1.	Write a Python program to manage the borrowing records of books in a library. Implement the following functionalities: • Compute the average number of books borrowed by all library members. • Find the book with the highest and lowest number of borrowings in the library. • Count the number of members who have not borrowed any books (denoted by a borrow count of 0). • Display the most frequently borrowed
2.	In an e-commerce system, customer account IDs are stored in a list, and you are tasked with writing a program that implements the following: **Linear Search: Check if a particular customer account ID exists in the list.**
3.	In an e-commerce system, customer account IDs are stored in a list, and you are tasked with writing a program that implements the following: **Binary Search: Implement Binary search to find if a customer account ID exists, improving the search efficiency over the basic linear**
4.	In a company, employee salaries are stored in a list as floating-point numbers. Write a Python program that sorts the employee salaries in ascending order using the following two algorithms: **Selection Sort: Sort the salaries using the selection sort algorithm.** After sorting the salaries, the program should display top five highest salaries in the company
5.	In a company, employee salaries are stored in a list as floating-point numbers. Write a Python program that sorts the employee salaries in ascending order using the following two algorithms: **Bubble Sort: Sort the salaries using the bubble sort algorithm**. After sorting the salaries, the program should display top five highest salaries in the company
6.	Implementing a real-time undo/redo system for a text editing application using a Stack data structure. The system should support the following operations: • Make a Change: A new change to the document is made. • Undo Action: Revert the most recent change and store it for potential redo. • Redo Action: Reapply the most recently undone action. • Display Document State: Show the current state of the document after undoing or redoing an action
7.	Implement a real-time event processing system using a Queue data structure. The System should support the following features: • Add an Event: When a new event occurs, it should be added to the event queue. • Process the Next Event: The system should process and remove the event that has been in the queue the longest. • Display Pending Events: Show all the events currently waiting to be processed. • Cancel an Event: An event can be canceled if it has not been processed.
8.	Implement a real-time event processing system using a **Circular Queue data structure using array.** The System should support the following features: • Add an Event: When a new event occurs, it should be added to the event queue. • Process the Next Event: The system should process and remove the event that has been in the queue the longest. • Display Pending Events: Show all the events currently waiting to be processed. • Cancel an Event: An event can be canceled if it has not been processed.

9.	Create a Student Record Management System using linked list • Use a singly/doubly linked list to store student data (Roll No, Name, Marks). • **Perform operations: Add, Display, Delete, Update.** • **Display records in ascending/descend**ing order based on marks or roll number.
10.	Create a Student Record Management System using linked list • Use a singly/doubly linked list to store student data (Roll No, Name, Marks). • **Perform operations: Add, Display, Search, and Sort.** • Display records in ascending/descending order based on marks or roll number.
11.	Implement a hash table of size 10 and use the division method as a hash function. In case of a collision, use chaining. Implement the following operations: • Insert(key): Insert key-value pairs into the hash table. • Search(key): Search for the value associated with a given key. • Display(): Displays the hash table.
12.	Implement a hash table of size 10 and use the division method as a hash function. In case of a collision, use chaining. Implement the following operations: • Insert(key): Insert key-value pairs into the hash table. • Delete(key): Delete a key-value pair from the hash table. • Display(): Displays the hash table.
13.	Design and implement a hash table of fixed size. Use the division method for the hash function and resolve collisions using linear probing. Allow the user to perform the following operations: • Insert a key • Search for a key • Display the table
14.	Design and implement a hash table of fixed size. Use the division method for the hash function and resolve collisions using linear probing. Allow the user to perform the following operations: • Insert a key • Delete a key • Display the table
15.	A pizza shop receives multiple orders from several locations. Assume that one pizza boy is tasked with delivering pizzas in nearby locations, which is represented using a graph. The time required to reach from one location to another represents node connections. Solve the problem of delivering a pizza to all customers in the minimum time. Use appropriate data structures.
16.	Construct an expression tree from the given prefix expression, e.g., +--a*bc/def, traverse it using post-order traversal (non-recursive), and then delete the entire tree.
17.	In a computer system, multiple functions call one another during program execution. To manage these nested function calls efficiently, a **Stack** data structure is used to store the state of each function call. When a function is called, its details are **pushed** onto the stack, and when the function returns, its details are **popped**. Implement this mechanism using a **Linked List**, which allows dynamic allocation of memory and avoids stack overflow issues seen in static arrays. Demonstrate operations such as **Push**, **Pop**, and **Display**, simulating how function call management is performed in real-time systems like compilers or interpreters.

18.	Consider a real-time scenario where customers arrive at a **ticket booking counter** one after another. The customers must be served in the same order they arrive — the **First In First Out (FIFO)** manner. Design a **Queue** using a **Linked List** to handle this system dynamically. Each node in the linked list represents a customer waiting in line. Implement operations such as **Enqueue** (customer joins the queue), **Dequeue** (customer is served and leaves), and **Display** (show waiting customers). This practical simulates real-time service queue management systems used in banks, hospitals, or ticketing applications.
19.	There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. **Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency matrix representation** of the graph. Check whether the graph is connected or not. Justify the storage representation used.
20.	Implement a real-time event processing system using a **Circular Queue data structure using linked list.** The System should support the following features: • Add an Event: When a new event occurs, it should be added to the event queue. • Process the Next Event: The system should process and remove the event that has been in the queue the longest. • Display Pending Events: Show all the events currently waiting to be processed. • Cancel an Event: An event can be canceled if it has not been processed.