LinkedIn

Software Engineer

Complete Interview Guide 2025

Coding · System Design · Behavioral · Solutions

Based on 300+ real candidate interviews (2024–2025) | Glassdoor · LeetCode · IGotAnOffer

📌 Section 1 — Interview Overview & Process

LinkedIn's Software Engineer interview process is thorough, centralized, and spans approximately 3–6 weeks. Candidates are not interviewed by specific teams — team matching happens after the offer. The process is evaluated on a 4-point scale per round, with 3 being a passing score.

1.1 Interview Stages at a Glance

Stage	Duration	Format	Weight
Recruiter Phone Screen	15–30 min	Phone / Video	Qualifier
Technical Phone Screen	60 min	CoderPad / Collabedit	High
Mid-Process Recruiter Call	30 min	Phone	Retention / Timeline
Onsite — Coding Round 1	45–60 min	Virtual Whiteboard	High
Onsite — Coding Round 2	45–60 min	Virtual Whiteboard	High
Onsite — System Design	45–60 min	Virtual Whiteboard	Very High
Onsite — Behavioral / Cultural	45–60 min	Video Call with HM	High
Onsite — Team Collaboration	45–60 min	Video Call	Medium
Hiring Committee Review	1–2 weeks	Internal	Final Decision

1.2 Key Facts (Sourced from Candidates)

✅ Average hiring time: 23 days from application to offer

✅ Difficulty rating: 3.1 / 5 — moderately to highly difficult

✅ AI usage during interviews is strictly prohibited

✅ Question bank is relatively small and stable — LeetCode-tagged questions are reliable prep

✅ Onsite lasts 5–6 hours with up to 7 rounds for borderline candidates

✅ LinkedIn is under Microsoft umbrella — some cultural alignment expected

✅ Compensation ranges from $134K (L2) to $2.15M+ (Distinguished Engineers)

1.3 Salary Bands by Level

Level	Total Compensation (USD)
L2 — Entry-Level SWE	$134,000 – $200,000
L3 — Software Engineer	$200,000 – $280,000
L4 — Senior SWE	$302,000 – $400,000
L5 — Staff SWE	$448,000 – $600,000
L6 — Principal / Distinguished	$700,000 – $2,150,000+

📞 Section 2 — Recruiter Phone Screen

The first round is a 15–30 minute phone call. The recruiter checks qualifications, discusses your resume, and explains the process. This is NOT a technical round — it is primarily a culture and communications fit check.

2.1 Commonly Asked Recruiter Questions

Q: "Tell me about yourself."

> 💡 Answer Strategy: Prepare a 2-minute structured pitch: current role → key accomplishments → why LinkedIn. Focus on impact, not just responsibilities.

Q: "Why do you want to work at LinkedIn?"

> 💡 Answer Strategy: Reference LinkedIn's mission ('create economic opportunity for every member'), specific products you use, or engineering blog posts you've read.

Q: "What are your salary expectations?"

> 💡 Answer Strategy: Do NOT name a number. Say: 'I'm focused on finding the right fit — I'm happy to discuss compensation once I understand the role better.'

Q: "Where else are you interviewing?"

> 💡 Answer Strategy: Be vague. Don't reveal competing offers prematurely. If you have tight timelines later, share them on the mid-process recruiter call.

Q: "Walk me through a challenging project."

> 💡 Answer Strategy: Use STAR format. Focus on scale, ambiguity, and your specific contribution.

Q: "What's your experience with large-scale systems?"

💡 Answer Strategy: Quantify scale: millions of users, terabytes of data, thousands of QPS. Show you've operated at LinkedIn-relevant scale.

Section 3 — Technical Phone Screen

A 60-minute video call on CoderPad or Collabedit with 1–2 interviewers. Expect 1–2 coding questions ranging from Easy to Medium difficulty. The interviewers spend 5–10 minutes on your background before diving into code. They are generally willing to give hints.

3.1 Format & What to Expect

- Platform: CoderPad, Collabedit, or HackerRank
- Duration: 60 minutes total — ~10 min intro + ~40 min coding + ~10 min Q&A
- Questions: 1–2 LeetCode-style problems (Easy + Medium, or two Mediums)
- Topics: Arrays, strings, linked lists, trees, hashmaps, sorting
- Tip: Think aloud. Interviewers value communication as much as correctness

3.2 Real Phone Screen Questions (Reported by Candidates)

Q1. Valid Parentheses

Easy | Stack

Problem Statement

Given a string s containing only '(', ')', '{', '}', '[', ']', determine if the input string is valid.

