Питання до іспиту

Алгоритми та структури даних

1. **Проблема вибору структури даних:** Опишіть, як вибір між **масивом**, **зв'язаним списком** та **хеш-таблицею** впливає на загальну продуктивність (часова та просторова складність) інформаційної системи. Наведіть конкретний приклад, де використання хеш-таблиці є критично важливим для швидкодії.
2. **Аналіз складності:** Поясніть поняття **часової складності алгоритмів** (O-нотація) та її значення для архітектора ІС. Порівняйте алгоритм із O(n) та алгоритм із O(log n) у контексті обробки великих обсягів даних.
3. **Порівняння структур для пріоритету:** Порівняйте використання **бінарної купи (Binary Heap)** та **відсортованого зв'язаного списку** для реалізації **Черги з пріоритетом** у системі. Яка структура даних є кращою з точки зору часової складності операцій **вставки** та **вилучення** елемента з найвищим пріоритетом, і чому?
4. **Алгоритми, розрахунок ресурсів:** Поясніть, як архітектор ІС використовує аналіз часової складності (наприклад, O(n) або O(n^2)) для прогнозування потреби системи в ресурсах (CPU, пам'ять) при зростанні обсягу даних. Наведіть приклад, де неефективний алгоритм стає причиною архітектурного 'вузького місця'.
5. **Хеш-таблиці та колізії:** Поясніть, що таке **хеш-колізії** та **коефіцієнт заповнення (load factor)** у хеш-таблицях. Які архітектурні методи використовуються для вирішення колізій

(наприклад, метод ланцюжків чи відкрита адресація) і як вони впливають на теоретичну продуктивність пошуку O(1)?

Кешування

6. **Рівні та стратегії кешування:** Опишіть архітектурну модель багаторівневого кешування в розподіленій системі (наприклад, кеш браузера, CDN/зворотне проксі, кеш додатку, кеш БД). Які переваги та недоліки має кожен рівень?
7. **Вибір політики витіснення:** Порівняйте та протиставте політики витіснення кешу **LRU** (Least Recently Used) та **LFU** (Least Frequently Used). В яких сценаріях (типовий робочий набір даних) краще використовувати кожну з них?
8. **Проблема узгодженості кешу:** Поясніть проблему **когерентності (узгодженості) кешу** в розподілених системах. Опишіть два основні підходи до її вирішення (наприклад, **Cache-Aside** з TTL, **Write-Through** або **Invalidation Strategy**).
9. **Безпека та кешування:** Які **ризики безпеки** пов'язані з кешуванням даних (наприклад, кешування конфіденційних даних, атаки на кеш (Cache Poisoning) або неправильне очищення)? Які архітектурні механізми (наприклад, ізоляція, контроль доступу, TTL для чутливих даних) слід застосовувати для захисту кешованої інформації?

Контейнеризація та оркестрування

10. **Контейнеризація vs. Віртуалізація:** Проаналізуйте ключові архітектурні відмінності між **контейнеризацією** (наприклад,

Docker) та **повною віртуалізацією** (наприклад, VM). Чому контейнеризація стала домінуючим підходом для розгортання мікросервісів?

11. **Роль Kubernetes:** Опишіть основні функції та архітектурні компоненти **Kubernetes** (Master/Control Plane, Node, Pod, Service, Deployment). Як Kubernetes забезпечує **самовідновлення** та **автоматичне масштабування** додатків?
12. **CI/CD з контейнерами:** Поясніть, як контейнеризація інтегрується в сучасний **конвеєр CI/CD** (Continuous Integration/Continuous Deployment) і як це впливає на швидкість та надійність розгортання ІС.
13. **Стійкість даних:** Яким чином вирішується проблема **збереження стійких даних** (**persistent storage**) у контейнеризованих середовищах, які за своєю природою є тимчасовими? Опишіть концепцію **Volumes** (томи) у Docker/Kubernetes.
14. **Контейнери, безпека:** Опишіть ключові проблеми безпеки в середовищах контейнеризації (наприклад, **Docker**) та оркестрування (наприклад, **Kubernetes**). Які архітектурні практики та інструменти слід використовувати для їх мінімізації (наприклад, сканування образів, політики мережі)?

Бази даних

15. **ACID та CAP:** Поясніть принципи **ACID** (для реляційних БД) та теорему **CAP** (для розподілених систем). Чому розподілені NoSQL системи часто жертвують 'C' (Consistency) на користь 'A' (Availability) та 'P' (Partition Tolerance)?

16. **Порівняння SQL та NoSQL:** Порівняйте реляційні (SQL) та нереляційні (NoSQL) бази даних за критеріями **схеми даних**, **масштабованості** та **моделі транзакцій**. В яких архітектурних сценаріях доцільно використовувати документо-орієнтовані NoSQL БД?
17. **Схеми БД (Lab 5-6):** Опишіть різницю між **концептуальною**, **логічною** та **фізичною** схемами БД. Яку роль відіграє кожна схема в процесі проєктування ІС, і які абстракції вона надає?
18. **Масштабування БД:** Поясніть архітектурні стратегії **горизонтального (sharding)** та **вертикального масштабування** реляційних БД. Які проблеми виникають при шардуванні і як їх можна мінімізувати (наприклад, транзакції між шардів)?
19. **Бази даних, індексація:** Детально поясніть принцип роботи **індексації** в реляційних СУБД (наприклад, B-tree індекс). Обговоріть компроміс, який виникає між швидкістю читання (SELECT) та швидкістю запису/оновлення (INSERT/UPDATE) при використанні індексів.

