Data Structure

Section 1 – Array

*1. Write a program to find the sum of all elements in an array.
*2. Write a program to find the largest and smallest elements in an array.
*3. Write a program to remove duplicates from an array.
*4. Write a program to reverse an array in-place.
*5. Write a program to check if an array is sorted in ascending order.

#1. Write a program to find the second largest element in an array.
#2. Write a program to find the common elements between two arrays.
#3. Write a program to merge two sorted arrays into a single sorted array.
#4. Write a program to rotate an array by a given number of positions.
#5. Write a program to find the missing number in a given array of integers.

Section 2 – Two-Dimensional Array

*1. Write a program to find the sum of all elements in a two-dimensional array.
*2. Write a program to calculate the transpose of a matrix.
*3. Write a program to multiply two matrices.
*4. Write a program to find the largest element in each row of a matrix.
*5. Write a program to check if a given matrix is symmetric.

#1. Write a program to find the saddle point in a matrix.
#2. Write a program to sort the elements of a matrix in ascending order.
#3. Write a program to find the sum of the diagonals in a matrix.
#4. Write a program to find the number of rows and columns in a matrix.
#5. Write a program to check if two matrices are equal.

Section 3 – Stack

*1. Write a program to implement a stack using an array.
*2. Write a program to reverse a string using a stack.
*3. Write a program to check if a given expression has balanced parentheses using a stack.
*4. Write a program to convert an infix expression to postfix using a stack.
*5. Write a program to evaluate a postfix expression using a stack.

#1. Write a program to implement a stack using a linked list.
#2. Write a program to reverse a linked list using a stack.
#3. Write a program to convert a decimal number to binary using a stack.
#4. Write a program to check if a given string is a palindrome using a stack.
#5. Write a program to implement the Tower of Hanoi problem using a stack.

Section 4 – Queue

*1. Write a program to implement a queue using an array.
*2. Write a program to implement a circular queue using an array.

*3. Write a program to reverse the elements of a queue.
*4. Write a program to implement a queue using a linked list.
*5. Write a program to implement a priority queue using a heap.

#1. Write a program to simulate a printer queue using a queue.
#2. Write a program to implement a double-ended queue (deque) using a doubly linked list.
#3. Write a program to implement a circular buffer using a queue.
#4. Write a program to check if a given string is a palindrome using a queue.
#5. Write a program to generate the binary numbers from 1 to n using a queue.

Section 5 – LinkedList – Single, Double

*1. Write a program to create and display a singly linked list.
*2. Write a program to insert an element at the beginning of a singly linked list.
*3. Write a program to insert an element at the end of a singly linked list.
*4. Write a program to delete the first occurrence of an element in a singly linked list.
*5. Write a program to reverse a singly linked list.

#1. Write a program to find the middle element of a singly linked list.
#2. Write a program to create and display a doubly linked list.
#3. Write a program to insert an element at the beginning of a doubly linked list.
#4. Write a program to insert an element at the end of a doubly linked list.
#5. Write a program to delete the last occurrence of an element in a doubly linked list.

Section 6 – Recursion

*1. Write a program to calculate the factorial of a number using recursion.
*2. Write a program to find the nth Fibonacci number using recursion.
*3. Write a program to calculate the sum of digits of a number using recursion.
*4. Write a program to find the GCD of two numbers using recursion.
*5. Write a program to calculate the power of a number using recursion.

#1. Write a program to reverse a string using recursion.
#2. Write a program to generate all possible subsets of a set using recursion.
#3. Write a program to find the number of ways to climb n stairs using recursion.
#4. Write a program to solve the Tower of Hanoi problem using recursion.
#5. Write a program to check if a string is a palindrome using recursion.

Section 7 – Sequential, Binary

*1. Write a program to search for an element in an array using sequential search.
*2. Write a program to search for an element in a sorted array using binary search.
*3. Write a program to find the first and last occurrences of a number in an array using binary search.
*4. Write a program to search for an element in a two-dimensional array using binary search.
*5. Write a program to search for a substring in a string using sequential search.

#1. Write a program to count the occurrences of a word in a given text using sequential search.
#2. Write a program to search for a file in a directory using sequential search.
#3. Write a program to find the maximum and minimum elements in an array using sequential search.
#4. Write a program to search for a pattern in a DNA sequence using sequential search.
#5. Write a program to search for a given key in a binary search tree.