Valid if: every opening bracket is closed by the correct type, and in the correct order.

Example

```
Input:  s = "()[]{}"
Output: true


Input:  s = "(]"
Output: false
```

🔍 Approach

Use a stack. Push open brackets. For each closing bracket, check if the top of the stack is the matching opener. Return true if the stack is empty at the end.

⏱ **Complexity: Time: O(n) | Space: O(n)**

☕ Java Solution

```
public boolean isValid(String s) {
    Stack<Character> stack = new Stack<>();
    for (char c : s.toCharArray()) {
        if (c == '(' || c == '[' || c == '{') {
            stack.push(c);
        } else {
            if (stack.isEmpty()) return false;
            char top = stack.pop();
            if (c == ')' && top != '(') return false;
            if (c == ']' && top != '[') return false;
            if (c == '}' && top != '{') return false;
        }
    }
    return stack.isEmpty();
}
```

Q2. Search in Rotated Sorted Array

Medium | Binary Search

Problem Statement

A sorted array of distinct integers has been rotated at an unknown pivot.

Given nums and target, return the index if target is found, or -1 otherwise.

Must run in O(log n) time.

Example

```
Input:  nums = [4,5,6,7,0,1,2], target = 0
Output: 4


Input:  nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Approach

Modified binary search. At each step, determine which half is sorted. Check if target lies in the sorted half to decide which side to continue searching.

Complexity: **Time: O(log n) | Space: O(1)**

Java Solution

```
public int search(int[] nums, int target) {

    int left = 0, right = nums.length - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (nums[mid] == target) return mid;

        // Left half is sorted

        if (nums[left] <= nums[mid]) {

            if (target >= nums[left] && target < nums[mid])

                right = mid - 1;

            else

                left = mid + 1;

        } else { // Right half is sorted

            if (target > nums[mid] && target <= nums[right])

                left = mid + 1;

            else

                right = mid - 1;

        }

    }

    return -1;

}
```

Q3. Find Leaves of Binary Tree

Medium | DFS / Tree

Problem Statement

Given the root of a binary tree, collect the tree's nodes as if you were doing:

- Collect and remove all leaf nodes

- Repeat until the tree is empty

Return a list of lists of node values in the order they are collected.

💡 Example

```
Input:  root = [1,2,3,4,5]
Output: [[4,5,3],[2],[1]]
```

🔍 Approach

Use DFS. Each node's 'height' (0 for leaves) determines which result bucket it goes into. Recursively compute height = 1 + max(left height, right height).

⏱ **Complexity: Time: O(n) | Space: O(n)**

☕ Java Solution

```
public List<List<Integer>> findLeaves(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    dfs(root, result);
    return result;
}
private int dfs(TreeNode node, List<List<Integer>> result) {
    if (node == null) return -1;
    int leftH  = dfs(node.left,  result);
    int rightH = dfs(node.right, result);
    int height = 1 + Math.max(leftH, rightH);
    if (result.size() == height) result.add(new ArrayList<>());
    result.get(height).add(node.val);
    return height;
}
```

Q4. Shortest Word Distance

Easy | Array (LinkedIn Special)

📋 Problem Statement

Given an array of strings wordsDict and two different strings word1 and word2,

return the shortest distance between occurrences of word1 and word2 in the list.

💡 Example

```
Input:  ["practice","makes","perfect","coding","makes"],
         word1 = "coding", word2 = "practice"
Output: 3
```

🔍 Approach

Single pass: track the last seen index of word1 and word2. Whenever either is encountered, update its index and compute the absolute distance between the two tracked indices.

⏱ **Complexity:** **Time: O(n) | Space: O(1)**

☕ Java Solution

```java
public int shortestDistance(String[] wordsDict, String word1, String word2) {

    int i1 = -1, i2 = -1, minDist = Integer.MAX_VALUE;

    for (int i = 0; i < wordsDict.length; i++) {

        if (wordsDict[i].equals(word1))      i1 = i;

        else if (wordsDict[i].equals(word2)) i2 = i;

        if (i1 != -1 && i2 != -1)

            minDist = Math.min(minDist, Math.abs(i1 - i2));

