

- Android: Apps may not be as consistently optimized for the variety of Android devices, which can result in performance variations.and Android devices may have more background processes and apps running, potentially affecting performance and battery life if not managed properly.

(3) business model (how does iOS make money for Apple? How does Android make money for Google?) ***Apple*** makes money from iOS primarily through hardware sales and the ecosystem of services and apps surrounding it.,**App Store ,iTunes and Media Services** ,**iCloud,Apple Pay**:Apple takes a small fee for each transaction made through its digital payment system, Apple Pay and **Hardware Accessories**: Apple sells a wide range of accessories, including cases, headphones, and chargers, which contribute to the company's revenue .
Android (Google): **Google Search**: Google is the default search engine on most Android devices. Google generates a substantial portion of its revenue from advertising, and Android helps drive search traffic and ad revenue ,**Google Play Store**,**Google Cloud Services** ,**Licensing Agreements**

Problem 2: AOT vs. JIT

(1) Why does Android use Ahead-of-Time (AOT) compilation rather than Just-in–Time (JIT) compilation? Android uses Ahead-of-Time (AOT) compilation as its primary compilation method rather than Just-in-Time (JIT) compilation due to several practical and performance-related reasons:

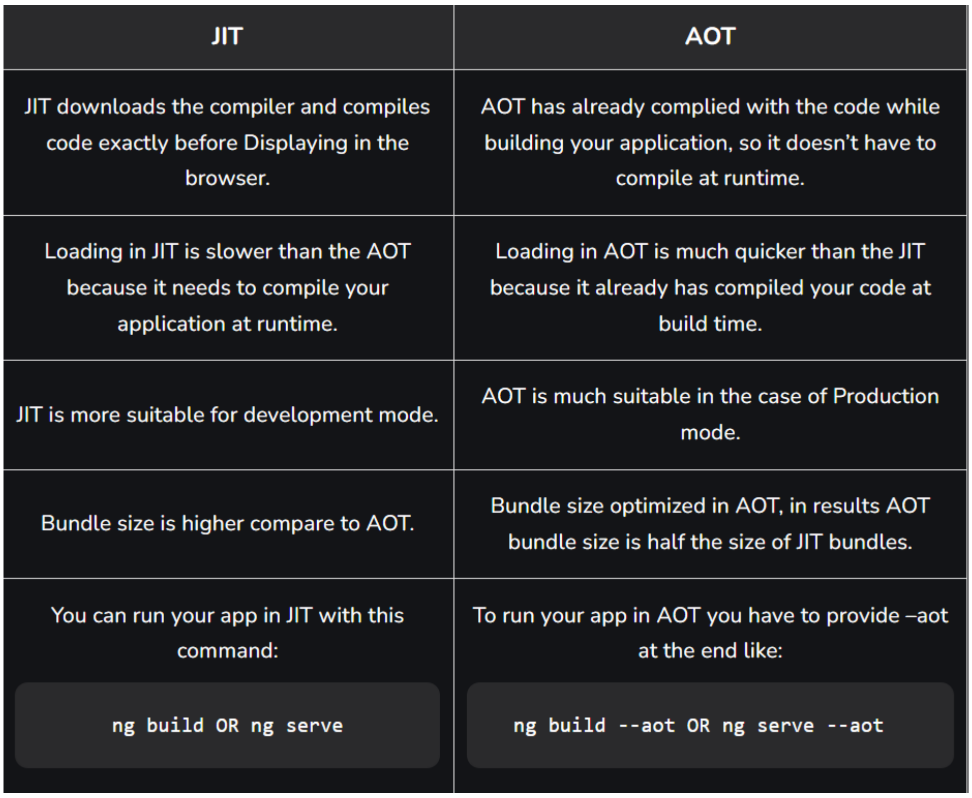
1. Consistency and Predictable Performance: AOT compilation ensures that code is compiled prior to running the program. As a result, performance is more predictable and stable, making it ideal for a wide range of Android devices with diverse hardware capabilities. This predictability is critical for providing a pleasant user experience. .

