Data Structure Programes

Arrays:

1.Implement a program to find the sum of elements in an array.
2.Write a program to find the maximum element in an array.
3.Implement an array reversal algorithm.
4.Write a program to find the intersection of two arrays.
5.Implement a program to rotate an array by k positions.
6.Write a program to remove duplicates from a sorted array.
7.Implement a program to merge two sorted arrays.

Linked Lists:

1.Create a linked list and perform basic operations (insertion, deletion, traversal).
2.Write a program to find the middle element of a linked list.
3.Implement a linked list reversal algorithm.
4.Write a program to detect a cycle in a linked list.
5.Implement a program to find the intersection point of two linked lists.
6.Write a program to remove duplicates from an unsorted linked list.
7.Implement a program to add two numbers represented by linked lists.

Stacks:

1.Implement a stack using an array.
2.Write a program to check for balanced parentheses using a stack.
3.Implement a program to convert infix expressions to postfix.
4.Write a program to evaluate a postfix expression.
5.Implement a stack-based algorithm to reverse a string.
6.Write a program to implement the Tower of Hanoi using stacks.

Queues:

1.Implement a queue using an array.
2.Write a program to reverse the first k elements of a queue.
3.Implement a circular queue.
4.Write a program to generate binary numbers from 1 to n using a queue.
5.Implement a program to design a data structure that supports push, pop, top, and retrieving the
6.minimum element in constant time.
7.Write a program to implement a double-ended queue (Deque).

Trees:

1.Implement a program to create and traverse a binary tree.
2.Write a program to find the height of a binary tree.
3.Implement an algorithm to check if a binary tree is balanced.

4.Write a program to perform a level-order traversal of a binary tree.
5.Implement a program to check if two trees are identical.
6.Write a program to find the lowest common ancestor in a binary tree.
7.Implement an algorithm to perform an in-order traversal without recursion.

Graphs:

1.Create a graph and perform basic operations (add vertex, add edge, remove vertex, remove edge).
2.Write a program to perform depth-first search (DFS) on a graph.
3.Implement a program to perform breadth-first search (BFS) on a graph.
4.Write a program to find the shortest path in a weighted graph using Dijkstra's algorithm.
5.Implement a program to find the connected components in an undirected graph.
6.Write a program to detect a cycle in a directed graph.
7.Implement an algorithm to check if a graph is bipartite.

Sorting Algorithms:

1.Implement the bubble sort algorithm.
2.Write a program to perform selection sort.
3.Implement the insertion sort algorithm.
4.Write a program to perform merge sort.
5.Implement the quicksort algorithm.
6.Write a program to perform heap sort.
7.Implement the counting sort algorithm.
8.Write a program to perform radix sort.

Searching Algorithms:

1.Implement a linear search algorithm.
2.Write a program to perform binary search on a sorted array.
3.Implement an algorithm to find the first and last occurrences of an element in a sorted array.
4.Write a program to search an element in a rotated sorted array.
5.Implement an interpolation search algorithm.
6.Write a program to perform depth-first search in a binary search tree.
7.Implement a breadth-first search in a binary search tree.

Hashing:

1.Implement a hash table with basic operations (insert, delete, search).
2.Write a program to find the first non-repeating character in a string using hashing.
3.Implement a program to detect a cycle in an undirected graph using hashing.
4.Write a program to find the intersection of two arrays using hashing.

Dynamic Programming:

1.Implement a program to find the nth Fibonacci number using dynamic programming.
2.Write a program to find the length of the longest increasing subsequence in an array.
3.Implement an algorithm to solve the coin change problem.
4.Write a program to find the longest common subsequence of two strings.
5.Implement an algorithm to find the edit distance between two strings.

Advanced Data Structures:

1.Implement a trie data structure.
2.Write a program to implement a suffix array.
3.Implement a program to construct a suffix tree.
4.Write a program to implement a segment tree for range queries.
5.Implement an algorithm to perform range updates in a segment tree.

Miscellaneous:

1.Write a program to reverse a sentence without reversing the words.
2.Implement a program to check if a string is a palindrome.
3.Write a program to find the majority element in an array.
4.Implement an algorithm to find the kth smallest/largest element in an array.
5.Write a program to implement a circular linked list and perform basic operations.

Application-based:

1.Implement a program to evaluate a postfix expression using a stack.
2.Write a program to implement a priority queue using a heap.
3.Implement a program to simulate a basic file system using a tree structure.
4.Write a program to find the shortest path in a maze using BFS.
5.Implement an algorithm to detect a cycle in an undirected graph using DFS.

Graph Algorithms:

1.Write a program to find the strongly connected components in a directed graph.
2.Implement an algorithm to find the articulation points in an undirected graph.
3.Write a program to perform topological sorting on a directed acyclic graph.
4.Implement an algorithm to find the maximum flow in a network using the Ford-Fulkerson method.

Trie Applications:

1.Write a program to implement autocomplete using a trie.
2.Implement a spell-checker using a trie.
3.Write a program to find all words in a dictionary that are anagrams of a given input word.

Huffman Coding:

1.Implement a program to perform Huffman coding for text compression.
2.Write a program to decode a Huffman-encoded message.

AVL Trees:

1.Implement an AVL tree and perform basic operations (insertion, deletion, search).
2.Write a program to convert a sorted array into a balanced BST.

B-Trees:

1.Implement a B-tree and perform basic operations (insertion, deletion, search).
2.Write a program to implement a B+ tree for efficient search and retrieval.

Spatial Data Structures:

1.Implement a Quadtree for spatial indexing.
2.Write a program to implement a 2D range search using a Quadtree.

Network Flow:

1.Implement an algorithm to find the maximum bipartite matching in a graph.
2.Write a program to find the minimum cut in a flow network using the Ford-Fulkerson method.

Geometric Algorithms:

1.Implement an algorithm to find the convex hull of a set of points.
2.Write a program to determine whether two rectangles overlap.

Interval Trees:

1.Implement an interval tree and perform basic operations (insertion, deletion, search).
2.Write a program to find all overlapping intervals in a set.
3.These exercises cover a wide range of data structure concepts and can be adapted to various
4.programming languages. Feel free to choose the ones that align with your language of choice and programming environment.