JOB SWITCH ROADMAP

120 concepts for your job switch

8 areas, 15 concepts each. Every concept explained in one line, the order to study each area, and two practice questions to test yourself.

How to use it: one area at a time, tick each concept only when you can explain it out loud in under a minute, then answer the two practice questions without notes.

@chalkesahil · chalkesahil.ai

AREA 01 / 08

DSA

Study order: Do arrays to binary search first, then trees and graphs, then DP and backtracking last.

01	**Arrays and strings**	Contiguous memory; most problems are a clever single pass.
02	**Hashing**	Trade memory for O(1) lookups: counts, seen-sets, index maps.
03	**Two pointers**	Walk from both ends or at two speeds to avoid nested loops.
04	**Sliding window**	Keep a moving range valid instead of recomputing it each step.
05	**Binary search**	Halve the search space on any monotonic condition, not just sorted arrays.
06	**Stacks**	Last in, first out: matching brackets, undo, expression parsing.
07	**Monotonic stack**	Keep the stack sorted to find next greater or smaller elements in O(n).
08	**Linked lists**	Pointer surgery: reverse, detect cycles, merge, find the middle.
09	**Trees**	Recursion on left and right; know preorder, inorder, postorder and level order.
10	**BST**	Left smaller, right larger; inorder traversal gives sorted order.
11	**Heaps**	Always get the min or max fast: top K, scheduling, merging sorted lists.
12	**Graphs (BFS, DFS)**	BFS for shortest steps, DFS for exploring and connected parts.
13	**Topological sort**	Order tasks so every dependency comes first; detects cycles too.
14	**Dynamic programming**	Solve overlapping subproblems once and reuse the answers.
15	**Backtracking**	Try a choice, recurse, undo: permutations, subsets, N-Queens.

Practice

1. Find the longest substring without repeating characters.
2. Return the K most frequent elements in an array.

AREA 02 / 08

System design

Study order: Learn the building blocks first, then practise one full design a day out loud.

01	**Load balancer**	Spreads requests across servers so no single one is overloaded.
02	**Caching**	Keep hot data in memory close to the user to cut latency and load.
03	**CDN**	Serves static files from servers near the user.
04	**Replication**	Copies of data on several machines for reads and failover.
05	**Sharding**	Split data across machines by a key so each holds a part.
06	**Consistent hashing**	Assign keys to servers so adding one moves only a few keys.
07	**Message queues**	Decouple producers and consumers; absorb spikes with async work.
08	**Rate limiter**	Caps requests per user or IP: token bucket, sliding window.
09	**CAP theorem**	During a network split you pick consistency or availability.
10	**SQL vs NoSQL**	Relations and transactions versus flexible schema and easy scaling.
11	**Indexing**	A sorted structure that makes lookups fast at the cost of slower writes.
12	**Idempotency**	Repeating a request has the same effect as doing it once; key for payments.
13	**API gateway**	One entry point for auth, routing, rate limits and logging.
14	**Monolith vs microservices**	One deployable versus many small services; know the trade-offs.
15	**Design a URL shortener**	The classic warm-up: ID generation, storage, redirects, caching.

Practice

1. Design a notification system that sends 10 million messages a day.
2. How would you make a payment API safe to retry?

AREA 03 / 08

Low-level design

Study order: OOP and SOLID first, then patterns, then full machine-coding problems.

01	**OOP pillars**	Encapsulation, abstraction, inheritance, polymorphism.
02	**SOLID**	Five rules that keep classes small, focused and easy to change.
03	**Composition vs inheritance**	Prefer has-a over is-a to avoid rigid class trees.
04	**Singleton**	Exactly one instance, shared; careful with threads and tests.
05	**Factory**	Create objects without the caller knowing the concrete class.
06	**Builder**	Construct complex objects step by step instead of huge constructors.
07	**Strategy**	Swap an algorithm at runtime behind one interface.
08	**Observer**	Subscribers get notified when something changes.
09	**Adapter**	Wrap an incompatible interface so it fits yours.
10	**Dependency injection**	Pass dependencies in instead of creating them inside.
11	**Class diagrams**	Sketch classes and relationships before writing code.
12	**Design a parking lot**	Spots, vehicles, tickets, pricing: the classic LLD round.
13	**Design Splitwise**	Users, groups, expenses and simplifying who owes whom.
14	**Design an elevator**	States, requests and a scheduling strategy.
15	**Thread-safe classes**	Locks, atomics and immutability when many threads share state.

Practice

1. Design the classes for a movie ticket booking app.
2. Refactor a class that breaks the single responsibility rule.

AREA 04 / 08

DBMS and SQL

Study order: Write SQL daily from the start; theory makes sense once you have run queries.

01	**Normalization**	Split tables to remove duplicate data and update anomalies.
02	**Keys**	Primary, foreign, unique and composite keys and what each guarantees.
03	**Joins**	Inner, left, right, full and self joins, and when to use each.
04	**GROUP BY and HAVING**	Aggregate rows into groups, then filter the groups.
05	**Subqueries**	A query inside a query; know when a join is faster.
06	**Window functions**	RANK, ROW_NUMBER, running totals without collapsing rows.
07	**Indexes**	B-tree lookups; which columns to index and what it costs.
08	**Query plans**	Read EXPLAIN to see scans, index use and join order.
09	**Transactions**	A group of operations that succeed or fail together.
10	**ACID**	Atomicity, consistency, isolation, durability.
11	**Isolation levels**	Read uncommitted to serializable, and the anomalies each allows.
12	**Locks and deadlocks**	Row and table locks, and how two transactions block each other.
13	**Stored procedures**	Logic that runs inside the database.
14	**N+1 queries**	One query per row from an ORM loop; fix with joins or batching.
15	**Pagination**	Offset versus cursor-based paging and why cursors scale better.