2. Optimization:The code can be optimized for the exact hardware architecture on which it will operate during the AOT compilation process. Because the compiler has more time to use sophisticated optimization techniques, this optimization can result in improved performance and resource usage.

   .

3. Reduced Overhead: When compared to JIT, AOT compilation often leads in lower runtime overhead. JIT compilation entails interpreting and compiling code at runtime, which adds delay and consumes more memory and computing power. AOT compilation saves most of this runtime cost.

4. Resource-Constrained Devices: Android operates on a wide range of devices with various hardware capabilities. AOT compilation enables programs to be optimized for low-end devices with limited resources, ensuring that they execute effectively.



JIT	AOT
JIT downloads the compiler and compiles code exactly before Displaying in the browser.	AOT has already complied with the code while building your application, so it doesn't have to compile at runtime.
Loading in JIT is slower than the AOT because it needs to compile your application at runtime.	Loading in AOT is much quicker than the JIT because it already has compiled your code at build time.
JIT is more suitable for development mode.	AOT is much suitable in the case of Production mode.
Bundle size is higher compare to AOT.	Bundle size optimized in AOT, in results AOT bundle size is half the size of JIT bundles.
You can run your app in JIT with this command: `ng build OR ng serve`	To run your app in AOT you have to provide –aot at the end like: `ng build --aot OR ng serve --aot`

Figure 1: Enter Caption

5. Security: Because the code is built and checked before execution, AOT compilation can help to improve security. JIT compilation, on the other hand, may present security issues owing to dynamic code compilation during runtime.

6. Application Start-Up Speed: Because the code is already compiled, AOT compilation often results in speedier application start-up times. JIT compilation, on the other hand, may result in some delay at the start of execution because it compiles code on-the-fly.

7. Battery Efficiency: AOT compilation can save energy on mobile devices by reducing CPU utilization related with runtime compilation, resulting in longer battery life.

(2) What are the advantages and disadvantages of each method?

AOT offers better performance, as code is pre-compiled. However, it may increase app size and installation time. JIT can be more memory-efficient but may have a slight performance overhead due to runtime compilation

Problem 3: Native vs. cross-Platform

(1) What is the difference between native mobile code and cross-platform mobile code?

Native Mobile Code:

1. Platform-Specific Development:
   - Native mobile development involves writing code that is specific to a particular mobile platform, such as iOS or Android.
   - For iOS, developers typically use Swift or Objective-C, while for Android, Java or Kotlin is used.
2. Performance and User Experience:
   - Native apps generally offer the best performance and user experience because they can leverage platform-specific features and optimizations.
   - They have direct access to device hardware and APIs, which allows for seamless integration with the operating system.
3. Development Time and Cost:
   - Developing native apps for multiple platforms requires separate codebases, which can be time-consuming and costly.
   - Maintenance and updates are platform-dependent, which can further increase development and support expenses.
4. Access to Platform Ecosystem: Native apps can take full advantage of platform-specific features and libraries, including tools and widgets, offering a more integrated experience.

   Cross-Platform Mobile Code:

   (a) Write Once, Deploy Anywhere: Cross-platform development allows developers to write code once and run it on multiple mobile platforms, such as iOS, Android, and sometimes even web.

   (b) Development Frameworks:
   - Cross-platform frameworks like React Native, Flutter, Xamarin, and others enable developers to use a single codebase for multiple platforms.
   - They often use languages like JavaScript, Dart, or C# and provide access to native device features through abstraction layers.

   (c) Performance and User Experience:
   - Cross-platform apps tend to be slightly less performant compared to native apps because they rely on an additional layer of abstraction.
   - While they can access many native features, there may be limitations in utilizing the latest platform-specific technologies.