    }

    return minDist;

}
```

🖥️ Section 4 — Onsite Coding Rounds

The onsite coding interviews consist of 2 rounds, each 45–60 minutes. Each round typically contains 1 Medium + 1 Hard problem with 40 minutes to solve both. Interviewers grade on a 4-point scale (3 = pass). They want to see clear thinking, clean code, and optimization discussion.

4.1 What Interviewers Evaluate

Evaluation Dimension	Description
Correctness	Does the solution handle all edge cases and produce correct output?
Code Quality	Is the code clean, readable, and well-structured?
Optimization	Can you identify and improve time/space complexity?
Communication	Do you explain your thinking clearly throughout?
Problem Decomposition	Do you break the problem into manageable steps?

4.2 Onsite Coding Questions (Medium)

Q5. LRU Cache	**Hard | Design + LinkedList + HashMap**

📋 Problem Statement

Design and implement a data structure for a Least Recently Used (LRU) cache.

It should support: get(key) → O(1) and put(key, value) → O(1).

When capacity is exceeded, evict the least recently used item.

💡 Example

```
LRUCache cache = new LRUCache(2);

cache.put(1, 1);    // cache = {1=1}

cache.put(2, 2);    // cache = {1=1, 2=2}

cache.get(1);       // returns 1

cache.put(3, 3);    // evicts key 2 → cache = {1=1, 3=3}

cache.get(2);       // returns -1 (not found)
```

🔍 Approach

Combine a HashMap (O(1) lookup) with a Doubly Linked List (O(1) insert/delete). Most recently used items go to the head. Evict from the tail when over capacity.

⏱ **Complexity:** **Time: O(1) for get/put | Space: O(capacity)**

☕ Java Solution

```
public class LRUCache {
    class Node { int key, val; Node prev, next; Node(int k,int v){key=k;val=v;} }
    private Map<Integer,Node> map = new HashMap<>();
    private int cap;
    private Node head = new Node(0,0), tail = new Node(0,0);

    public LRUCache(int capacity) {
        cap = capacity;
        head.next = tail; tail.prev = head;
    }
    public int get(int key) {
        if (!map.containsKey(key)) return -1;
        Node n = map.get(key); remove(n); addFront(n); return n.val;
    }
    public void put(int key, int val) {
        if (map.containsKey(key)) remove(map.get(key));
        Node n = new Node(key, val); map.put(key, n); addFront(n);
        if (map.size() > cap) { map.remove(tail.prev.key); remove(tail.prev); }
    }
    private void remove(Node n) { n.prev.next = n.next; n.next.prev = n.prev; }
    private void addFront(Node n) {
        n.next = head.next; n.prev = head; head.next.prev = n; head.next = n;
    }
}
```

Q6. Merge K Sorted Lists

Hard | Heap / Divide & Conquer

📋 Problem Statement

You are given an array of k linked-lists, each sorted in ascending order.

Merge all lists into one sorted linked list and return it.

💡 Example

```
Input:  lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]
```

🔍 Approach

Use a min-heap (PriorityQueue) seeded with all k list heads. Repeatedly extract the minimum, append it to result, and push its next node back into the heap.

⏱ **Complexity: Time: O(N log k) | Space: O(k)**

☕ Java Solution

```
public ListNode mergeKLists(ListNode[] lists) {
    PriorityQueue<ListNode> pq =
        new PriorityQueue<>((a, b) -> a.val - b.val);
    for (ListNode node : lists) {
        if (node != null) pq.offer(node);
    }
    ListNode dummy = new ListNode(0), curr = dummy;
    while (!pq.isEmpty()) {
        ListNode node = pq.poll();
        curr.next = node;
        curr = curr.next;
        if (node.next != null) pq.offer(node.next);
    }
    return dummy.next;
}
```

Q7. Serialize and Deserialize Binary Tree

Hard | Tree / BFS

📋 Problem Statement

Design an algorithm to serialize a binary tree to a string and deserialize

that string back to the original tree structure.

(You can use any format you choose — your encode/decode just need to be inverses.)

💡 Example

