Meritshot
EDUCATION

SDE I Questions Frequesntly asked in

amazon

Meritshot
EDUCATION

1) Two Sum (array, hash map)
Question: Given nums and target, return indices i,j with nums[i]+nums[j]=target.

Approach: Single pass hash map: store needed = target - x; check before insert to avoid self-pair.

Code:

```python
def two_sum(a, target):
    seen = {}
    for i, x in enumerate(a):
        if target - x in seen: return [seen[target - x], i]
        seen[x] = i
```

Mindset: Don’t double count; handle duplicates; confirm no-solution behavior if unspecified.

2) Longest Substring Without Repeating (sliding window)

Approach: Move right pointer; maintain last index map; shrink left when duplicate seen.

```python
def lengthOfLongestSubstring(s):
    last, left, best = {}, 0, 0
    for r, ch in enumerate(s):
        if ch in last and last[ch] >= left:
            left = last[ch] + 1
        last[ch] = r
        best = max(best, r - left + 1)
    return best
```

Complexity: O(n)/O(min(n,Σ)).
Mindset: Update left with max(left, last[ch]+1); off-by-one bugs are common.

Meritshot
EDUCATION

3) Product of Array Except Self (prefix/suffix)
Approach: Prefix pass, then suffix accumulator—no division.

```python
def productExceptSelf(nums):
    n = len(nums)
    res = [1]*n
    for i in range(1, n): res[i] = res[i-1]*nums[i-1]
    suffix = 1
    for i in range(n-1, -1, -1):
        res[i] *= suffix
        suffix *= nums[i]
    return res
```

Complexity: O(n)/O(min(n,Σ)).
Mindset: Update left with max(left, last[ch]+1); off-by-one bugs are common.

Meritshot
EDUCATION

3) Product of Array Except Self (prefix/suffix)
Approach: Prefix pass, then suffix accumulator—no division.

```python
def productExceptSelf(nums):
    n = len(nums)
    res = [1]*n
    for i in range(1, n): res[i] = res[i-1]*nums[i-1]
    suffix = 1
    for i in range(n-1, -1, -1):
        res[i] *= suffix
        suffix *= nums[i]
    return res
```

Complexity: O(n)/O(1) extra.
Mindset: Division fails with zeros; confirm integer ranges.

Meritshot
EDUCATION

4) Merge Intervals
Approach: Sort by start; sweep and merge when cur.start ≤ last.end.

```python
def merge(intervals):
    intervals.sort()
    out = []
    for s, e in intervals:
        if not out or s > out[-1][1]: out.append([s, e])
        else: out[-1][1] = max(out[-1][1], e)
    return out
```

Complexity: O(n log n).
Mindset: Sorting key; empty/one interval edge cases.

Meritshot
EDUCATION

5) Minimum Window Substring (two maps)
Approach: Need/Have counters; expand right to satisfy; contract left to minimize.

```python
from collections import Counter
def minWindow(s
    need = Coun          = len(t)
    left = start = end = 0
    for right, ch in enumerate(s, 1):
        if need[ch] > 0: missing -= 1
        need[ch] -= 1
        while missing == 0:
            if end == 0 or right - left < end - start:
                start, end = left, right
            need[s[left]] += 1
            if need[s[left]] > 0: missing += 1
            left += 1
    return s[start:end]
```


Complexity: O(n).
Mindset: Indices are tricky; test with repeated letters.

Meritshot
EDUCATION

6) Kth Largest Element (heap or quickselect)
Approach: Min-heap of size k (simple) or quickselect (avg O(n)).

```python
import heapq
def findKthLargest(a, k):
    h = []
    for x in a:
        if len(h) < k: heapq.heappush(h, x)
        elif x > h[0]: heapq.heapreplace(h, x)
    return h[0]
```

Complexity: O(n log k).
Mindset: For huge n, prefer quickselect; be clear on “kth largest” vs “kth smallest”.

Meritshot
EDUCATION

7) Top K Frequent Elements

Approach: Count + bucket sort (O(n)) or heap (O(n log k)).

```python
from collections import Counter
def topKFrequent(nums, k):
    freq = Counter(nums)
    buckets = [[] for _ in range(len(nums)+1)]
    for v, c in freq.items():
        buckets[c].append(v)
    out = []
    for c in range(len(nums), 0, -1):
        for v in buckets[c]:
            out.append(v)
            if len(out) == k: return out
```

Mindset: Tie order doesn’t matter unless specified.

