Q1: Compilation translates source code into machine code creating an executable file. Interpretation translates and executes code line by line without an executable.
Q2: Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling method overriding.
Q3: Encapsulation bundles data and methods in a class, restricting direct data access. Example: class with private data and public methods.
Q4: An abstract class can't be instantiated and can have abstract and concrete methods. An interface only has method signatures without implementations.
Q5: OOP principles include encapsulation, inheritance, polymorphism, and abstraction, promoting organized and maintainable code.
Q6: A constructor initializes object properties upon class instantiation, ensuring a well-defined state.
Q7: Stack memory stores local variables and function calls; heap memory is for dynamic allocation. Stack operates in LIFO, heap managed manually or by garbage collection.
Q8: Design patterns are solutions to common design problems. Examples: Singleton, Factory, Observer, MVC.
Q9: DRY (Don't Repeat Yourself) advocates for avoiding code duplication by reusing existing code.
Q10: SOLID represents five design principles for OOP: Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
Q11: An array has fixed size and stores elements in contiguous memory; a linked list consists of nodes with data and references, allowing dynamic size.
Q12: Time complexity measures the time an algorithm takes relative to its input size, expressed in Big O notation.
Q13: A binary search tree is hierarchical, maintaining order; a hash table maps keys to values for fast retrieval, without maintaining order.
Q14: A linked list is a series of nodes each containing data and a reference to the next node, allowing dynamic memory allocation and efficient insertions/deletions.
Q15: Recursion is when a function calls itself to solve subproblems, with a base case to terminate recursion.
Q16: Big O notation describes the upper bound of algorithm time complexity, important for comparing efficiency and growth rates.
Q17: Binary search divides the search interval in half, repeatedly comparing the middle element to the target.
Q18: Sorting algorithms vary in time/space complexity and stability. Quick Sort and Merge Sort are fast but more complex; Insertion and Bubble Sort are simple but slower.
Q19: A hash table uses a hash function to map keys to values in an array, allowing fast O(1) access.
Q20: Dynamic programming solves complex problems by dividing them into smaller subproblems, avoiding redundant calculations.
Q21: Java is a compiled, statically-typed language used for server-side, mobile, and desktop apps. JavaScript is an interpreted, dynamically-typed language for web development.
Q22: MVC divides an application into Model (data), View (UI), and Controller (input handling), promoting separation of concerns.
Q23: RESTful API is a web service implementation using HTTP methods to perform CRUD operations on resources, adhering to stateless, client-server architecture.
Q24: "this" in JavaScript refers to the execution context, varying based on function calling, global scope, or event handlers.
Q25: A closure is a function with access to its outer scope variables even after the outer function has executed.
Q26: Python 3 has print as a function, true division, Unicode support by default, and different syntax for exceptions, unlike Python 2.
Q27: Package managers manage installation, update, and dependency resolution of libraries, simplifying library management in development.
Q28: Multi-threading in Java allows concurrent execution of multiple threads, improving application responsiveness and performance.

Q29: Singleton ensures a class has only one instance and provides a global access point to it, useful for shared resources.
Q30: Virtual functions in C++ allow derived classes to override them, enabling runtime polymorphism and dynamic method dispatch.
Q31: A database index speeds up data retrieval, similar to a book's index, improving query performance.
Q32: SQL databases use structured query language with a predefined schema; NoSQL databases store schema-less data with flexible models.
Q33: A foreign key links two tables by referring to the primary key in another table, ensuring referential integrity.
Q34: ACID: Atomicity (indivisible transactions), Consistency (consistent state transitions), Isolation (independent transactions), Durability (persisted changes).
Q35: Optimize using indexes, efficient SQL, limiting data retrieval, analyzing query performance, and considering denormalization.
Q36: Normalization organizes data into separate tables to reduce redundancy and improve integrity, following normalization forms.
Q37: INNER JOIN returns matching rows from both tables; LEFT JOIN returns all rows from the left table and matching rows from the right.
Q38: Stored procedures are precompiled SQL statements for data manipulation and logic, used for repetitive tasks and improving performance.
Q39: Denormalization introduces redundancy for performance, useful in read-heavy scenarios at the expense of storage and complexity.
