-------DSA popular Question -----

Arrays and Strings:
Two Sum:

Given an array of integers, find two numbers such that they add up to a specific target.
Reverse String:

Reverse the characters in a string.
Longest Substring Without Repeating Characters:

Find the length of the longest substring without repeating characters.
Merge Intervals:

Given a collection of intervals, merge overlapping intervals.
Linked Lists:
Reverse Linked List:

Reverse a singly linked list.
Detect Cycle in a Linked List:

Determine if a linked list has a cycle.
Merge Two Sorted Lists:

Merge two sorted linked lists into a new sorted list.
Trees:
Binary Tree Traversal (Inorder, Preorder, Postorder):

Implement various tree traversal algorithms.
Validate Binary Search Tree (BST):

Check if a binary tree is a valid BST.
Lowest Common Ancestor (LCA) in a Binary Tree:

Find the lowest common ancestor of two nodes in a binary tree.
Sorting and Searching:
Binary Search:

Implement the binary search algorithm.
Merge Sort:

Implement the merge sort algorithm.
QuickSort:

Implement the quicksort algorithm.
Dynamic Programming:
Fibonacci Sequence:

Solve the Fibonacci sequence problem using dynamic programming.
Longest Increasing Subsequence:

Find the length of the longest increasing subsequence of a given array of integers.

0/1 Knapsack Problem:

Solve the 0/1 knapsack problem.
Graphs:
Breadth-First Search (BFS) and Depth-First Search (DFS):

Implement BFS and DFS for a graph.
Dijkstra's Algorithm:

Find the shortest path in a weighted graph using Dijkstra's algorithm.
Topological Sorting:

Perform topological sorting of a directed acyclic graph (DAG).
Miscellaneous:
LRU Cache Design:

Design and implement an LRU (Least Recently Used) cache.
String Matching Algorithms (e.g., KMP or Rabin-Karp):

Implement efficient string matching algorithms.

--------Arrays and Strings:-------
Container With Most Water:

Given n non-negative integers representing the height of each bar in a bar chart, find the area of the largest rectangle formed by the bars.
Product of Array Except Self:

Given an array nums, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].
First Missing Positive:

Given an unsorted integer array, find the smallest missing positive integer.
Linked Lists:
Add Two Numbers (Represented as Linked Lists):

Given two non-empty linked lists representing two non-negative integers, return their sum as a linked list.
Palindrome Linked List:

Check if a linked list is a palindrome.
Trees:
Serialize and Deserialize Binary Tree:

Serialize and deserialize a binary tree.
Maximum Depth of Binary Tree:

Find the maximum depth of a binary tree.
Binary Tree Level Order Traversal:

Given a binary tree, return the level order traversal of its nodes' values.

Sorting and Searching:
Find Peak Element:

Find the peak element in an array. A peak element is an element that is strictly greater than its neighbors.
Median of Two Sorted Arrays:

There are two sorted arrays nums1 and nums2 of size m and n, respectively. Find the median of the two sorted arrays.
Dynamic Programming:
Longest Palindromic Subsequence:

Find the length of the longest palindromic subsequence of a given string.
Minimum Path Sum:

Given a m x n grid filled with non-negative numbers, find a path from the top-left corner to the bottom-right corner that minimizes the sum of numbers along its path.
Graphs:
Word Ladder:

Given two words (beginWord and endWord), and a dictionary's word list, find the length of the shortest transformation sequence from beginWord to endWord.
Clone Graph:

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.
Miscellaneous:
Sliding Window Maximum:

Given an array nums, there is a sliding window of size k which moves from the leftmost side of the array to the rightmost, find the maximum value in each window.
Design a Tiny URL Service:

Design a system that shortens long URLs and expands short URLs.
Maximum Subarray Sum:

Find the contiguous subarray with the largest sum.