Meritshot
EDUCATION

8) Valid Parentheses (stack)
Approach: Push opens; match with map on close; empty at end.

```python
def isValid(s):
    m = {')':'(', ']':'[', '}':'{'}
    st = []
    for ch in s:
        if ch in m.values(): st.append(ch)
        elif not st or st.pop() != m[ch]: return False
    return not st
```

Complexity: O(n).
Mindset: Non-paren chars? Clarify; otherwise ignore or return False.

Meritshot
EDUCATION

9) Add Two Numbers (linked list)

Approach: Simulate addition with carry; create new list.

```python
class Node:
    def __init__(self, v=0, n=None): self.v, self.n = v, n

def addTwoNumbers(l1, l2):
    dummy = cur = Node()
    carry = 0
    while l1 or l2 or carry:
        s = (l1.v if l1 else 0) + (l2.v if l2 else 0) + carry
        carry, d = divmod(s, 10)
        cur.n = Node(d); cur = cur.n
        l1 = l1.n if l1 else None
        l2 = l2.n if l2 else None
    return dummy.n
```


Complexity: O(m+n).

Mindset: Forward vs reverse order—confirm with interviewer.

Meritshot
EDUCATION

10) Detect Cycle in Linked List (Floyd)

Approach: Fast/slow pointers; meet ⇒ cycle; then find entry.

```python
def detectCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow, fast = slow.next, fast.next.next
        if slow == fast:
            slow = head
            while slow != fast:
                slow, fast = slow.next, fast.next
            return slow
    return None
```

Complexity: O(n)/O(1).

Mindset: Explain proof briefly (distance algebra) if asked.

Meritshot
EDUCATION

11) Binary Tree Level Order (BFS)

Approach: Queue per level; append children.

```python
from collections import deque
def levelOrder(root):
    if not root: return []
    q, ans = deque([root]), []
    while q:
        level = []
        for _ in range(len(q)):
            node = q.popleft()
            level.append(node.val)
            if node.left: q.append(node.left)
            if node.right: q.append(node.right)
        ans.append(level)
    return ans
```

Complexity: O(n).

Mindset: Consider very skewed trees.

Meritshot
EDUCATION

12) Lowest Common Ancestor (BST)
Approach: Use BST ordering to walk down.

```python
def lca_bst(root, p, q):
    cur = root
    while cur:
        if p.val < cur.val and q.val < cur.val: cur = cur.left
        elif p.val > cur.val and q.val > cur.val: cur = cur.right
        else: return cur
```

Complexity: O(h).
Mindset: For general BT, need parent pointers or recursion + flags.

Meritshot
EDUCATION

13) Number of Islands (grid DFS/BFS)
Approach: Visit every ‘1’; flood-fill to mark visited.

```python
def lca_bst(root, p, q):
    cur = root
    while cur:
        if p.val < cur.val and q.val < cur.val: cur = cur.left
        elif p.val > cur.val and q.val > cur.val: cur = cur.right
        else: return cur
```

Complexity: O(RC).
Mindset: Clarify diagonal adjacency; avoid recursion depth limits (use stack/queue if needed).

Meritshot
EDUCATION

14) Course Schedule (topo sort / cycle detect)
Approach: Kahn’s algorithm with indegrees; or DFS colors.

```python
from collections import defaultdict, deque
def canFinish(n, prereq):
    g = defaultdict(list); indeg=[0]*n
    for a,b in prereq: g[b].append(a); indeg[a]+=1
    q = deque([i for i in range(n) if indeg[i]==0])
    visited = 0
    while q:
        u = q.popleft(); visited+=1
        for v in g[u]:
            indeg[v]-=1
            if indeg[v]==0: q.append(v)
    return visited==n
```

Complexity: O(n+m).
Mindset: Confirm edge direction.

Meritshot
EDUCATION

15) LRU Cache (design + hashmap + doubly list)

Approach: Hash map for O(1) lookup; doubly-linked list for recency order.
Key ops: get(k): move node to head; put(k,v): insert/move; evict tail when full.
Complexity: O(1) per op; O(capacity) space.
Mindset: Dummy head/tail nodes simplify pointers; handle updates vs inserts distinctly.

