DS -1 PRACTICE

❖ Linked List

Basic Operations

1. **Insert at Beginning**: Write a function to insert a node at the beginning of a linked list.
2. **Insert at End**: Write a function to insert a node at the end of a linked list.
3. **Delete a Node**: Write a function to delete a node from a linked list given its value.
4. **Find a Node**: Write a function to find a node in a linked list given its value.
5. **Length of Linked List**: Write a function to count the number of nodes in a linked list.

Intermediate Problems

6. **Reverse a Linked List**: Write a function to reverse a linked list.
7. **Detect Loop**: Write a function to detect if there is a loop in a linked list.
8. **Remove Duplicates**: Write a function to remove duplicate nodes from a sorted linked list.
9. **Nth Node from End**: Write a function to find the Nth node from the end of a linked list.
10. **Middle of Linked List**: Write a function to find the middle node of a linked list.

Advanced Problems

11. **Merge Two Sorted Lists**: Write a function to merge two sorted linked lists into one sorted linked list.
12. **Intersection Point**: Write a function to find the intersection point of two linked lists.
13. **Detect and Remove Loop**: Write a function to detect and remove a loop in a linked list.
14. **Palindrome Linked List**: Write a function to check if a linked list is a palindrome.
15. **Flatten a Linked List**: Given a linked list where each node has two pointers, one to the next node and one to a child linked list, write a function to flatten the list.

Specialized Problems

16. **Clone a Linked List with Random Pointers**: Each node in the linked list has an additional random pointer which could point to any node in the list or null. Write a function to clone this linked list.
17. **Rotate Linked List**: Given a linked list, rotate the list to the right by k places.
18. **Partition Linked List**: Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.
19. **Add Two Numbers**: Given two numbers represented by linked lists, write a function that adds the two numbers and returns the sum as a linked list.
20. **Swap Nodes in Pairs**: Write a function to swap every two adjacent nodes in a linked list.
21.

❖ ARRAY

Basic Operations

1. **Insert an Element**: Write a function to insert an element at a specific position in an array.
2. **Delete an Element**: Write a function to delete an element from a specific position in an array.
3. **Search for an Element**: Write a function to find an element in an array and return its index.
4. **Reverse an Array**: Write a function to reverse an array.

Intermediate Problems

5. **Find Maximum and Minimum**: Write a function to find the maximum and minimum elements in an array.
6. **Rotate Array**: Write a function to rotate an array by `k` positions.
7. **Remove Duplicates**: Write a function to remove duplicates from a sorted array.
8. **Move Zeroes**: Write a function to move all zeroes in an array to the end while maintaining the relative order of the non-zero elements.
9. **Merge Two Sorted Arrays**: Write a function to merge two sorted arrays into one sorted array.
10. **Find Intersection**: Write a function to find the intersection of two arrays.

Advanced Problems

11. **Two Sum**: Write a function to find two numbers in an array that add up to a given target.
12. **Subarray Sum Equals K**: Write a function to find the total number of continuous subarrays whose sum equals to `k`.
13. **Find Missing Number**: Write a function to find the missing number in an array containing numbers from 1 to n.
14. **Find Duplicates**: Write a function to find all the duplicates in an array.
15. **Maximum Subarray (Kadane's Algorithm)**: Write a function to find the contiguous subarray within an array which has the largest sum.

Specialized Problems

16. **Trapping Rain Water**: Write a function to calculate how much water it can trap after raining, given `n` non-negative integers representing an elevation map.
17. **Product of Array Except Self**: Write a function to return an array such that each element at index `i` is the product of all the numbers in the original array except the one at `i`.
18. **Longest Consecutive Sequence**: Write a function to find the length of the longest consecutive elements sequence in an array.
19. **3Sum**: Write a function to find all unique triplets in the array which gives the sum of zero.

20. **Sliding Window Maximum**: Write a function to find the maximum in each sliding window of size `k` in the array.

Sorting and Searching

21. **Binary Search**: Write a function to perform binary search on a sorted array.
22. **Quick Sort**: Implement the Quick Sort algorithm.
23. **Merge Sort**: Implement the Merge Sort algorithm.
24. **Find Peak Element**: Write a function to find a peak element in an array.
25. **Find First and Last Position of Element in Sorted Array**: Write a function to find the first and last position of a given element in a sorted array.

Matrix Operations

26. **Search in a 2D Matrix**: Write a function to search for a value in an `m x n` matrix. This matrix has the following properties:
    - Integers in each row are sorted from left to right.
    - The first integer of each row is greater than the last integer of the previous row.
27. **Rotate Image**: Write a function to rotate an `n x n` 2D matrix representing an image by 90 degrees clockwise.
28. **Set Matrix Zeroes**: Write a function to set the entire row and column to zeroes if an element is zero.
29. **Spiral Matrix**: Write a function to return all elements of an `m x n` matrix in spiral order.
30. **Word Search**: Write a function to find if a word exists in a `2D` board of characters

❖ Binery Search

Basic Binary Search

1. **Basic Binary Search**: Implement binary search on a sorted array to find the index of a given target element.
2. **Binary Search with Recursion**: Implement binary search recursively on a sorted array.