-------System design:-------

System design interviews in product-based companies typically assess a candidate's ability to architect scalable, efficient, and reliable systems. Here are some common system design questions asked in interviews:

1. Design a URL Shortening Service:

Design a system that shortens long URLs and expands short URLs efficiently. Discuss considerations such as scalability, fault tolerance, and analytics.
2. Design a Social Media Feed:
Design the backend system for a social media feed, considering factors like content ranking, user timelines, and real-time updates.
3. Design a TinyURL Service:
Similar to the URL shortening service, design a system for generating and resolving short URLs while handling issues like collision and scalability.
4. Design a Cache System:
Design a distributed cache system considering factors such as cache eviction policies, consistency, and scalability.
5. Design a File Storage System:
Design a scalable and distributed file storage system. Discuss components like metadata storage, data distribution, and replication.
6. Design a Ride-Sharing System:
Architect the backend system for a ride-sharing service. Consider aspects like user matching, real-time tracking, and surge pricing.
7. Design a Chat Application:
Design a scalable chat application supporting features like one-to-one messaging, group chats, and real-time updates.
8. Design a Distributed Messaging System:
Architect a distributed messaging system supporting pub-sub, guaranteed delivery, and fault tolerance.
9. Design a Content Delivery Network (CDN):
Design a CDN system to efficiently serve content globally, considering caching, load balancing, and content distribution.
10. Design a Recommendation System:
css
Copy code
- Design a recommendation system for a platform, considering user preferences, item catalog, and personalization.
Tips for System Design Interviews:
Clarify Requirements: Always start by clarifying the requirements and constraints of the system. Understand the expected scale, latency requirements, and any other relevant factors.

Scalability: Emphasize scalable and distributed designs. Consider how the system will handle increased load and data growth.

Trade-offs: Discuss trade-offs between consistency and availability, and other trade-offs relevant to the specific system.

Data Modeling: Pay attention to how data will be modeled and stored. Discuss database choices and considerations.

Failures and Recovery: Address how the system will handle failures, recovery, and redundancy.

API Design: If applicable, discuss the design of the system's APIs, considering usability and extensibility.

Security: Touch upon security considerations, including data encryption, access control, and protection against common security threats.

Communication: Clearly communicate your thought process, and be open to discussing alternative solutions. Consider collaboration with the interviewer in refining the design.

-------Multi threading :-------

1. Basic Concepts:
What is multithreading?
Explain the difference between processes and threads.
What are the advantages and disadvantages of multithreading?
2. Thread Creation and Management:
How can you create a thread in Java/Python/C++?
Compare Thread class and Runnable interface in Java.
What is the life cycle of a thread?
3. Synchronization:
What is synchronization in multithreading?
Explain the concept of a critical section.
How can you achieve synchronization in Java/Python/C++?
Discuss the differences between synchronized methods and blocks.
4. Locks and Semaphores:
What is a lock, and how does it work?
Explain the purpose of semaphores.
Compare ReentrantLock and synchronized blocks in Java.
5. Deadlocks:
What is a deadlock?
How can you prevent deadlocks?
Discuss strategies for handling deadlocks.
6. Thread Safety:
What is thread safety?
How can you make a class thread-safe in Java/Python/C++?
Discuss the concept of atomic operations.
7. Thread Pools:
What is a thread pool?
How can you create a thread pool in Java/Python/C++?
Discuss the advantages of using a thread pool.
8. Concurrency Utilities (Java):
Explain the purpose of Executor and ExecutorService.
What is the Future interface, and how is it used?
Discuss the java.util.concurrent package.
9. Race Conditions:
What is a race condition?
How can you identify and prevent race conditions?
Discuss techniques for avoiding race conditions in multithreading.
10. Parallel Programming:
diff
Copy code
- What is parallel programming?
- Discuss parallel programming models and frameworks.
- How does parallel programming differ from concurrent programming?
11. Memory Visibility:

diff
Copy code
- Explain the concept of memory visibility.
- What is the Java Memory Model (JMM)?
- Discuss techniques for ensuring memory visibility in multithreading.
12. Atomicity and Volatility:
markdown
Copy code
- Explain atomicity in the context of multithreading.
- Discuss the purpose of the `volatile` keyword.
- How can you use atomic classes in Java?
13. Fork-Join Framework (Java):
markdown
Copy code
- What is the Fork-Join framework?
- How does it relate to parallel programming?
- Discuss the `RecursiveTask` and `RecursiveAction` classes.
14. Thread Communication:
markdown
Copy code
- How can threads communicate with each other?
- Discuss inter-thread communication mechanisms.
- Explain the purpose of the `wait()`, `notify()`, and `notifyAll()` methods.
15. Thread Safety in Collections:
markdown
Copy code
- How can you make collections thread-safe in Java?
- Discuss thread-safe collection classes.
- Explain the concept of a `ConcurrentHashMap`.

