34. What is Linked List? State different types of Linked Lists.

Linked List:

A *Linked List* is a dynamic data structure in which elements (called **nodes**) are connected using **pointers**.
Each node has two parts:

1. **Data field** → stores the actual data.
2. **Pointer field** → stores the address of the next (or previous) node.

Unlike arrays, linked lists do not require contiguous memory, and size can grow or shrink at runtime.

Types of Linked Lists:

1. **Singly Linked List** – Each node has data and a pointer to the next node.
2. **Doubly Linked List** – Each node has data, a pointer to the next node, and a pointer to the previous node.
3. **Circular Linked List** – The last node is connected back to the first node, forming a circle.
    - Can be singly circular or doubly circular.

35. Explain advantages of Linked List over Array.

Advantages of Linked List over Array:

1. **Dynamic Size** – Linked list can grow or shrink at runtime, while array size is fixed.
2. **Efficient Insertion/Deletion** – Adding or removing elements is easier in linked lists (just change pointers) compared to shifting elements in arrays.
3. **Better Memory Utilization** – No need for contiguous memory allocation, unlike arrays.

4. **Implementation of Complex Structures** – Useful for implementing advanced data structures like stacks, queues, graphs, and hash tables.
5. **No Wastage of Memory** – Memory is allocated as needed for each node, while arrays may waste memory if not fully used.

36. Consider an information management system that maintains data of students (fields - Roll No.

and Name). Apply suitable concepts of linked lists and write an algorithm to insert a data

record at the end of this list.

```
Step 1: Create a new node NEW.

Step 2: Set NEW.RollNo = given RollNo, NEW.Name = given Name.

Step 3: Set NEW.Next = NULL.

Step 4: IF HEAD = NULL (list is empty) THEN

            Set HEAD = NEW

        ELSE

            Set TEMP = HEAD

            WHILE TEMP.Next ≠ NULL

                TEMP = TEMP.Next

            END WHILE

            Set TEMP.Next = NEW

        END IF

Step 5: EXIT
```

37. Write an algorithm to implement insertion, deletion, traversal in Singly Linked List.

(a) Insertion at End

```
Step 1: Create a new node NEW.
Step 2: Set NEW.DATA = value, NEW.NEXT = NULL.
Step 3: IF HEAD = NULL THEN
            HEAD = NEW
        ELSE
            TEMP = HEAD
            WHILE TEMP.NEXT ≠ NULL
                TEMP = TEMP.NEXT
            END WHILE
            TEMP.NEXT = NEW
        END IF
Step 4: EXIT
```

(b) Deletion (of KEY)

```
Step 1: IF HEAD = NULL THEN PRINT "List empty", EXIT
Step 2: IF HEAD.DATA = KEY THEN
            HEAD = HEAD.NEXT, FREE old HEAD, EXIT
Step 3: TEMP = HEAD
        WHILE TEMP.NEXT ≠ NULL AND TEMP.NEXT.DATA ≠ KEY
            TEMP = TEMP.NEXT
        END WHILE
Step 4: IF TEMP.NEXT = NULL THEN PRINT "Not found"
        ELSE
            DEL = TEMP.NEXT
        TEMP.NEXT = TEMP.NEXT.NEXT
        FREE DEL
Step 5: EXIT
```

(c) Traversal

```
Step 1: IF HEAD = NULL THEN PRINT "List empty", EXIT
Step 2: TEMP = HEAD
Step 3: WHILE TEMP ≠ NULL
        PRINT TEMP.DATA
        TEMP = TEMP.NEXT
    END WHILE
Step 4: EXIT
```

38. Write an algorithm to implement insertion, deletion, traversal in Doubly Linked List.

(a) Insertion at End

```
Step 1: Create a new node NEW, NEW.DATA = value, NEW.NEXT =
NULL.
Step 2: IF HEAD = NULL THEN
        NEW.PREV = NULL, HEAD = NEW
    ELSE
        TEMP = HEAD
        WHILE TEMP.NEXT ≠ NULL
            TEMP = TEMP.NEXT
        END WHILE
        TEMP.NEXT = NEW
        NEW.PREV = TEMP
    END IF
Step 3: EXIT
```

(b) Deletion (of KEY)

```
Step 1: IF HEAD = NULL THEN PRINT "List empty", EXIT
Step 2: TEMP = HEAD
    WHILE TEMP ≠ NULL AND TEMP.DATA ≠ KEY
        TEMP = TEMP.NEXT
    END WHILE
Step 3: IF TEMP = NULL THEN PRINT "Not found", EXIT
Step 4: IF TEMP.PREV ≠ NULL THEN TEMP.PREV.NEXT = TEMP.NEXT
    ELSE HEAD = TEMP.NEXT
Step 5: IF TEMP.NEXT ≠ NULL THEN TEMP.NEXT.PREV = TEMP.PREV
Step 6: FREE TEMP
Step 7: EXIT
```