Хмарні провайдери

20. **Модель спільної відповідальності (Shared Responsibility Model):** Поясніть концепцію **Моделі спільної відповідальності** у хмарних обчисленнях. Наведіть конкретні приклади, які сфери відповідальності (за безпеку, патчі, конфігурацію) залишаються за клієнтом, а які – за провайдером, для моделей **IaaS** та **PaaS**.

21. **Vendor Lock-in та стратегії його уникнення:** Що таке **Vendor Lock-in (прив'язка до постачальника)** у контексті хмарних провайдерів? Опишіть архітектурні стратегії та технології (наприклад, використання контейнерів, інфраструктура як код, абстракція даних), які допомагають мінімізувати цей ризик.
22. **Serverless (FaaS):** Що таке архітектура **Serverless (FaaS)**? Обговоріть її ключові переваги (наприклад, автоматичне масштабування, оплата за використання) та архітектурні виклики (наприклад, 'холодний старт', Vendor Lock-in).
23. **Хмара, еластичність та відмовостійкість:** Поясніть, як хмарні сервіси (наприклад, AWS Auto Scaling, Azure VM Scale Sets) забезпечують **еластичність** та **відмовостійкість** ІС. Яка архітектурна відмінність між цими двома поняттями?

Архітектура розподілених систем. REST та gRPC

24. **Принципи REST:** Опишіть ключові архітектурні принципи **REST** (Representational State Transfer), включаючи **Statelessness** (Безстатутність) та **Uniform Interface** (Єдиний інтерфейс). Як ці принципи сприяють масштабованості та надійності розподілених систем?
25. **Відмінності REST vs. gRPC:** Здійсніть детальний архітектурний та технічний порівняльний аналіз **REST (JSON/HTTP 1.1)** та **gRPC (Protocol Buffers/HTTP/2)**. Зосередьтеся на таких аспектах, як **продуктивність**, **формат даних**, **тип зв'язку** та **генерування коду**.

26. **Idempotency та HTTP-методи:** Поясніть концепцію **ідемпотентності** у контексті RESTful API. Наведіть приклади HTTP-методів, які є ідемпотентними, і тих, які не є, та поясніть архітектурне значення цього для надійних розподілених систем.
27. **Синхронна та Асинхронна комунікація:** Порівняйте синхронні (наприклад, REST, gRPC) та асинхронні (наприклад, черги повідомлень, Event Bus) підходи до комунікації між сервісами. В яких сценаріях використання асинхронності є необхідним архітектурним рішенням?
28. **Маршалінг та Protobuf:** Яка роль **маршалінгу** та **демаршалінгу** даних в архітектурі gRPC? Поясніть, як використання **Protocol Buffers (protobuf)** сприяє високій продуктивності, крос-мовній сумісності та зменшенню обсягу переданих даних порівняно з текстовими форматами (JSON/XML).

SOA та Мікросервісна архітектура

29. **SOA vs. Мікросервіси:** Поясніть архітектурну еволюцію від **Моноліту** до **Сервіс-орієнтованої архітектури (SOA)**, а потім до **Мікросервісної архітектури**. Яку роль у цій еволюції відігравав **Enterprise Service Bus (ESB)** у SOA та чим він замінюється у мікросервісах?
30. **Проблеми розподілених транзакцій:** Обговоріть проблеми, пов'язані з реалізацією **розподілених транзакцій** (тобто, транзакцій, що охоплюють кілька мікросервісів).

Опишіть шаблон **Saga** як альтернативний підхід до забезпечення узгодженості даних.

31. **API Gateway:** Яку критичну архітектурну роль виконує **API Gateway** у мікросервісній архітектурі? Назвіть щонайменше чотири функції (крім простого роутингу), які він може виконувати.
32. **Залежності та комунікація в мікросервісах:** Опишіть проблему **сильної зв'язаності** (*tight coupling*) між мікросервісами. Як архітектор може мінімізувати цю проблему, використовуючи принципи **Domain-Driven Design (DDD)** та **асинхронну комунікацію**?
33. **Шаблон Circuit Breaker:** Опишіть шаблон **Circuit Breaker (Автоматичний вимикач)**. Яку проблему **каскадних збоїв (cascading failures)** він вирішує в мікросервісній архітектурі, і які три основні **стани** має цей компонент, а також, як він переходить між ними?

Діаграми

34. **UML, діаграми взаємодії**: Опишіть призначення та ключові елементи діаграми Послідовності (Sequence Diagram) UML. Як архітектор використовує цю діаграму для документування та валідації сценаріїв міжсервісної комунікації в розподіленій системі?