-------Recurssion problems:-------

1. Factorial Calculation:
Write a recursive function to calculate the factorial of a given number.
2. Fibonacci Series:
Implement a recursive function to generate the nth Fibonacci number.
3. Sum of Digits:
Write a recursive function to find the sum of digits of a given number.
4. Power Function:
Implement a recursive function to calculate the power of a number.
5. Greatest Common Divisor (GCD):
Write a recursive function to find the GCD of two numbers using Euclid's algorithm.
6. Binary Search:
Implement binary search using recursion.
7. Merge Sort:
Implement the merge sort algorithm using recursion.
8. QuickSort:
Implement the quicksort algorithm using recursion.
9. Palindrome Check:

Write a recursive function to check if a given string is a palindrome.
10. Subset Generation:
sql
Copy code
- Given a set of distinct integers, write a recursive function to generate all possible subsets.
11. Permutations:
css
Copy code
- Implement a recursive algorithm to generate all permutations of a given string.
12. Combination Sum:
css
Copy code
- Given an array of integers and a target sum, write a recursive function to find all unique combinations that sum up to the target.
13. Towers of Hanoi:
arduino
Copy code
- Solve the Towers of Hanoi problem using recursion.
14. Binary Tree Traversal:
sql
Copy code
- Write recursive functions for inorder, preorder, and postorder traversals of a binary tree.
15. Path Sum in a Binary Tree:
css
Copy code
- Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.
16. Expression Parsing:
css
Copy code
- Write a recursive function to parse and evaluate a mathematical expression.
17. Generate Parentheses:
css
Copy code
- Given n pairs of parentheses, write a recursive function to generate all combinations of well-formed parentheses.
18. Subset Sum Problem:
css
Copy code
- Given a set of positive integers and a target sum, write a recursive function to determine if there exists a subset that sums up to the target.
19. Reverse Linked List:
arduino
Copy code
- Reverse a linked list using recursion.
20. Tree Diameter:
sql
Copy code

- Find the diameter of a binary tree using recursion.

-------Linked list :-------

1. Reverse a Linked List:
Implement a function to reverse a singly linked list.
2. Detect a Cycle in a Linked List:
Determine if a linked list has a cycle and find the starting node of the cycle.
3. Find the Middle of a Linked List:
Find the middle node of a singly linked list.
4. Merge Two Sorted Linked Lists:
Merge two sorted linked lists into a new sorted list.
5. Remove Nth Node From End of List:
Remove the nth node from the end of a singly linked list.
6. Detect Intersection Point in Y Shaped Linked List:
Given two linked lists that intersect, find the intersection point.
7. Remove Duplicates from a Sorted/Unsorted Linked List:
Remove duplicates from a sorted or unsorted linked list.
8. Add Two Numbers Represented by Linked Lists:
Given two linked lists representing two non-negative numbers, add them and return the sum as a linked list.
9. Palindrome Linked List:
Check if a linked list is a palindrome.
10. Flatten a Multilevel Doubly Linked List:
css
Copy code
- Flatten a multilevel doubly linked list.
11. Copy Random Pointer in a Linked List:
sql
Copy code
- Implement a deep copy of a linked list with each node having an additional random pointer.
12. Rotate a Linked List:
css
Copy code
- Given a linked list, rotate the list to the right by k places.
13. Insert in a Sorted Circular Linked List:
sql
Copy code
- Insert a new node into a sorted circular linked list.
14. Clone a Linked List with Next and Random Pointer:
sql
Copy code
- Clone a linked list with each node having pointers to both the next and random nodes.
15. Swap Nodes in Pairs:
kotlin
Copy code
- Given a linked list, swap every two adjacent nodes and return the modified list.
16. Intersection of Two Linked Lists:
bash
Copy code

- Find the node at which the intersection of two singly linked lists begins.
17. Reverse Nodes in k-Group:
csharp
Copy code
- Reverse nodes in k-group in a linked list.
18. Sort a Linked List:
css
Copy code
- Sort a linked list in O(n log n) time using constant space complexity.
19. Convert Binary Search Tree to Sorted Doubly Linked List:
css
Copy code
- Convert a Binary Search Tree (BST) to a sorted doubly linked list.
20. LRU Cache Implementation using Linked List:
arduino
Copy code
- Implement an LRU (Least Recently Used) cache using a linked list.