Q40: ORM simplifies database interactions and is language-agnostic. It can introduce performance overhead and may limit database features.
Q41: The DOM is a tree-like representation of a web page's structure, allowing manipulation of content, structure, and style via programming languages.
Q42: HTTP is an unsecured data transmission protocol; HTTPS is secure, encrypting data in transit using SSL/TLS.
Q43: CORS is a security measure allowing or restricting resources requested from another domain, managed via HTTP headers.
Q44: Web servers handle HTTP requests, serve content, manage security, routing, and can act as reverse proxies for application servers.
Q45: Cookies are data stored on the user's computer by the web server, sent with HTTP requests for session management, tracking, and storing preferences.
Q46: A session maintains stateful information across multiple HTTP requests, typically for user authentication and data storage.
Q47: Responsive design ensures web content functions across different devices and screen sizes, using CSS media queries and flexible layouts.
Q48: GET requests retrieve data and include parameters in the URL; POST requests send data to the server, encapsulating data in the request body.
Q49: SEO enhances a website's visibility in search engine results, improving organic traffic and user reach through optimized content and structure.
Q50: Browsers parse HTML to create a DOM, fetch resources, build a rendering tree, apply CSS, calculate layout, and paint the page on the screen.
Q51: Unit testing evaluates individual code components, ensuring correctness and facilitating early defect detection.
Q52: Black-box tests functionality without internal code knowledge; white-box tests internal code logic and structure.
Q53: Regression testing ensures new code changes don't break existing features, maintaining functionality over updates.
Q54: Code reviews identify defects, improve quality, enforce standards, and facilitate knowledge sharing.

Q55: CI involves frequent code integration and testing; CD extends CI by deploying changes to production automatically after testing.
Q56: Code coverage measures the extent of code tested, assessing test thoroughness and identifying untested areas.
Q57: A test case outlines test conditions, inputs, and expected results, structured with objective, steps, and documentation.
Q58: Load testing evaluates system performance under expected load conditions, identifying bottlenecks and scalability issues.
Q59: Manual testing is human-driven; suitable for exploratory and UX testing. Automated testing uses tools for repetitive tasks; suitable for regression and performance testing.
Q60: A bug tracking system logs, manages, and resolves issues in software development, ensuring systematic problem handling.
Q61: Git is a distributed version control system for tracking changes in source code, allowing collaborative work and branch management.
Q62: Git is distributed, with local repository copies; SVN is centralized, requiring network connectivity for repository access.
Q63: Merge conflicts occur when changes in different branches clash. Resolve by manually editing files and committing the result.
Q64: Branching isolates development work without affecting other parts of the repository, aiding in feature development and experimentation.
Q65: A PR is a request to merge code from one branch to another, facilitating code review and discussion before integration.
Q66: Resolve code conflicts through communication, careful review, manual merging, testing, and documenting resolutions.
Q67: Refactoring improves code structure and readability without altering functionality, enhancing maintainability and quality.
Q68: GitFlow organizes branches and releases, defining naming conventions and branch purposes for structured and organized development.
Q69: Git rebase re-applies commits onto another base for a cleaner history. Use with caution to maintain a linear project history.
Q70: Distributed systems allow offline work, flexible branching/merging, faster operations, redundancy, and collaborative workflows.
Q71: Microservices architecture consists of small, independent services communicating via APIs, each responsible for specific functionality, promoting scalability and maintenance.
Q72: A load balancer distributes incoming traffic across servers, ensuring resource efficiency, fault tolerance, and high availability.
Q73: Caching stores frequently accessed data for faster retrieval, reducing backend load, improving performance, and enhancing user experience.
Q74: A CDN is a network of servers for delivering content efficiently to users based on geographic proximity, reducing latency and load times.
Q75: Monolithic is simple but less scalable; microservices offer scalability and flexibility but are complex to manage.
Q76: Stateless services don't retain client data between requests; stateful services maintain client state, useful for sessions and transactions.
Q77: The CAP theorem states that in a distributed system, you cannot simultaneously guarantee Consistency, Availability, and Partition Tolerance at all times.
Q78: Ensure consistency using strong consistency models, two-phase commits, optimistic concurrency control, and conflict resolution strategies.