16) Design a URL Shortener

Requirements: Shorten and redirect; custom aliases; high read QPS; low latency; eventual analytics.

Approach:

- API: POST /shorten, GET /{code}.
- ID gen: Base62 of monotonically increasing IDs or hash(longURL)+collision-resolution; consider Snowflake IDs for scale.
- Storage: Hot path in Redis (code→URL), persistent in RDBMS/NoSQL.
- Scale: CDN for redirects, read-through cache, write-behind analytics queue (Kafka → ClickHouse).
- Consistency: Redirect path must be highly available; analytics can be eventual.
- Bottlenecks: Cache churn, hot keys; use TTL + LFU; rate-limit custom alias creation.
- Mindset: Mention unique code space, link expiration, abuse/fraud checks.

Meritshot
EDUCATION

17) Design a Rate Limiter (per user/IP)

Approach: Token bucket or sliding-window log in Redis.

- Token bucket: refill rate r/s, burst B; DECR in Lua for atomicity.
- Data model: rate:{user} → tokens, lastRefillTs.
- Global + endpoint-level limits; fallbacks for Redis failures.
- Tradeoffs: Token bucket smooths bursts; leaky bucket vs fixed window.
- Mindset: Idempotency, clock skew, multi-DC replication.

18) Design a News Feed (pull model)

Approach: Fan-out on write for normal users; fan-out on read for celebrities.

- Writes: Append to follower timelines (Redis sorted sets).
- Reads: Merge user timeline ZSET with pagination cursors.
- Storage: Posts in object store + metadata in Cassandra.
- Extras: Ranking features, denorm counters, backfill jobs.
- Mindset: Cold-start, dedupe, privacy, online/offline recompute.

19) Design an E-commerce Cart

Approach: Session or user-scoped carts in Redis (fast mutate) + async persist.

- Concurrency: CAS with version or hash-field updates; inventory check on checkout.
- Resilience: TTL + periodic snapshot to DB; idempotent checkout with order tokens.
- Mindset: Cross-device merge, abandoned-cart email, promo conflicts.

20) Design a Chat Service

Approach:

- Transport: WebSockets via gateway; rooms/shards by chatId.
- Storage: Messages in Kafka → Cassandra/Scylla; read replicas for history.
- Delivery: Per-user offsets/ack; retries; presence service.
- Extras: Typing, read receipts, and encryption keys management.
- Mindset: Ordering within room, back-pressure, rate-limits, mobile reconnect.

21) “Tell me about a time you took ownership of a failing project.”

Approach: Problem → actions you personally drove → quant impact.
Keep in mind: Tie to Ownership, Bias for Action.
Sample skeleton (STAR):

- S: Quarterly release slipping 3 weeks due to flaky tests.
- T: Reduce failures <1% and ship on time.
- A: Added failure taxonomy, quarantined top 10 flaky cases, parallelized CI, added retry with jitter.
- R: Flakiness 22%→0.7%, release on time 3 quarters straight.
- Avoid: “We” language—be specific about your decisions.

22) “Describe a time you disagreed and committed.”

LPs: Have Backbone; Disagree and Commit.
STAR: Proposed eventual-consistency store; you argued for strong consistency for the payments path; after the decision for eventual, you implemented idempotency keys + reconciliation job; chargebacks ↓ 35%.
Mindset: Show respectful challenge, then execution excellence.

23) “A time you dove deep into a production issue.”

LP: Dive Deep, Are Right, A Lot.
STAR: P99 latency spike; built flamegraphs, found N+1 in ORM; batched queries + added read-through cache; P99 1.4s→220ms.
Keep in mind: Metric names, dashboards, exact before/after numbers.

24) “Invented and simplified a process.”

LP: Invent and Simplify.
STAR: Release notes manual → built script to diff APIs + autogenerate change log; cut release time 6h→45m, errors ↓ 80%.
Mindset: Show customer impact (internal/external).

25) “Handled ambiguous requirements under tight deadlines.”

LP: Bias for Action, Deliver Results.
STAR: Unclear analytics spec; you created a minimal schema, defined events, shipped v1 in 4 days; iterated with A/B; uplift +6.2% CTR.
Mindset: Explicit risks/tradeoffs and how you de-risked.

Meritshot
EDUCATION

Your one-step destination for your Career Upskilling

Lets make a community of 20k+ learnerrs

meritshoteducation