(c) Traversal (Forward)

```
Step 1: TEMP = HEAD
Step 2: WHILE TEMP ≠ NULL
        PRINT TEMP.DATA
        TEMP = TEMP.NEXT
    END WHILE
Step 3: EXIT
```

39. Explain the data structure that is capable to efficiently utilize holes in memory while loading data.

The data structure that efficiently utilizes *holes in memory* while loading data is the Linked List.

Explanation:

- **In arrays, memory must be allocated in a contiguous block, which may not always be available due to fragmentation (holes in memory).**
- **A linked list, however, does not require contiguous memory. Each node can be stored in any available memory location (hole), and nodes are connected using pointers.**
- **This allows efficient utilization of scattered free memory (holes).**

Example:
If memory has small free blocks at different locations, we can store each node of a linked list in those blocks and link them together, thus avoiding wastage.

---

40. Sketch the process of insertion at middle in a Singly Linked List.

Algorithm: InsertAtMiddle(DATA, POS)

```
Step 1: Create NEW node, NEW.DATA = DATA
Step 2: TEMP = HEAD
Step 3: Repeat (POS-1) times → TEMP = TEMP.NEXT
Step 4: NEW.NEXT = TEMP.NEXT
Step 5: TEMP.NEXT = NEW
Step 6: EXIT
```

---

41. Write an algorithm to implement insertion, deletion, traversal in Circular Linked List.

(a) Insertion at End

```
Step 1: Create NEW, NEW.DATA = value
Step 2: IF HEAD = NULL THEN
            HEAD = NEW, NEW.NEXT = HEAD
        ELSE
            TEMP = HEAD
            WHILE TEMP.NEXT ≠ HEAD
                TEMP = TEMP.NEXT
            END WHILE
            TEMP.NEXT = NEW
            NEW.NEXT = HEAD
        END IF
Step 3: EXIT
```

(b) Deletion (KEY)

```
Step 1: IF HEAD = NULL THEN PRINT "List empty", EXIT
Step 2: TEMP = HEAD, PREV = NULL
Step 3: WHILE TEMP.DATA ≠ KEY
            IF TEMP.NEXT = HEAD THEN PRINT "Not found", EXIT
            PREV = TEMP, TEMP = TEMP.NEXT
        END WHILE
Step 4: IF TEMP = HEAD AND TEMP.NEXT = HEAD THEN HEAD = NULL
        ELSE IF TEMP = HEAD THEN
            LAST = HEAD
            WHILE LAST.NEXT ≠ HEAD
            LAST = LAST.NEXT
        END WHILE
        HEAD = HEAD.NEXT
        LAST.NEXT = HEAD
    ELSE PREV.NEXT = TEMP.NEXT
Step 5: FREE TEMP
Step 6: EXIT
```

(c) Traversal

```
Step 1: IF HEAD = NULL THEN PRINT "List empty", EXIT
Step 2: TEMP = HEAD
Step 3: REPEAT
        PRINT TEMP.DATA
        TEMP = TEMP.NEXT
    UNTIL TEMP = HEAD
Step 4: EXIT
```

---

42. Consider the real-life application of Personal Computers, where multiple applications are

running. All the running applications are kept in the memory and the OS gives a fixed time

slot to all for running. Apply suitable concepts of linked lists and write an algorithm to delete

an application that the user closes. Applications can be represented as with application IDs.

Algorithm: DeleteApplication(AppID)

```
Step 1: IF HEAD = NULL THEN PRINT "No applications", EXIT
Step 2: TEMP = HEAD, PREV = NULL
Step 3: WHILE TEMP.AppID ≠ AppID
      IF TEMP.NEXT = HEAD THEN PRINT "Not found", EXIT
      PREV = TEMP, TEMP = TEMP.NEXT
    END WHILE
Step 4: IF TEMP = HEAD AND TEMP.NEXT = HEAD THEN HEAD = NULL
    ELSE IF TEMP = HEAD THEN
      LAST = HEAD
      WHILE LAST.NEXT ≠ HEAD
         LAST = LAST.NEXT
      END WHILE
      HEAD = HEAD.NEXT
      LAST.NEXT = HEAD
    ELSE PREV.NEXT = TEMP.NEXT
Step 5: FREE TEMP
Step 6: PRINT "Application closed"
Step 7: EXIT
```