Section 8 – Sorting – Bubble, Merge, Heap, Selection

*1. Write a program to sort an array of integers using bubble sort.

*2. Write a program to sort an array of integers using merge sort.
*3. Write a program to sort an array of integers using heap sort.
*4. Write a program to sort an array of integers using selection sort.
*5. Write a program to sort an array of strings using bubble sort.

#1. Write a program to sort an array of strings using merge sort.
#2. Write a program to sort an array of strings using heap sort.
#3. Write a program to sort an array of strings using selection sort.
#4. Write a program to sort a two-dimensional array of integers using bubble sort.
#5. Write a program to sort a two-dimensional array of integers using merge sort.

Section 9 – Backtracking Algorithm

*1. Write a program to solve the N-Queens problem using backtracking.
*2. Write a program to generate all permutations of a string using backtracking.

#1. Write a program to solve the Sudoku puzzle using backtracking.
#2. Write a program to solve the Knapsack problem using backtracking.
#3. Write a program to generate all possible combinations of a set of numbers using backtracking.

Section 10 – Greedy Algorithm

*1. Write a program to solve the Fractional Knapsack problem using a greedy algorithm.
*2. Write a program to solve the Activity Selection problem using a greedy algorithm.

#1. Write a program to find the minimum number of coins needed to make a change using a greedy algorithm.
#2. Write a program to solve the Huffman coding problem using a greedy algorithm.
#3. Write a program to solve the Job Scheduling problem using a greedy algorithm.

Section 11 – Heap

*1. Write a program to build a max heap from an array of integers.
*2. Write a program to build a min heap from an array of integers.
*3. Write a program to insert an element into a max heap.
*4. Write a program to delete the root element from a max heap.
*5. Write a program to extract the minimum element from a min heap.

#1. Write a program to merge two max heaps into a single max heap.
#2. Write a program to check if a binary tree is a max heap.
#3. Write a program to find the kth largest element in an array using a min heap.
#4. Write a program to sort an array of integers using a heap sort algorithm.
#5. Write a program to implement a priority queue using a heap data structure.

Section 12 – Tree – Binary Tree, AVL Tree, Balanced

*1. Write a program to construct a binary tree from a given array representation and perform an inorder traversal to display its elements.
*2. Write a program to check if two binary trees are identical, i.e., they have the same structure and same values at corresponding positions.
*3. Write a program to find the height of a binary tree.
*4. Write a program to perform a rotation in an AVL tree, either left rotation or right rotation.
*5. Implement a program to delete a node from an AVL tree while maintaining balance.

*6. Write a program to check if a given binary tree is height-balanced or not.
*7. Write a program to find the diameter of a binary tree, which is the longest path between any two leaf nodes.
*8. Write a program to check if a binary tree is a perfect binary tree.

#1. Implement a program to find the maximum value in a binary tree iteratively, without using recursion.
#2. Implement a program to count the number of leaf nodes in a binary tree.
#3. Implement an AVL tree and write a program to perform an insertion operation, ensuring the tree remains balanced.
#4. Write a program to sort an array of integers using a heap sort algorithm.
#5. Write a program to find the minimum value in an AVL tree.
#6. Implement a program to convert an AVL tree into a balanced binary search tree.
#7. Implement a program to convert an unbalanced binary tree into a balanced binary tree.
#8. Implement a program to perform a right rotation on an unbalanced binary tree to make it balanced.

Section 13 – Graph – SPF, Dijkstra Algo

*1. Implement Dijkstra's algorithm to find the shortest path between two vertices in a weighted directed graph.
*2. Implement Kruskal's algorithm to find the minimum spanning tree (MST) of a weighted undirected graph.
*3. Implement the Floyd-Warshall algorithm to find the shortest paths between all pairs of vertices in a weighted directed graph.

#1. Write a program to find the minimum spanning tree (MST) of a weighted undirected graph using Prim's algorithm.
#2. Write a program to detect cycles in a directed graph using depth-first search (DFS).

Section 14 – Dynamic Programming

*1. Write a program to solve the Fibonacci sequence using dynamic programming.
*2. Write a program to solve the 0/1 knapsack problem using dynamic programming.
*3. Write a program to calculate the nth term of the Pascal's triangle using dynamic programming.

#1. Implement a program to find the longest common subsequence (LCS) of two given strings using dynamic programming.
#2. Implement the coin change problem using dynamic programming to find the minimum number of coins required to make a given sum.