```
Tree:     1
           / \
          2   3
             / \
            4   5
serialize → "1,2,3,null,null,4,5"
deserialize → original tree
```

🔍 Approach

BFS: serialize each node's value level by level, using 'null' for missing children. Deserialize by reading values into a queue and rebuilding the tree using BFS.

⏱ **Complexity:** **Time: O(n) | Space: O(n)**

☕ Java Solution

```java
public String serialize(TreeNode root) {

    if (root == null) return "";

    StringBuilder sb = new StringBuilder();

    Queue<TreeNode> q = new LinkedList<>();

    q.offer(root);

    while (!q.isEmpty()) {

        TreeNode node = q.poll();

        if (node == null) { sb.append("null,"); continue; }

        sb.append(node.val).append(",");

        q.offer(node.left); q.offer(node.right);

    }

    return sb.toString();

}


public TreeNode deserialize(String data) {

    if (data.isEmpty()) return null;

    String[] vals = data.split(",");

    TreeNode root = new TreeNode(Integer.parseInt(vals[0]));

    Queue<TreeNode> q = new LinkedList<>();

    q.offer(root);

    for (int i = 1; i < vals.length; i++) {

        TreeNode node = q.poll();

        if (!vals[i].equals("null")) {

            node.left = new TreeNode(Integer.parseInt(vals[i]));

            q.offer(node.left);

        }

        if (++i < vals.length && !vals[i].equals("null")) {

            node.right = new TreeNode(Integer.parseInt(vals[i]));

            q.offer(node.right);

        }

    }

    return root;

}
```

Q8. Max Stack

Hard | Design / Stack

📋 Problem Statement

Design a max stack that supports:

- push(val): push element onto the stack

- pop(): remove the top element

- top(): get the top element

- peekMax(): retrieve max element in the stack

- popMax(): remove and return the maximum element

All operations should be as efficient as possible.

💡 Example

```
MaxStack stack = new MaxStack();
stack.push(5);   // stack: [5]
stack.push(1);   // stack: [5,1]
stack.push(5);   // stack: [5,1,5]
stack.top();     // returns 5
stack.popMax();  // returns 5, stack: [5,1]
stack.top();     // returns 1
```

🔍 Approach

Use two stacks: one for all values, one tracking max at each level. For popMax, pop elements until max is found, then restore popped elements.

⏱ **Complexity: Time: O(n) popMax, O(1) others | Space: O(n)**

☕ Java Solution

```
class MaxStack {
    Stack<Integer> stack = new Stack<>();
    Stack<Integer> maxStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        maxStack.push(maxStack.isEmpty() ? x : Math.max(x, maxStack.peek()));
    }
    public int pop() {
        maxStack.pop();
        return stack.pop();
    }
    public int top()     { return stack.peek(); }
    public int peekMax() { return maxStack.peek(); }
    public int popMax() {
        int max = peekMax();
        Stack<Integer> tmp = new Stack<>();
        while (top() != max) tmp.push(pop());
        pop(); // remove the max
        while (!tmp.isEmpty()) push(tmp.pop());
        return max;
    }
}
```

🏗️ Section 5 — System Design Interview

The system design round is 45–60 minutes. For senior+ engineers, this carries very high weight. You will be asked to design a large-scale distributed system from scratch, often LinkedIn-themed. Strong candidates communicate trade-offs clearly and proactively address scalability, reliability, and security.

5.1 Top 6 System Design Questions Asked at LinkedIn

Question	Core Challenge
Design LinkedIn's News Feed	Most frequently asked. 1B+ users, personalized, low latency.
Design People You May Know (PYMK)	Graph-based, ML recommendation, mutual connections.
Design LinkedIn Messaging / InMail	Real-time, 1:1 and group chat, push notifications.
Design a Notification Service	Multi-channel (web, mobile, email), deduplication, rate limiting.
Design LinkedIn Job Search	Billions of documents, personalized ranking, fast query latency.
Design a URL Shortener (lnkd.in)	Analytics, expiration, redirect performance at scale.

5.2 Deep Dive: Design LinkedIn News Feed for 1 Billion Users

STEP 1: Clarify Requirements (first 5 minutes)

Functional: Show personalized posts from connections + followed entities. Support reactions, comments, resharing.

Non-functional: 1B+ users, sub-500ms latency, 99.99% availability, eventual consistency is acceptable.

Scale: Assume 500M DAU, average 100 connections/user, 10M posts/day.

STEP 2: High-Level Architecture

