Problem 1: Longest Substring Without Repeating Characters

- **Pattern:** Sliding Window / Hashing
- **LeetCode Link:** Longest Substring Without Repeating Characters – LeetCode #3

Hints:

- Use a hash map to store characters and their last index
- Maintain a sliding window [start, end]
- Move start when duplicate is found

Full C++ Solution:

```
#include <bits/stdc++.h>
using namespace std;

int lengthOfLongestSubstring(string s) {
   unordered_map<char,int> mp;
   int maxLen = 0, start = 0;
   for(int end = 0; end < s.size(); end++) {
      if(mp.find(s[end]) != mp.end())
         start = max(start, mp[s[end]] + 1);
      mp[s[end]] = end;
      maxLen = max(maxLen, end - start + 1);
   }
   return maxLen;
}

int main() {
   string s = "abcabcbb";
   cout << lengthOfLongestSubstring(s);
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(min(n, charset))
Edge Cases: Empty string, all unique characters, all same characters
Common Mistakes: Forgetting to move start correctly

Problem 2: Merge Two Sorted Linked Lists

- **Pattern:** Linked List / Merge Technique
- **LeetCode Link:** Merge Two Sorted Lists – LeetCode #21

Hints:

- Use dummy node to simplify pointer handling
- Compare nodes from both lists and append smaller one

Full C++ Solution:

```
#include <bits/stdc++.h>
using namespace std;

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x): val(x), next(NULL) {}
};

ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;
    while(l1 && l2) {
        if(l1->val < l2->val) { tail->next = l1; l1 = l1->next; }
        else { tail->next = l2; l2 = l2->next; }
        tail = tail->next;
    }
    tail->next = l1 ? l1 : l2;
    return dummy.next;
}

void printList(ListNode* head) {
    while(head) { cout << head->val << " "; head = head->next; }
}

int main() {
    ListNode* l1 = new ListNode(1);
    l1->next = new ListNode(3);
    l1->next->next = new ListNode(5);

    ListNode* l2 = new ListNode(2);
    l2->next = new ListNode(4);
    l2->next->next = new ListNode(6);

    ListNode* merged = mergeTwoLists(l1, l2);
    printList(merged);
    return 0;
}
```

Time Complexity: O(n + m)
Space Complexity: O(1)
Edge Cases: Empty list, one list longer than other
Common Mistakes: Not handling dummy node correctly

2️⃣ SQL Tasks

Query 1: Employees with salary above department average

```
SELECT e.Name, e.Salary, e.Department
FROM Employees e
WHERE e.Salary > (
    SELECT AVG(Salary)
    FROM Employees
    WHERE Department = e.Department
);
```

- **Goal:** Practice subquery and correlated queries
- **Optimization Tip:** Index on Department and Salary

Query 2: Count number of employees in each department