Intermediate Problems

3. **Find First and Last Position**: Given a sorted array of integers, find the starting and ending position of a given target value. If the target is not found, return [-1, -1].
4. **Find Peak Element**: A peak element is an element that is strictly greater than its neighbors. Given an array of integers, find a peak element and return its index.

5. **Search in Rotated Sorted Array**: Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. Given such an array and a target value, determine if the target exists in the array.
6. **Find Minimum in Rotated Sorted Array**: Find the minimum element in a sorted array that has been rotated.

Advanced Problems

7. **Search Insert Position**: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.
8. **Find K Closest Elements**: Given a sorted array, two integers `k` and `x`, find the `k` closest elements to `x` in the array. The result should also be sorted in ascending order.
9. **Single Element in a Sorted Array**: You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Find this single element.
10. **Capacity to Ship Packages Within D Days**: A conveyor belt has packages that must be shipped from one port to another within `D` days. The `i-th` package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `D` days.

Specialized Problems

11. **Median of Two Sorted Arrays**: Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays.
12. **Allocate Minimum Number of Pages**: Given `n` books, each with some number of pages, and `k` students. Allocate books to students such that the maximum number of pages assigned to a student is minimized.
13. **Aggressive Cows**: Given `n` stalls and `k` cows, place the cows in the stalls such that the minimum distance between any two cows is maximized.
14. **Maximum Length of Repeated Subarray**: Given two integer arrays `nums1` and `nums2`, return the maximum length of a subarray that appears in both arrays.
15. **Kth Smallest Element in a Sorted Matrix**: Given an `n x n` matrix where each row and each column is sorted in ascending order, find the `k-th` smallest element in the matrix.

❖ Recursion

Basic Problems

1. **Factorial**: Write a function to compute the factorial of a number using recursion.
2. **Fibonacci Sequence**: Write a function to compute the nth Fibonacci number using recursion.
3. **Sum of Digits**: Write a function to compute the sum of digits of a number using recursion.
4. **Power of a Number**: Write a function to compute x^n (x raised to the power n) using recursion.
5. **Reverse a String**: Write a function to reverse a string using recursion.

Intermediate Problems

6. **Palindrome Check**: Write a function to check if a string is a palindrome using recursion.
7. **Array Sum**: Write a function to compute the sum of all elements in an array using recursion.
8. **Greatest Common Divisor (GCD)**: Write a function to compute the GCD of two numbers using recursion.
9. **Binary Search**: Write a recursive implementation of binary search.
10. **Permutations of a String**: Write a function to generate all permutations of a string using recursion.

Advanced Problems

11. **Tower of Hanoi**: Write a function to solve the Tower of Hanoi problem for n disks.
12. **N-Queens Problem**: Write a function to solve the N-Queens problem.
13. **Subset Sum**: Write a function to determine if there is a subset of the given set with a sum equal to a given number.
14. **Generate Parentheses**: Write a function to generate all combinations of well-formed parentheses for a given number of pairs.
15. **Word Search**: Given a 2D board and a word, write a function to check if the word exists in the board. The word can be constructed from letters of sequentially adjacent cells.

Dynamic Programming with Recursion (Memoization)

16. **Climbing Stairs**: Write a function to count the number of ways to climb a staircase with n steps, where you can take 1 or 2 steps at a time.
17. **Longest Common Subsequence**: Write a function to find the longest common subsequence of two strings using recursion with memoization.
18. **0/1 Knapsack Problem**: Write a recursive function to solve the 0/1 knapsack problem with memoization.
19. **Minimum Path Sum**: Write a function to find the minimum path sum from the top left to the bottom right of a 2D grid using recursion with memoization.
20. **Edit Distance**: Write a function to compute the edit distance between two strings using recursion with memoization.

Tree and Graph Problems

21. **Binary Tree Traversals**: Implement recursive functions for pre-order, in-order, and post-order traversals of a binary tree.
22. **Maximum Depth of Binary Tree**: Write a function to find the maximum depth of a binary tree using recursion.
23. **Path Sum**: Write a function to determine if the binary tree has a root-to-leaf path such that adding up all the values along the path equals a given sum.
24. **Combination Sum**: Write a function to find all unique combinations in an array where the candidate numbers sum to a given target.
25. **Sudoku Solver**: Write a function to solve a Sudoku puzzle using recursion and backtracking.