-------Binary Tree:-------

1. Binary Tree Traversals:
Implement recursive and iterative solutions for inorder, preorder, and postorder traversals of a binary tree.
2. Binary Search Tree (BST) Operations:
Implement operations for insertion, deletion, and searching in a Binary Search Tree (BST).
3. Find the Height/Depth of a Binary Tree:
Write a function to find the height or depth of a binary tree.
4. Check if a Binary Tree is Balanced:
Determine if a binary tree is balanced, i.e., the height of the left and right subtrees of any node differ by at most one.
5. Lowest Common Ancestor (LCA) in a Binary Tree:
Find the lowest common ancestor of two nodes in a binary tree.
6. Diameter of a Binary Tree:
Find the diameter (longest path between two leaf nodes) of a binary tree.
7. Convert Binary Tree to Doubly Linked List:
Convert a binary tree to a doubly linked list in-place.
8. Binary Tree Level Order Traversal:
Perform a level-order traversal of a binary tree and return the nodes at each level as separate lists.
9. Symmetric Tree:
Check if a binary tree is symmetric, i.e., the left and right subtrees are mirror images of each other.
10. Serialize and Deserialize Binary Tree:
css
Copy code
- Implement functions to serialize and deserialize a binary tree.
11. Maximum Depth of N-ary Tree:
mathematica
Copy code
- Find the maximum depth of an N-ary tree.
12. Path Sum in a Binary Tree:

css
Copy code
- Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.
13. Construct Binary Tree from Preorder and Inorder Traversal:
diff
Copy code
- Given preorder and inorder traversal sequences, construct the binary tree.
14. Validate Binary Search Tree (BST):
sql
Copy code
- Check if a binary tree is a valid Binary Search Tree.
15. Binary Tree Zigzag Level Order Traversal:
css
Copy code
- Perform a zigzag level order traversal of a binary tree.
16. Populating Next Right Pointers in Each Node:
sql
Copy code
- Connect each node in a binary tree to its right neighbor.
17. Count Complete Tree Nodes:
css
Copy code
- Count the number of nodes in a complete binary tree.
18. Binary Tree Cameras:
css
Copy code
- Place cameras in a binary tree to monitor all nodes.
19. Recover Binary Search Tree:
sql
Copy code
- Recover a Binary Search Tree where two nodes are swapped.
20. Binary Tree Longest Consecutive Sequence:
css
Copy code
- Find the length of the longest consecutive sequence path in a binary tree.

-------Arrays:-------

1. Find the Missing Number:
Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing.
2. Two Sum:
Given an array of integers, return indices of the two numbers such that they add up to a specific target.
3. Best Time to Buy and Sell Stock:
Given an array representing stock prices on different days, find the maximum profit that can be earned by buying and selling.
4. Rotate Array:

Rotate an array to the right by k steps.
5. Product of Array Except Self:
Given an array nums, return an array output such that output[i] is equal to the product of all elements except nums[i].
6. Maximum Subarray Sum:
Find the contiguous subarray with the largest sum.
7. Merge Intervals:
Given a collection of intervals, merge overlapping intervals.
8. Search in Rotated Sorted Array:
Search for a target value in a rotated sorted array.
9. Container With Most Water:
Given n non-negative integers representing the height of each bar in a bar chart, find the area of the largest rectangle formed by the bars.
10. Implement Stack Using Arrays:
css
Copy code
- Implement a stack using arrays and support push, pop, top, and retrieving the minimum element in constant time.
11. Jump Game:
sql
Copy code
- Determine if you can reach the last index of an array starting from the first index.
12. Majority Element:
bash
Copy code
- Find the majority element (element appearing more than n/2 times) in an array.
13. Kth Largest Element in an Array:
c
Copy code
- Find the kth largest element in an unsorted array.
14. Set Matrix Zeroes:
css
Copy code
- Given a matrix, set its rows and columns to 0 if an element is 0.
15. Move Zeroes:
scss
Copy code
- Given an array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
16. Pascal's Triangle:
sql
Copy code
- Generate the first n rows of Pascal's triangle.
17. Spiral Order Matrix:
css
Copy code
- Given a matrix, return the elements of the matrix in spiral order.
18. Maximum Product Subarray:
sql
Copy code
- Find the contiguous subarray within an array (containing at least one number) that has the largest product.

