Data Structure Assignments (Linked list)

Inserting a Node:

1. At the beginning of the list.
2. At the end of the list.
3. After a given node in the list.
4. Before a given node in the list.

Deleting a Node:

1. Delete the first node of the list.
2. Delete the last node of the list.
3. Delete a node with a specific value.
4. Delete a node at a specific position.
5. Traversing the List: Traverse and print all elements in the list.

Searching a Node:

1. Search for a node by its value.
2. Search for a node by its position.
3. Reversing the List: Reverse the entire linked list.
4. Finding the Length of the List:
5. Count the number of nodes in the list.
6. Detecting a Loop- Check if the linked list contains a loop.
7. Finding the Middle Element: Find the middle element of the list.
8. Finding the Nth Node from the End:
9. Find the Nth node from the end of the list.
10. Merging Two Linked Lists- Merge two sorted linked lists.

Removing Duplicates:

1. Remove duplicate nodes from a sorted linked list.
2. Remove duplicate nodes from an unsorted linked list.
3. Splitting the List: Split the linked list into two halves.
4. Sorting the List: Sort the linked list using various algorithms (e.g., merge sort, quick sort).
5. Swapping Nodes: Swap two nodes in the list without swapping the data.
6. Deleting the Entire List: Delete all nodes and free up memory.
7. Intersection of Two Linked Lists: Find the intersection point of two linked lists.
8. Detecting and Removing Loop: Detect if there is a loop in the list and remove it.
9. Palindrome Check: Check if the linked list is a palindrome.
10. Clone a Linked List with Random Pointers:
11. Clone a linked list where each node has an additional random pointer.
12. Union and intersection of list 1 – as 67 98 90 101 45 34. List -2 98 56 48 101 78
13. Merging two sorted linked lists.
14. Finding the nth node from the end.
15. Checking for palindrome linked lists.
16. Flattening a linked list with child pointers.

Hacker Rank (All programs to be completed and screenshot to be attached)

https://www.hackerrank.com/challenges/arrays-ds/problem?isFullScreen=true

https://www.hackerrank.com/challenges/dynamic-array/problem?isFullScreen=true

https://www.hackerrank.com/challenges/insert-a-node-at-the-tail-of-a-linked-list/problem?isFullScreen=true

https://www.hackerrank.com/challenges/detect-whether-a-linked-list-contains-a-cycle/problem?isFullScreen=true

Library Management System

Data Structures: Linked List, Stack, Queue

Description: Create a system to manage books in a library. Implement features like adding/removing books, issuing/returning books using stacks and queues.

2. Contact Management System

Data Structures: Hash Table, Trie

Description: Develop a contact management system where users can add, search, and delete contacts. Use a hash table for quick lookups and a trie for prefix-based searches.

3. Movie Ticket Booking System

Data Structures: Array, Queue, Linked List

Description: Implement a system to book movie tickets where users can select seats from an array. Use a queue for handling booking requests and a linked list to manage booking history.

4. Text Editor with Undo/Redo Functionality

Data Structures: Stack, Doubly Linked List

Description: Build a simple text editor that supports basic editing and undo/redo functionality using stacks to store editing actions.

5. Student Record Management System

Data Structures: Binary Search Tree (BST)

Description: Implement a system to manage student records, where you can add, delete, and search for student information. Use a BST to organize records.

6. Mini Social Media Platform

Data Structures: Graph, Hash Table

Description: Create a simple social media platform where users can follow/unfollow each other. Use a graph to represent the relationships and a hash table to manage user information.

7. Word Frequency Counter

Data Structures: Hash Map, Heap

Description: Write a program to count the frequency of words in a text file. Use a hash map to store word frequencies and a heap to keep track of the most frequent words.

8. Maze Solver

Data Structures: Stack, Queue, Graph

Description: Implement a maze solver using depth-first search (DFS) or breadth-first search (BFS). Represent the maze as a graph and use a stack or queue to find the path.

9. Hospital Management System

Data Structures: Queue, Priority Queue

Description: Build a system to manage patient records and prioritize patient care based on the severity of their condition using a priority queue.

10. File Directory Structure Simulation

Data Structures: Tree

Description: Simulate a file directory structure where users can create, delete, and move files/folders. Use a tree to represent the hierarchical structure.

11. Simple Compiler

Data Structures: Stack, Hash Map

Description: Implement a basic compiler that can evaluate simple arithmetic expressions and check for balanced parentheses using a stack.

12. Banking System

Data Structures: Linked List, Queue, Hash Map

Description: Develop a banking system where users can create accounts, deposit/withdraw money, and transfer funds. Use linked lists to manage transactions and hash maps for account details.

13. Chessboard Problem

Data Structures: Array, Linked List

Description: Implement a solution for the N-Queens problem using backtracking. Represent the chessboard as a 2D array and track queen positions using a linked list.

14. Parking Lot Management System

Data Structures: Stack, Queue

Description: Create a system to manage parking lots where cars can enter and leave. Use a stack to simulate the parking lot and a queue for managing the entry and exit of cars.

15. Railway Reservation System

Data Structures: Queue, Linked List, Hash Table

Description: Implement a railway reservation system where users can book and cancel tickets. Use a queue for managing booking requests and a hash table for storing passenger information.

16. Dictionary with Spell Checker