```
Client → CDN / Load Balancer

           ↓

     Feed API Service (stateless, horizontally scaled)

           ↓

  ┌─────────────────┬─────────────────────────────────┐

  │  Feed Generator  │       Ranking Service (ML)       │

  │ (Hybrid Push/Pull│  Feature Store + Ranker Model    │

  └─────────────────┴─────────────────────────────────┘

           ↓

  ┌──────────────────────────────────────────────────┐

  │  Cache Layer: Redis / Memcached (pre-computed feeds)│

  └──────────────────────────────────────────────────┘

           ↓

  ┌──────────────────────────────────────────────────┐

  │  Database: Espresso (NoSQL) + Kafka (event stream) │

  └──────────────────────────────────────────────────┘
```

STEP 3: Feed Generation Strategy (Push vs Pull Hybrid)

Strategy	Description	Trade-off
Pure Push (Fan-out on write)	Pre-compute and cache feeds for all followers on post creation	High write amplification for users with millions of followers
Pure Pull (Fan-out on read)	Compute feed on-demand when user opens app	High read latency — must query all connections
Hybrid (LinkedIn's approach)	Push for active users (<5K followers); pull for celebrities / inactive users	Best balance of latency vs infrastructure cost

STEP 4: Ranking — LinkedIn's Two-Pass Architecture

First Pass Rankers (FPRs): Multiple rankers each select ~10 candidates based on different signals.

→ FollowFeed FPR contributes ~80% of feed candidates.

→ Other FPRs: Job Feed, Viral Content, Sponsored Content, etc.

Second Pass Ranker (SPR/Feed Mixer): Combines all FPR outputs, re-ranks using a Viewer Feature Store

→ Feature Store: Indexes user activity signals (clicks, dwell time, connections interacted with).

→ Final ranked list sent to client via cache layer.

STEP 5: Data Storage

Data Type	Storage Solution
User profiles & posts	Espresso (LinkedIn's NoSQL) — horizontally scalable, eventual consistency
Graph (connections)	RocksDB / Graph DB — fast BFS for PYMK, 2nd-degree connections
Feed cache	Memcached / Redis — pre-computed user feeds, TTL-based invalidation
Event streaming	Apache Kafka — async fan-out, activity signals, analytics pipeline
Analytics / batch jobs	Hadoop + Azkaban — offline feature computation, model retraining

STEP 6: Scale & Reliability

Sharding: Shard Espresso by user_id. Shard Kafka topics by entity_id.

Geo-replication: Multiple data centers with local reads; use conflict-free replicated data types (CRDTs).

Rate limiting: Token bucket at the API gateway per user_id.

Security: TLS in transit, AES-256 at rest, RBAC for internal services, OAuth/mTLS between microservices.

Cache invalidation: When a post is liked/deleted/edited, invalidate affected feed caches via Kafka event.

5.3 Design: People You May Know (PYMK)

PYMK recommends new connections using mutual connections, shared employers, education, and interaction signals.

```
Core Algorithm: BFS on Connection Graph

For user U, expand 2-degree connections:

  1. Get all 1st-degree connections of U → Set A

  2. For each node in A, get their connections → Set B

  3. Score: score(V) = mutual_connections + employer_match + school_match +
interaction_weight

  4. Rank B by score, filter out A (already connected) and U



Data structure: Adjacency list stored in graph DB (RocksDB/Neo4j)