Practice

1. Find the second highest salary in each department.
2. Why can adding an index make inserts slower?

AREA 05 / 08

Operating systems

Study order: Processes and threads first; memory and scheduling make sense after.

01	**Process vs thread**	Separate memory versus shared memory inside one process.
02	**Context switch**	Saving and restoring CPU state to run something else.
03	**CPU scheduling**	FCFS, round robin, priority: who runs next and why.
04	**Mutex**	A lock that lets only one thread into a critical section.
05	**Semaphore**	A counter that allows a limited number of threads in.
06	**Deadlock**	Threads waiting on each other forever; know the four conditions.
07	**Race condition**	The result depends on timing because shared state is unprotected.
08	**Virtual memory**	Each process sees its own address space, mapped to real memory.
09	**Paging**	Memory split into fixed pages mapped by a page table.
10	**Page faults**	The page is not in RAM, so the OS loads it from disk.
11	**Memory leaks**	Allocated memory that is never freed keeps growing.
12	**Threads vs async**	Parallel threads versus one thread juggling many waiting tasks.
13	**System calls**	How a program asks the kernel to open files, read, allocate.
14	**File descriptors**	Small integers the OS gives you for open files and sockets.
15	**Producer-consumer**	The classic bounded-buffer sync problem.

Practice

1. Explain what happens when two threads increment the same counter.
2. How does the OS give each process its own memory?

AREA 06 / 08

Computer networks

Study order: Start with the URL question; it touches almost every other concept.

01	**OSI model**	Seven layers from physical wires to applications.
02	**TCP vs UDP**	Reliable ordered delivery versus fast, fire-and-forget packets.
03	**TCP handshake**	SYN, SYN-ACK, ACK before any data flows.
04	**HTTP vs HTTPS**	Plain text versus encrypted with TLS.
05	**TLS**	Certificates and key exchange that make HTTPS private and verified.
06	**DNS**	Turns a domain name into an IP address.
07	**What happens when you type a URL**	DNS, TCP, TLS, HTTP request, server, response, render.
08	**REST vs gRPC**	JSON over HTTP versus binary contracts over HTTP/2.
09	**WebSockets**	A long-lived two-way connection for live updates.
10	**Cookies vs sessions vs JWT**	Where login state lives and the trade-offs of each.
11	**CORS**	Browser rules for which other origins may call your API.
12	**Proxies**	Forward proxies act for clients; reverse proxies sit in front of servers.
13	**Latency vs bandwidth**	How long one trip takes versus how much fits through.
14	**HTTP status codes**	2xx success, 3xx redirect, 4xx client error, 5xx server error.
15	**Keep-alive**	Reuse one connection for many requests.

Practice

1. Walk through everything that happens when you open a website.
2. When would you choose WebSockets over polling?

AREA 07 / 08

Backend fundamentals

Study order: Build one small API that uses all of these; interviews love real examples.

01	**API design**	Clear resources, verbs, versioning and error shapes.
02	**Authentication vs authorization**	Who you are versus what you are allowed to do.
03	**OAuth**	Let users log in or share access through another provider.
04	**Redis caching**	In-memory store for hot reads, sessions and counters.
05	**Background jobs**	Move slow work like emails out of the request path.
06	**Retries and backoff**	Retry failed calls with growing waits to avoid storms.
07	**Logging**	Structured logs that let you trace one request end to end.
08	**Monitoring**	Metrics and alerts for errors, latency and saturation.
09	**Unit vs integration tests**	Test one function in isolation versus parts working together.
10	**Docker**	Package the app and its dependencies into one container.
11	**CI/CD**	Every push is built, tested and deployed automatically.
12	**Environment config**	Keep secrets and settings out of code, per environment.
13	**Error handling**	Fail clearly, return useful errors, never leak internals.
14	**File uploads**	Stream large files, validate them, store them in object storage.
15	**Webhooks**	Another service calls your endpoint when something happens.

Practice

1. How would you stop a third-party API outage from taking your app down?
2. Design the auth flow for a mobile app.

AREA 08 / 08

Interview stories

Study order: Write each story down once in STAR form, then say it out loud until it fits in 2 minutes.

01	**Tell me about yourself**	A 60-second story: now, how you got here, what you want next.
02	**Current project in 2 minutes**	The problem, your part, the stack, one result with a number.
03	**Hardest bug**	Symptom, how you dug in, the root cause, what you changed after.
04	**A conflict**	Disagree with facts, listen, land on a decision, keep the relationship.
05	**A failure**	Own it, the cost, what you learned, what you do differently now.
06	**A decision you owned**	The options, the trade-off you chose, how it turned out.
07	**Why you're switching**	Pull towards growth, never complaints about your current team.
08	**Why this company**	One specific reason from their product, tech or team.
09	**STAR method**	Situation, task, action, result: keeps every answer tight.
10	**Questions to ask them**	Team, on-call, how success is measured in the first 6 months.
11	**Notice period answer**	State it honestly and whether a buyout or early release is possible.
12	**Salary expectations**	Give a researched range, anchored on the role, not your current pay.
13	**Counter offer**	Decide before you resign whether you would ever stay.
14	**Offer comparison**	Compare total pay, growth, team and stability, not just the base.
15	**Joining date**	Confirm in writing; keep your notice period promises.

Practice

1. Tell me about a time you disagreed with your manager.
2. Why should we hire you over someone with more experience?