CS Fundamentals

> Basic knowledge-base for computer science, especially for technical interviews.

Data Structure

1. Hash Table
    - hash(key) % arrayLength
    - Collision (If we got collision, keys are required to stored together with value)
        - Chaining: linked list or balanced BST
        - Open Addressing: find another available spot

    - Dynamic Resizing
        - Resize when load factor too large (e.g. Java HashMap 0.75, Python dict 2/3)
        - Requires copying all entries

2. Array v.s. Linked List

	Array	**Linked List**
Access	Random	Sequential
Memory Structure	Continuous memory location	Any available location (extra space for pointers)
Insert/Delete	Shift elements	Only change pointers
Memory Allocation	Compile time	Run time

Operating Systems

1. Process v.s. Thread

Process	Thread
Run in separated memory space	Run in shared memory space (heap, data, code, file descriptor **EXCEPT Stack and Registers**)
It has its own copy of data segment of the parent process	It has direct access to data segment of its process
Interprocess Communication (IPC)	Directly communication with threads of its process

2. Stack v.s. Heap

Stack	Heap
Very fast access	*Relatively* slow access
Local variables only	Variables can be accessed globally
Limit on stack size (OS-dependent)	No limit on memory size
Space managed by CPU, no memory fragmented	No guaranteed space efficiency, memory may be fragmented. You **MUST** manage memory (allocate/free variables)
Variables can not be resized	Variables can be resized with `realloc()`

Database

1. Inner Join v.s. Outer Join

    - **Inner Join**: "Intersection" of two tables, must be some common attributes and rows.
    - **Outer Join**: *Inner Join + no corresponding matching rows*, for those unshared data attributes, use NULL

        - Left/Right Outer Join: All data in left/right table and shared data.
        - Full Outer Join: All data in both tables. Un shared attributes got NULL

2. SQL v.s NoSQL

SQL	NoSQL
Structured data, Schema, Tables	Semi or no structure. Key-Value/Document based/Column-oriented...
ACID	Eventually Consisitency
well designed can perform better than bad-designed NoSQL	Usually faster as no JOINs or complex SQL queries
	Simpler data model can be easier to scale
Better data integrity, constrains like foreign key or so	

- Usually we choose SQL(RDMS) for logical related discrete data, essential data integrity/consistency, standards-based proven tech and better community support
- Usually we choose NoSQL for bad structured data, speed and scalability is more important than consistency.

Computer Networks & Web

1. TCP v.s. UDP

	TCP	UDP
Connection	Connection Oriented. Three way handshake	Connectionless
Overhead	Large	Small
Order	Guaranteed	No guarantee. Best effort
Reliability	Guaranteed	No guarantee. Best effort
Error Checking	Checksum and error recovery	Checksum but discard error message directly
Data Flow Control	Yes	No
Usage	Require high reliability, transmission time is less critical	Needs fast and efficient transmission

2. What happens when visit *google.com*?

    - Browser looks up IP address for the domain name
        - browser cache -> OS host files -> router cache -> ISP DNS cache –> recursive search[root nameserver, .com/.org... nameserver, Wikipedia.com nameserver]
    - Browser initiates a TCP connection
    - Browser sends an HTTP GET request to the web server
        - May not be a web server, may be load balancer and then redirect to server
    - Web server handle the HTTP request, maybe also talk to DB
    - Web server sends back a HTTP response through TCP
    - Browser received HTTP response, may close the TCP connection, or reuse it for another request
    - Browser builds DOM tree and render
    - Browser sends GET requests for objects (img, CSS, JS...) embedded in HTML (Maybe requests to CDN)
    - User interaction. Browser sends further asynchronous requests (AJAX)

3. How to solve slowly website visiting problem?

    - Network
        - [Diagnose] Everyone or just you? **traceroute** show how long it takes at each hop from you to the server.
        - [Solution] Contact your ISP or just wait (most ISP will have it up within few hours)?
    - Client Side
        - [Diagnose] Serve a pure simple HTML page. If it is not slow, then the problem may not lie on network or server side.
        - [Solution]
            - No cache for IP? May be the time to query DNS
            - CSS/JS combination, compression, uglify; image compression, sprites; js async or delay
            - Content lazy loading
            - CDN for static resources
    - Server
        - Too many requests at a time? More servers, dispatcher, load balancer
        - Server down? Replica for backup
    - Database
        - [Diagnose] Load a static page without database interaction. If it is not slow, then the problem may be at database part.

        - [Solution] Query slow? Build index

4. GET v.s. POST

	GET	**POST**
Query Data	In URL	In HTTP body
Cache	CAN be cached	CANNOT be cached
Browser History	Remain in browser history	NOT remain in browser history
Bookmark	CAN be bookmarked	CANNOT be bookmarked
Size	Have data length restriction	No data length restriction

    - Never use GET for sensitive data like password
    - POST v.s PUT
        - PUT is to put a file or resource **EXACTLY** at the URI
        - POST sends data to specified URI and expects the resource there to **handle the request**. The web server can now determine what to do with the data

5. REST

    - **RE**presentational **S**tate **T**ransfer
    - Use nouns (resources) instead of verbs in SOAP-RPC
    - Use HTTP methods GET/POST/PUT/DELETE for what you want to do for the resources
    - Use URIs as identifiers for the resources
    - Use JSON or XML for state transfer between client and server. Server maintain no state for clients.