19. Minimum Size Subarray Sum:
css
Copy code
- Given an array of positive integers and a target sum, find the minimum length of a contiguous subarray whose sum is greater than or equal to the target.
20. Third Maximum Number:
c
Copy code
- Find the third maximum number in an array.

-------Bit Manipulation :-------

Count Set Bits:

Write a function to count the number of set bits (1s) in an integer.
Check Power of Two:

Given an integer, write a function to check if it is a power of two.
Single Number:

Given an array of integers where every element appears twice except for one, find that single element.
Bitwise AND of Range:

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.
Bit Swap Required:

Write a function to determine the number of bits needed to convert integer A to integer B.
Missing Number:

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing.
Reverse Bits:

Reverse the bits of a given 32-bit unsigned integer.
Binary Addition:

Write a function to add two binary numbers represented as strings.
Maximum XOR of Two Numbers in an Array:

Given an array of integers, find two numbers such that the XOR of those two numbers is the maximum possible.
Maximum Product of Word Lengths:

Given a list of words, find the maximum product of lengths of any two words where the two words do not share common letters.
Gray Code:

Generate the n-bit gray code sequence.

Swap Bits:

Given an integer, swap the odd and even bits.
Count Total Set Bits in All Numbers from 1 to N:

Count the total number of set bits in all numbers from 1 to N.
Bitwise ORs of Subarrays:

Given an array of integers, find the bitwise OR of all possible subarrays.
Power of Four:

Given an integer, write a function to check if it is a power of 4.

--------Stack Questions:--------

Balanced Parentheses:
Given a string of parentheses, check whether the parentheses are balanced.
Evaluate Postfix Expression:

Given a postfix expression, evaluate it using a stack.
Next Greater Element:
Given an array, find the next greater element for each element. The next greater element is the first element to the right that is greater than the current element.
Min Stack:

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Implement Two Stacks in an Array:

Implement two stacks in an array, efficiently utilizing the available space.
Valid Parentheses:

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
Largest Rectangle in Histogram:

Given an array representing the heights of bars in a histogram, find the largest rectangle in the histogram.
Design a Stack With Max Operation:

Design a stack that supports push, pop, top, and retrieving the maximum element in constant time.
Implement a Stack Using Queues:

Implement a stack using queues.
Simplify Directory Path:

Given an absolute path for a file (Unix-style), simplify it.

Decode String:

Given an encoded string, return its decoded string. The encoding rule is k[encoded_string], where k is a positive integer.
Next Greater Element II:

Given a circular array, find the next greater element for each element. The array wraps around, so the next greater element may be in the first part of the array.
Online Stock Span:

Write a class StockSpanner that collects daily stock prices and returns the span of the stock's price for the current day.
Expression Evaluation:

Given an arithmetic expression in Reverse Polish Notation (postfix), evaluate it.
Trapping Rain Water:

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

--------Queue Questions:---------

Implement a Circular Queue:

Implement a circular queue.
Design and Implement a Queue Using Stacks:

Implement a queue using stacks.
Generate Binary Numbers:

Generate the first N binary numbers.
Sliding Window Maximum:

Given an array and an integer k, find the maximum for each and every contiguous subarray of size k.
Rotting Oranges:

Given a grid representing oranges, determine the minimum time to rot all oranges.
Implement Dequeue (Double-ended Queue):

Implement a deque with insert and delete operations at both ends.
Generate Binary Numbers from 1 to N using Queue:

Generate binary numbers from 1 to N using a queue.
Design Hit Counter:

Design a hit counter that keeps track of a number of hits within a given time window.
Reconstruct Itinerary:

Given a list of airline tickets represented by pairs of departure and arrival airports, reconstruct the itinerary in order.
Moving Average from Data Stream:

Design a class that maintains the moving average of a stream of integers.