Scale: Pre-compute PYMK scores offline (Hadoop batch) for all users daily

Real-time: Update scores incrementally when new connections are formed (Kafka
event)
```

🧠 Section 6 — Behavioral & Cultural Interview

The behavioral round is with the Hiring Manager and lasts 45–60 minutes. LinkedIn evaluates against its core values: Transformation, Integrity, Collaboration, and Results. All answers should use the STAR method (Situation → Task → Action → Result) and be quantified wherever possible.

6.1 LinkedIn's Core Cultural Values

Value	What LinkedIn Looks For
Transformation	Innovating, driving change, not just maintaining status quo
Integrity	Honesty, transparency, ethical decision-making under pressure
Collaboration	Cross-functional partnership, elevating teammates, inclusive culture
Results	Data-driven execution, measurable outcomes, ownership mindset

6.2 Most Frequently Asked Behavioral Questions + STAR Answers

Q1: Tell me about a time you had to solve a problem without clear guidance.

S — Situation	I joined a team mid-sprint with a critical internal analytics tool that had zero documentation, legacy code, and two engineers already having left.
T — Task	I needed to understand the tool's purpose, identify pain points, and propose a redesign — with no handoff.
A — Action	I interviewed 8 internal stakeholders over 5 days, documented all workflows, created a prioritized feature gap analysis, and proposed a phased redesign. I built a proof-of-concept in 2 weeks using React + Node.
R — Result	Tool adoption increased 60% in 3 months. The redesign became the company standard for all internal dashboards. I presented findings to the VP of Engineering.

Q2: Describe a time when you received difficult feedback on your work.

S — Situation	After a major product release, my tech lead gave me tough feedback that my code lacked test coverage and my PR descriptions were unclear, impacting team velocity.
T — Task	I needed to address the feedback constructively and improve team code quality practices.
A — Action	I immediately added unit + integration tests to all my PRs, wrote a PR template for the team, and gave a lunch-and-learn on 'writing PRs that review themselves'. I asked for weekly check-ins with my lead.
R — Result	My next performance review rated me 'Exceeds Expectations' on collaboration. The PR template was adopted team-wide, reducing review cycles by 40%.

Q3: How do you prioritize tasks in a fast-paced environment?

S — Situation	During a high-traffic product launch week, I had 3 competing priorities: a critical P0 bug, a deadline-driven feature, and onboarding a new team member.
T — Task	I had to triage effectively, communicate clearly to stakeholders, and ensure nothing fell through the cracks.
A — Action	I used an impact/urgency matrix: P0 bug first (customer-facing), delegated onboarding documentation to a senior peer, negotiated a 2-day extension for the feature after communicating risk clearly to PM. I used daily standups to re-prioritize.
R — Result	P0 bug resolved in 4 hours. Feature shipped on revised timeline with full test coverage. New engineer was ramped up effectively with zero disruption to sprint velocity.

Q4: Describe a situation where you had to influence without authority.

S — Situation	I noticed our team's microservices used 5 different logging libraries, making debugging across services extremely difficult. No one 'owned' this problem.
T — Task	I wanted to standardize logging across all services but had no authority to mandate changes across 4 different teams.
A — Action	I created a one-pager comparing libraries with benchmark data, hosted a cross-team engineering review, and built a shared logging utility as a proof-of-concept. I addressed each team's specific objections individually.
R — Result	3 of 4 teams adopted the standard within 6 weeks. Mean time to debug production incidents dropped from 45 min to 12 min. The initiative was recognized at the engineering all-hands.

6.3 Staff/Senior Level Behavioral Questions (Reported June 2025)

1. How do you define a quality product or system?
2. Describe your strategy for code reviews — what do you look for?

3. A massive legacy application has no tests, poor monitoring, and slow response times. You just joined the team. What do you do first?
4. Rank in priority: bugs, no testing, no monitoring, slowness. Justify your ranking.
5. What are your principles of technical mentorship?
6. Tell me about a time you helped grow engineers on your team.
7. You have a great idea for a project you don't own. How do you move it forward?
8. What do you look for when determining the success of a service?

⚙️ Section 7 — Concurrency, OS & Computer Science Fundamentals

LinkedIn's phone screens and onsite rounds often include conceptual CS questions. These are especially common for backend and infrastructure roles.

7.1 Frequently Asked CS Concepts Questions

Q: "What is the difference between a process and a thread?"

```
Process: An independent program in execution with its own memory space (heap,
stack, data segment). Processes are isolated — one crash doesn't affect others.

Thread: A lighter-weight unit of execution within a process. Threads share the heap
and code segment but have their own stack. Context switching between threads is
cheaper than between processes.

Key insight: LinkedIn's microservices use multi-threading heavily within each
service for concurrency.
```

Q: "What are the pros and cons of different caching strategies?"

```
Cache-aside (Lazy Loading): App checks cache first; on miss, loads from DB and
updates cache.