```
SELECT Department, COUNT(*) AS NumEmployees

FROM Employees

GROUP BY Department;
```

- **Goal:** Practice GROUP BY + aggregation

1️⃣ Operating Systems (OS)

Short Notes:

- **Deadlock:** Circular wait causing system halt
- **Synchronization:** Mutex, Semaphore
- **Threads:** Lightweight process, share memory
- **Context Switching:** Saving & loading CPU state

Q&A with Answers:

1. **What is a semaphore?**
    - A semaphore is a synchronization tool used to control access to shared resources.
    - **Types:**
        - Binary (0 or 1) – like a mutex
        - Counting (0 to n) – multiple accesses allowed
    - **Example:** Controlling access to a printer by multiple processes.
2. **Difference between mutex and semaphore?**

Feature	Mutex	Semaphore
Ownership	Only the thread that locks can unlock	Any thread can signal
Count	1	≥0
Use case	Mutual exclusion	Resource counting

3. **How to prevent deadlocks?**
    - Avoid circular wait (resource hierarchy)
    - Hold-and-wait prevention
    - Preemption: reclaim resources if necessary
4. **Explain race condition.**
    - Happens when multiple threads access shared data simultaneously

    - **Example:** Two threads incrementing the same counter → inconsistent final value

5. **What is a critical section?**
    - Code accessing shared resources that must not run concurrently
    - Protect using mutex or semaphore

2️⃣ DBMS

Short Notes:

- **Joins:** INNER, LEFT, RIGHT, FULL OUTER
- **Subqueries:** Correlated & non-correlated
- **Indexes:** Improve query performance
- **Normalization:** 1NF, 2NF, 3NF

Q&A with Answers:

1. **Difference between INNER and LEFT JOIN**
    - **INNER:** Returns only matching rows in both tables
    - **LEFT:** Returns all rows from left table, NULL for unmatched
2. **When to use subquery vs join?**
    - Subquery: Cleaner for single-row results or aggregation
    - Join: Better for combining large tables efficiently
3. **What is a composite index?**
    - Index on multiple columns
    - Improves query performance when filtering on multiple columns
4. **Difference between correlated and non-correlated subquery**
    - **Correlated:** Depends on outer query → executed for each outer row
    - **Non-correlated:** Independent → executed once
5. **Aggregate functions examples**
    - SUM(), AVG(), COUNT(), MAX(), MIN()

3️⃣ Computer Networks (CN)

Short Notes:

- **TCP Handshake:** 3-way handshake (SYN → SYN-ACK → ACK)
- **Routing:** Static vs Dynamic

- **IP Classes:** A, B, C, NAT

Q&A with Answers:

1. **Explain TCP 3-way handshake**
    - **Step 1:** Client sends SYN to server
    - **Step 2:** Server replies SYN-ACK
    - **Step 3:** Client sends ACK → connection established
2. **Difference between static and dynamic routing**

Feature	**Static Routing**	**Dynamic Routing**
Configuration	Manual	Automatic
Adaptability	Low	High
Use Case	Small networks	Large/complex networks

3. **What is NAT?**
    - Network Address Translation: Converts private IPs to public IPs
    - Allows multiple devices to share single public IP
4. **Difference between IPv4 and IPv6**
    - IPv4: 32-bit, ~4 billion addresses
    - IPv6: 128-bit, huge address space, better security
5. **TCP vs UDP**

Feature	**TCP**	**UDP**
Connection	Connection-oriented	Connectionless
Reliability	Reliable	Unreliable
Speed	Slower	Faster
Use Case	Web, Email	Video streaming, gaming

4️⃣ OOPs

Short Notes:

- **Polymorphism:** Compile-time (overloading), Runtime (virtual function)
- **Encapsulation:** Hiding data with getters/setters
- **Abstraction:** Hiding implementation details

- **Interface vs Abstract Class:** Interface = only declarations; Abstract = can have some implementations

Q&A with Answers:

1. **Explain runtime polymorphism**
   - Achieved via virtual functions
   - Example: Base class pointer calls derived class method
2. **Difference between abstract class and interface**

Feature	Abstract Class	Interface
Methods	Can have implemented + abstract	Only abstract (pure virtual)
Inheritance	Single or multiple	Multiple inheritance possible
Constructor	Allowed	Not allowed

3. **Compile-time vs runtime polymorphism**
   - Compile-time: Method overloading, operator overloading
   - Runtime: Virtual functions, inheritance
4. **Example of abstraction**
5. class Vehicle {
6. public:
7.   virtual void start() = 0; // Abstract method
8. };
9. class Car : public Vehicle {
10. public:
11.   void start() { cout << "Car started"; }
12. };
13. **Multiple inheritance in C++**
    - A class inherits from more than one base class
    - Watch for diamond problem → use virtual inheritance

5️⃣ SDLC / Software Engineering

Short Notes:

- **Agile:** Iterative, Sprints, Scrum roles (PO, Scrum Master, Dev)
- **Waterfall:** Sequential

- **Software Metrics:** Code coverage, maintainability

Q&A with Answers:

1. **Explain Scrum process**
   - Sprint planning → Daily standups → Sprint review → Retrospective
2. **Roles in Agile**
   - Product Owner (requirements)
   - Scrum Master (removes blockers)
   - Development Team (implements features)
3. **Advantages of Agile over Waterfall**
   - Faster feedback, flexible changes, customer involvement
4. **Sprint Planning**
   - Team commits to deliverables for 2–4 week sprint
5. **Software Metrics**
   - LOC, cyclomatic complexity, defect density

6️⃣ Software Testing

Short Notes:

- **Integration Testing:** Top-down, Bottom-up
- **System Testing:** End-to-end verification
- **Regression Testing:** Ensure no new bugs after changes

Q&A with Answers:

1. **Integration testing types**
   - Top-down: Test top modules first
   - Bottom-up: Test bottom modules first
2. **Difference between system and acceptance testing**

Feature	System Testing	Acceptance Testing
Purpose	Verify system meets requirements	Verify system meets user needs
Performed By	QA team	End-users / clients

3. **Test case components**
   - Test ID, Description, Steps, Expected result, Actual result, Status

4. **Regression testing purpose**
    - Detect if new code changes break existing functionality
5. **Black-box vs White-box examples**
    - Black-box: Functional testing, no code knowledge
    - White-box: Code coverage, branch testing

4️⃣ System Design / LLD

Problem: Design a **URL Shortener Service**

Key Points:

- DB Schema: Table URL with id, short_code, long_url, clicks
- API Endpoints: /shorten, /redirect/{short_code}, /analytics/{short_code}
- Bottlenecks: Hash collision, scaling DB
- Trade-offs: Custom hash vs random code, cache frequently used URLs

---

5️⃣ Aptitude

1. LCM of 12, 15, 20 → 60
2. Simple Interest: P=5000, R=5%, T=3 → SI = 500053/100 = 750
3. Probability: Drawing 2 aces → (4C2)/(52C2) = 6/1326 = 1/221

6️⃣ Behavioral (STAR)

Question: Tell me about a time you handled multiple tasks under pressure

Answer:

- **S:** During SAR image project, multiple preprocessing + model tuning tasks
- **T:** Deliver working model on deadline
- **A:** Prioritized tasks, automated preprocessing, used version control
- **R:** Completed project on time, accuracy improved by 13%, demo successful

---

📅 Evening Slot

7️⃣ Projects / Resume Task

- Update **Technical Skills section**: Add SQL joins, DSA patterns, system design concepts

---

8️⃣ DSA Bible Update

Problem	Pattern	LeetCode Link	Complexity	Notes
Longest Substring Without Repeating Characters	Sliding Window	Link	O(n)/O(n)	Edge: empty, all same, all unique
Merge Two Sorted Lists	Linked List Merge	Link	O(n+m)/O(1)	Edge: empty list, unequal length

---

Reflection

- Learned: Sliding window, Linked List merge, SQL subqueries, Agile basics
- Struggled: Some edge cases in substring problem
- Plan for Day 3: Harder DSA, DBMS advanced queries, CN routing algorithms, OOP design examples