Q79: A reverse proxy routes client requests to appropriate servers, providing load balancing, SSL termination, caching, and security.
Q80: A message broker facilitates communication in distributed systems through asynchronous messaging, used in event-driven architectures and high-volume scenarios.

Q81: SQL injection exploits vulnerabilities to execute malicious SQL. Prevent with parameterized queries, input validation, and least privilege access.
Q82: XSS injects malicious scripts into web apps, executed by users' browsers. Prevent with input validation, output encoding, and CSP.
Q83: 2FA adds extra security by requiring two verification forms: something known (password) and something possessed (device).
Q84: Hashing transforms passwords into hashes using algorithms; salting adds randomness, enhancing security against attacks.
Q85: OAuth allows third-party app access to user data without exposing credentials, using access tokens for authorization.
Q86: Protect by regenerating session IDs post-authentication, using unpredictable IDs, and tying IDs to user authentication.
Q87: Least privilege limits access rights; defense in depth layers security. Both minimize attack surfaces and provide redundancy.
Q88: A DDoS attack overwhelms a target with traffic, causing unavailability. Mitigate with DDoS protection, rate limiting, and traffic analysis.
Q89: Secure data by encrypting at rest and in transit, using secure authentication, and following best practices.
Q90: API security is vital to protect data and prevent unauthorized access, using authentication, validation, rate limiting, and encryption.
Q91: Docker is a containerization platform packaging applications with dependencies, ensuring consistent environments across systems.
Q92: Container orchestration automates deployment, scaling, and management of containers, optimizing resource use and handling failures.
Q93: Kubernetes is an open-source container orchestration platform automating deployment and management, known for its scalability and community support.
Q94: CI/CD automates build, test, and deployment processes, delivering code changes rapidly and reliably to production.
Q95: IaC manages infrastructure using code, ensuring consistency, automation, and version control in deployments.
Q96: Monitor using tools to collect and analyze data on response times, resource utilization, error rates, and user experience.
Q97: Automated testing in CI/CD ensures code changes are defect-free, enhancing reliability and speeding up delivery.
Q98: Blue-Green deployment alternates between two production environments for easy rollbacks and minimal downtime during updates.
Q99: Configuration management tools automate provisioning and management of software and infrastructure, ensuring consistency and efficiency.
Q100: Cloud platforms offer scalability, cost-efficiency, global reach, and managed services, reducing operational burdens with security and compliance features.
Q101: A closure is a function that remembers its outer variables and can access them.
Q102: Docker allows for packaging applications in containers, facilitating consistent deployment across different environments.
Q103: A race condition occurs when the system's behavior depends on the sequence or timing of other uncontrollable events.
Q104: Optimizations can include minimizing HTTP requests, using CDNs, compressing files, caching, etc.
Q105: SQL databases are structured, use SQL, and are better for complex queries. NoSQL databases are flexible, scale well, and are good for hierarchical data storage.
Q106: State in React is an object that holds some information that may change over the lifecycle of the component.
Q107: Continuous integration is the practice of automating the integration of code changes into a software project.

Q108: Binary search involves repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed the possibilities to just one.
Q109: MVC architecture stands for Model-View-Controller, separating the application into three interconnected components.
Q110: Microservices are a software development technique■a variant of the service-oriented architecture architectural style that structures an application as a collection of loosely coupled services. In a monolithic architecture, all components are interconnected and interdependent.
Q111: '==' compares values after type conversion, while '===' compares both value and type.
Q112: Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts.
Q113: A RESTful API is implemented by setting up HTTP routes (GET, POST, PUT, DELETE) and handling requests and responses in a stateless manner, often using JSON.
Q114: Server-side rendering improves initial page load time and SEO, while client-side rendering is good for dynamic websites with less initial loading content.
Q115: NoSQL databases are generally more scalable and provide superior performance for large-scale applications due to their flexibility in handling unstructured data.
Q116: Hooks are functions that let you 'hook into' React state and lifecycle features from function components.
Q117: IaC is the management of infrastructure (networks, virtual machines, load balancers, etc.) in a descriptive model, using code, which increases development and deployment speed.