   (d) Development Time and Cost:
   - Cross-platform development is generally more cost-effective and faster than native development, as you don't need to write separate code for each platform.
   - Maintenance and updates are more efficient, as changes are made to a single codebase.

   (e) Code Reusability: One of the primary advantages is code reusability. You can reuse a significant portion of the code across different platforms, reducing redundancy.

   (f) Wider Reach: Cross-platform apps can reach a broader audience across different platforms with a single codebase.

(2) What are the advantages and disadvantages of each?

Disadvantages and Benefits:

The Benefits of Native Development: Best performance and platform-specific features. A smooth user experience. Direct access to hardware and APIs on the device. Native Development's Disadvantages:

Costlier development and a lengthier development time. Different platforms have their own codebases. Maintenance and support costs have risen. The Benefits of Cross-Platform Development:

Development is more cost-effective and quicker. Update efficiency and code reusability. Increased exposure across many platforms. Cross-Platform Development's Drawbacks:

Performance is somewhat reduced when compared to native. Utilizing the most recent platform-specific technology may have restrictions. Relies on cross-platform frameworks and plugins.

(3) How can a programming framework like FLUTTER produce mobile apps that work on both Android and iOS?

Flutter, a cross-platform mobile development framework, creates Android and iOS mobile apps using the following mechanisms:

Single Codebase: Using the Dart programming language, developers construct their app's logic and user interface in a single codebase.

Widgets: Flutter has a plethora of customisable widgets that are designed to imitate the appearance and feel of both the Android and iOS platforms.

Dart code is compiled to native ARM code, which ensures high performance on both systems.

Flutter provides a large selection of plugins for accessing native features and APIs on both Android and iOS devices.

Hot Reload: The "Hot Reload" feature of Flutter lets developers to observe real-time changes, which speeds up development and testing.

Flutter apps provide consistent performance and look across both platforms.

Problem 4: Describe google Chrome as an example of multi-process, multi-threaded application.

Multi-Process Architecture:
The browser process is the main or master process. It is in charge of user interfaces like as the browser's tabs, address bar, and settings. If you open Windows Task Manager (or anything similar on other systems), you'll find one "chrome.exe" process representing the browser itself. Renderer Processes: Each open tab in Chrome has its own renderer process. Isolation is a critical security element. If one tab crashes or experiences a security risk, it has no effect on the other tabs or the browser process as a whole. Plugin Processes: Plugins such as Adobe Flash run in their own processes. This separation guarantees that if a plugin fails, the entire browser does not crash. Utility Processes: Various utility processes perform duties like as download management, extension management, and more.

Multi-Threaded Approach:

Chrome employs many threads inside each of these processes to handle various tasks at the same time. These are some of the threads:

Main Thread: Handles user input, manages tabs, and coordinates numerous actions inside a process.

Threads for Rendering Web Content, JavaScript Execution, and managing Input Events: Each renderer process contains numerous threads for rendering web content, running JavaScript, and managing input events. This enables the execution of processes within a tab in parallel, boosting responsiveness.

Networking Threads: To efficiently manage network requests, Chrome downloads resources in distinct threads, which avoids the browser from becoming unusable while a webpage is loaded.

GPU Process: Chrome renders graphics using a distinct GPU process. This division improves performance and security.

Storage Threads: These are threads that manage local storage, cookies, and web data.

Benefits of the Multi-Process, Multi-Threaded Model:

- Security: Separating tabs and processes guarantees that security breaches or failures in one tab do not influence other tabs or jeopardize browser stability.
- Performance: Chrome may use multi-threading to split jobs over many CPU cores, resulting in quicker web page rendering and increased responsiveness.
- Resource Management: The architecture aids in the more effective management of system resources. If one tab takes too much memory or CPU power, it can be closed without impacting the rest of the browser.
- Robustness: The multi-process approach adds robustness to Chrome. Even if one component fails, the browser as a whole remains operational.