  ✅ Only caches what's actually needed   ❌ Cache miss causes 2 round trips

Write-through: Write to cache and DB simultaneously.

  ✅ Cache always fresh   ❌ Write latency increase

Write-back (Write-behind): Write to cache first, flush to DB asynchronously.

  ✅ Low write latency   ❌ Risk of data loss if cache fails

Read-through: Cache automatically populates itself on miss.

  ✅ Simpler app code   ❌ Cold start penalty

LinkedIn primarily uses Memcached for feed caching and Redis for real-time data
(rate limiting, sessions).
```

Q: "Explain the differences between synchronous and asynchronous programming."

```
Synchronous: Operations execute sequentially. Each operation blocks until complete.

  - Simple to reason about, easier debugging

  - Poor performance under I/O-heavy workloads (thread blocking)


Asynchronous: Operations can proceed without waiting. Callbacks, futures, or event
loops handle completion.

  - Higher throughput for I/O-bound tasks

  - More complex code; risk of callback hell without proper patterns

  - Java: CompletableFuture, reactive frameworks (RxJava, Project Reactor)

LinkedIn uses async processing extensively via Kafka for fan-out operations.
```

7.2 Randomized HashSet (Reported August 2025)

Implement a data structure with insert(val), remove(val), and getRandom() — each in O(1) average time.

```java
class RandomizedSet {

    private Map<Integer, Integer> map = new HashMap<>(); // val → index in list

    private List<Integer> list = new ArrayList<>();

    private Random rand = new Random();


    public boolean insert(int val) {

        if (map.containsKey(val)) return false;

        list.add(val);

        map.put(val, list.size() - 1);

        return true;

    }


    public boolean remove(int val) {

        if (!map.containsKey(val)) return false;

        int idx = map.get(val);

        int last = list.get(list.size() - 1);

        list.set(idx, last);      // swap target with last

        map.put(last, idx);       // update last's index

        list.remove(list.size() - 1); // remove last

        map.remove(val);

        return true;

    }


    public int getRandom() {

        return list.get(rand.nextInt(list.size()));

    }

}
```

🗺️ Section 8 — Preparation Roadmap & Tips

8.1 6-Week Preparation Plan

Week	Focus	Daily Goal
Week 1	Arrays, Strings, HashMaps	5 LeetCode Easy + 2 Medium tagged LinkedIn
Week 2	Trees, Graphs, BFS/DFS	3 Medium + 1 Hard; serialize/deserialize tree
Week 3	DP, Sliding Window, Two Pointers	3 Medium per day; master Kadane, jump game patterns
Week 4	System Design	Deep-dive 2 designs/ day: news feed, PYMK, messaging
Week 5	Design Patterns + Concurrency	LRU cache, producer-consumer, thread-safe data structures
Week 6	Mock Interviews + Behavioral	2 full mock interviews; write 10 STAR stories

8.2 Critical Do's and Don'ts

Do ✅	Don't ❌
Think out loud throughout — LinkedIn values communication	Jump to code without clarifying the problem first
Practice LeetCode LinkedIn-tagged problems (they barely change)	Ignore behavioral prep — it's 1 full round
Know LinkedIn's products and tech blog deeply	Reveal your other offers or salary expectations early
Ask clarifying questions before designing or coding	Write code before discussing your approach
Time-box system design to 5 min req + 30 min design + 10 min deep-dive	Let the interviewer guide all the discussion passively
Prepare 10 STAR stories covering all LinkedIn values	Use generic answers — always make it specific and quantified

8.3 Resources

- 📘 LeetCode (LinkedIn company tag) — most reliable question source
- 📘 Grokking the System Design Interview (Educative) — for design round
- 📘 Tech Interview Handbook (techinterviewhandbook.org) — behavioral prep
- 📘 LinkedIn Engineering Blog (engineering.linkedin.com) — shows real system context
- 📘 interviewing.io — mock interview practice with FAANG engineers
- 📘 IGotAnOffer LinkedIn Guide — comprehensive process breakdown

Sources: Glassdoor (307 interviews) · LeetCode Discuss · IGotAnOffer · Prepfully · Educative · Interviewing.io · CodingInterview.com

Last Updated: February 2025 | Based on real candidate interview reports