Q118: Memoization is an optimization technique used to speed up programs by storing the results of expensive function calls.
Q119: Advantages include easier scalability, flexibility in choosing technology, better fault isolation, and improved continuous deployment.
Q120: SOLID stands for Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion principles, guiding towards more maintainable, understandable, and flexible software.
Q121: Lazy loading is a design pattern that delays loading of non-critical resources at page load time, reducing initial load time and page weight.
Q122: A load balancer distributes network or application traffic across multiple servers to enhance responsiveness and availability of applications.
Q123: Indexing speeds up data retrieval operations by effectively creating a smaller, faster version of the database table.
Q124: Event delegation refers to the practice of using a single event listener to manage all events of a specific type for child elements.
Q125: A closure is a function that remembers its outer variables and can access them.
Q126: Docker allows for packaging applications in containers, facilitating consistent deployment across different environments.
Q127: A race condition occurs when the system's behavior depends on the sequence or timing of other uncontrollable events.
Q128: Optimizations can include minimizing HTTP requests, using CDNs, compressing files, caching, etc.
Q129: SQL databases are structured, use SQL, and are better for complex queries. NoSQL databases are flexible, scale well, and are good for hierarchical data storage.
Q130: State in React is an object that holds some information that may change over the lifecycle of the component.
Q131: Continuous integration is the practice of automating the integration of code changes into a software project.
Q132: Binary search involves repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed the possibilities to just one.
Q133: MVC architecture stands for Model-View-Controller, separating the application into three interconnected components.

Q134: Microservices are a software development technique■a variant of the service-oriented architecture architectural style that structures an application as a collection of loosely coupled services. In a monolithic architecture, all components are interconnected and interdependent.
Q135: '==' compares values after type conversion, while '===' compares both value and type.
Q136: Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts.
Q137: A RESTful API is implemented by setting up HTTP routes (GET, POST, PUT, DELETE) and handling requests and responses in a stateless manner, often using JSON.
Q138: Server-side rendering improves initial page load time and SEO, while client-side rendering is good for dynamic websites with less initial loading content.
Q139: NoSQL databases are generally more scalable and provide superior performance for large-scale applications due to their flexibility in handling unstructured data.
Q140: Hooks are functions that let you 'hook into' React state and lifecycle features from function components.
Q141: IaC is the management of infrastructure (networks, virtual machines, load balancers, etc.) in a descriptive model, using code, which increases development and deployment speed.
Q142: Memoization is an optimization technique used to speed up programs by storing the results of expensive function calls.
Q143: Advantages include easier scalability, flexibility in choosing technology, better fault isolation, and improved continuous deployment.
Q144: SOLID stands for Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion principles, guiding towards more maintainable, understandable, and flexible software.
Q145: Lazy loading is a design pattern that delays loading of non-critical resources at page load time, reducing initial load time and page weight.
Q146: A load balancer distributes network or application traffic across multiple servers to enhance responsiveness and availability of applications.
Q147: Indexing speeds up data retrieval operations by effectively creating a smaller, faster version of the database table.
Q148: Event delegation refers to the practice of using a single event listener to manage all events of a specific type for child elements.
Q149: A closure is a function that remembers its outer variables and can access them.
Q150: Docker allows for packaging applications in containers, facilitating consistent deployment across different environments.
Q151: Focus on data partitioning, replication for fault tolerance, consistency models, and handling node failures.
Q152: Use a recursive function to check the height of each subtree; return false if the difference is more than one.
Q153: Consider efficient hashing, collision resolution, database schema, scalability, and API rate limiting.
Q154: Use collaborative filtering, content-based filtering, or hybrid methods; consider scalability and real-time processing.
Q155: Use two heaps (max heap for lower half, min heap for upper half) to maintain the median.
Q156: Discuss leader election, log replication, safety, and how Raft achieves consensus in a distributed system.
Q157: Use strategies like caching, edge locations, load balancing, and optimizing routing and data compression.
Q158: Consider websocket protocol for real-time communication, efficient message broadcasting, and scalable backend architecture.
Q159: Understand memory management concepts like mark-and-sweep, reference counting, and generational collection.