Rotating the Box:

Given an m x n matrix of characters, rotate it 90 degrees clockwise.
Implement LRU Cache:

Design and implement a data structure for Least Recently Used (LRU) cache.
The Maze II:

Given a maze with walls, open spaces, and starting/ending points, find the shortest path from start to end.
Moving Average from Data Stream:

Design a class that maintains the moving average of a stream of integers. Implement methods for adding a new integer and calculating the moving average.
Reveal Cards In Increasing Order:

In a deck of cards, every card has a unique integer. You can perform the following operations any number of times: Take the top card of the deck and move it to the bottom, or pick up the top N cards and reverse the order. Find the final order of the deck.

----------Graph Traversal:--------
Depth-First Search (DFS):

Implement depth-first search for a graph.
Breadth-First Search (BFS):

Implement breadth-first search for a graph.
Number of Islands:

Given a 2D grid representing land and water, count the number of islands.
Word Ladder:

Given two words (start and end) and a dictionary, find the length of the shortest transformation sequence from start to end.
Clone Graph:

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.
Graph Representation:
Graph Representation:

Implement a graph using an adjacency matrix and an adjacency list.
Detect Cycle in a Directed Graph:

Given a directed graph, check whether it contains a cycle.
Course Schedule:

There are a total of n courses to take, labeled from 0 to n-1. Some courses may have prerequisites. Determine if it is possible to finish all courses.
Shortest Paths:
Dijkstra's Algorithm:

Implement Dijkstra's algorithm to find the shortest path in a weighted graph.
Bellman-Ford Algorithm:

Implement the Bellman-Ford algorithm to find the shortest path in a weighted graph with negative edges.
Minimum Spanning Tree:
Kruskal's Algorithm:

Implement Kruskal's algorithm to find the minimum spanning tree of a connected, undirected graph.
Prim's Algorithm:

Implement Prim's algorithm to find the minimum spanning tree of a connected, undirected graph.
Advanced Topics:
Topological Sorting:

Given a directed graph, perform a topological sort.
Strongly Connected Components (SCC):

Find the strongly connected components in a directed graph.
A Algorithm:*

Implement the A* algorithm for pathfinding.
Graph Coloring:

Given an undirected graph, determine if it can be colored with at most k colors such that no two adjacent vertices share the same color.
Interview-Style Problems:
Friend Circles:

Given an N x N matrix representing friendships between students (1 for friends, 0 for non-friends), find the number of friend circles.
Word Search II:

Given a 2D board and a list of words, find all words on the board.
Cheapest Flights Within K Stops:

Given a list of flights, find the cheapest price from source to destination with up to k stops.
Alien Dictionary:

Given a list of words in an alien language, find the order of characters in the language.

--------Hash Table Basics:---------

Design a Hash Map:
Implement a basic hash map with operations like put, get, and remove.
Two Sum:

Given an array of integers, return indices of the two numbers such that they add up to a specific target.
Intersection of Two Arrays:

Given two arrays, write a function to compute their intersection.
Contains Duplicate:

Given an array of integers, find if the array contains any duplicates.
Group Anagrams:

Given an array of strings, group anagrams together.
Hashing Techniques:
Longest Substring Without Repeating Characters:

Given a string, find the length of the longest substring without repeating characters.
Minimum Window Substring:

Given a string S and a string T, find the minimum window in S that contains all the characters in T.
Four Sum II:

Given four lists A, B, C, and D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.
Advanced Hashing:
LRU Cache Implementation:

Design and implement an LRU (Least Recently Used) cache.
Longest Consecutive Sequence:

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
Subarray Sum Equals K:

Given an array of integers and an integer k, find the total number of continuous subarrays whose sum equals k.
Interview-Style Problems:
Valid Sudoku:

Determine if a 9x9 Sudoku board is valid.
Isomorphic Strings:

Given two strings s and t, determine if they are isomorphic.

Alien Dictionary:

Given a list of words in an alien language, find the order of characters in the language.
Find All Anagrams in a String:

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.
Repeated DNA Sequences:

Find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.
Collision Resolution:
Open Addressing: Linear Probing:

Implement a hash table with open addressing using linear probing.
Separate Chaining:

Implement a hash table with separate chaining for collision resolution.
Detect Cycle in a Directed Graph:

Given a directed graph, check whether it contains a cycle using hash set for cycle detection.
First Non-Repeating Character in a Stream:

Design a data structure that supports adding and finding the first non-repeating character in a stream of characters.