Q160: Focus on system architecture, push vs. pull models, handling peak loads, database optimization, and message queuing.

Q161: Focus on the three-way handshake, congestion control (like TCP Fast Open, and CUBIC), and optimizing for reduced latency.
Q162: Implement with fine-grained locking or lock-free techniques to ensure thread safety and high concurrency.
Q163: Utilize a min-heap to keep track of the K largest elements, ensuring efficient insertion and extraction.
Q164: Use graph-based algorithms focusing on eigenvector calculation and iterative approaches.
Q165: Use token bucket or leaky bucket algorithms, consider distributed storage for scalability.
Q166: Focus on indexing, query optimization, using caching, database sharding, and efficient schema design.
Q167: Implement OAuth for third-party integrations, use JWT for stateless authentication, and ensure protection against common security vulnerabilities.
Q168: Optimize for write-heavy loads, use time-based partitioning, efficient indexing, and consider data compression techniques.
Q169: Focus on cryptographic hashing, decentralized consensus algorithms (like Proof of Work), and the maintenance of a distributed ledger.
Q170: Use object-oriented design principles, focus on efficiently handling different vehicle sizes, and optimizing space usage.
Q171: Consider time series analysis, regression models, and reinforcement learning; pay attention to features and data preprocessing.
Q172: Discuss memory pool allocation, handling fragmentation, and optimizing for allocation/deallocation speed.
Q173: Focus on handling high network traffic, efficient state synchronization, latency reduction, and scalability.
Q174: Address challenges in data distribution, replication, fault tolerance, consistency, and performance.
Q175: Implement distributed searching algorithms like MapReduce for scalability and efficiency.
Q176: Utilize stream processing frameworks (like Apache Kafka, Spark Streaming), ensure fault tolerance, and manage backpressure.
Q177: Focus on low latency, high throughput, reliable data feeds, order execution systems, and concurrent algorithms.
Q178: Use convolutional neural networks, pay attention to dataset quality and preprocessing, and handle class imbalances.
Q179: Implement advanced AI techniques like Monte Carlo Tree Search, deep learning, and reinforcement learning.
Q180: Use machine learning for anomaly detection, implement rule-based systems for known fraud patterns, ensure real-time processing.
Q181: Discuss vertex-centric computation, message passing between nodes, and optimizations for large-scale processing.
Q182: Focus on CDN usage, adaptive bitrate streaming, content caching strategies, and handling peak traffic loads.
Q183: Use NLP for parsing resumes, implement ranking algorithms, and optimize for search and matching efficiency.
Q184: Consider consistency, data partitioning, eviction policies, and fault tolerance in distributed caching.
Q185: Address latency, data loss, and bandwidth issues; optimize for long-distance and high-latency networks.
Q186: Implement algorithms considering real-time obstacle avoidance, dynamic path adjustments, and efficient routing.
Q187: Utilize message queues (like Kafka, RabbitMQ), ensure fault tolerance, and implement load balancing.
Q188: Use NLP techniques like tokenization, parsing, and deep learning models for understanding and generating responses.

Q189: Implement statistical models or machine learning algorithms to detect unusual patterns indicative of anomalies.
Q190: Focus on handling large-scale data influx, real-time processing, data storage, and analytics.
Q191: Discuss lexical analysis, parsing, syntax tree generation, semantic analysis, and code generation.
Q192: Consider efficient data structures for text storage (like gap buffers), and algorithms for syntax parsing.
Q193: Focus on handling high-volume traffic, data analytics, ad targeting algorithms, and ensuring low-latency responses.
Q194: Use NLP for text analysis, implement classification algorithms, and consider the challenge of unstructured data.
Q195: Focus on transaction isolation levels, indexing strategies, query optimization, and database sharding.
Q196: Address virtualization technologies, resource allocation, security, and remote access protocols.
Q197: Utilize streaming data processing, machine learning for pattern recognition, and efficient data storage solutions.
Q198: Discuss quantum computing principles, qubit manipulation, and specific algorithms like Grover's or Shor's algorithm.
Q199: Focus on security protocols, offline capabilities, user authentication, and low-resource optimizations.
Q200: Implement distributed processing, efficient storage, and consider ML techniques for feature extraction.