VIDUSH SOMANY INSTITUTE OF TECHNOLOGY AND RESEARCH, KADI

KADI SARVA VISHWAVIDYALAYA, GANDHINAGAR

CC303-N

DATA STRUCTURES & ALGORITHMS

LAB MANUAL

SEMESTER – 3

ENROLLMENT NO	
NAME	
BRANCH	

CERTIFICATE

This is to certify that Mr. / Ms. ______________________________ Of class ______________________ Enrollment No.: __________________________ has Satisfactorily completed the course in _____________________________ at Vidush Somany Institute Of Technology & Research,Kadi (Kadi Sarva Vishwavidyalaya) for ________ Year (B.E.) semester ______ of Computer Science & Engineering / Computer Engineering / Information Technology in the Academic year ______________.

Date of Submission: ______/______/______

Faculty Name with Signature **Head of Department**

INDEX

	4. Delete 5. Display 6. Exit	
08	Write a program to implement Searching Algorithms 1. Sequential search 2. Binary search	
09	Write a program to implement following sorting algorithms 1.Selection sort 2.Bubble sort 3.Merge sort 4.Quick sort	
10	Write a program to implement breadth first search (BFS) graph traversal algorithm.	
11	Write a program to implement depth first search (DFS) graph traversal algorithm.	

PRACTICAL 1

Write a menu driven program to perform the following operations on the STACK using an array.
1. Push
2. Pop
3. Peep
4. Change
5. Display the contents
6. Exit

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 5   // Maximum size of stack

int stack[SIZE];
int top = -1;    // Initially stack is empty

// Function to push an element
void push(int x) {
    if (top == SIZE - 1) {
        printf("Stack Overflow! Cannot push %d\n", x);
    } else {
        stack[++top] = x;
        printf("%d pushed to stack\n", x);
    }
}

// Function to pop an element
void pop() {
    if (top == -1) {
        printf("Stack Underflow! Nothing to pop\n");
    } else {
        printf("%d popped from stack\n", stack[top--]);
    }
}

// Function to peep (view) element at given position from top
void peep(int pos) {
    int index = top - pos + 1;
    if (index < 0) {
        printf("Invalid position! Stack has fewer elements.\n");
    } else {
        printf("Element at position %d from top is %d\n", pos, stack[index]);
    }
}

// Function to change element at given position from top
void change(int pos, int val) {
    int index = top - pos + 1;
    if (index < 0) {
        printf("Invalid position! Cannot change.\n");
    } else {
        stack[index] = val;
        printf("Element at position %d changed to %d\n", pos, val);
    }
}

// Function to display stack contents
void display() {
    if (top == -1) {
        printf("Stack is empty!\n");
    } else {
        printf("Stack contents: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

// Main function
int main() {
    int choice, val, pos;

    while (1) {
        printf("\n--- STACK MENU ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peep\n");
        printf("4. Change\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter value to push: ");
            scanf("%d", &val);
            push(val);
            break;

        case 2:
            pop();
            break;

        case 3:
            printf("Enter position from top to peep: ");
            scanf("%d", &pos);
            peep(pos);
            break;

        case 4:
            printf("Enter position from top to change: ");
            scanf("%d", &pos);
            printf("Enter new value: ");
            scanf("%d", &val);
            change(pos, val);
            break;

        case 5:
            display();
            break;

        case 6:
            printf("Exiting program...\n");
            exit(0);

        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Output:

MCQ Questions

1. In the given stack program, what does the variable top represent?

A) The maximum size of the stack
B) The index of the last inserted element
C) The total number of elements in the stack
D) The first element of the stack
Answer: ………………………………………………………………………………………

2. What happens if we try to push an element when the stack is full?

A) Stack Underflow occurs
B) Stack Overflow occurs
C) The element is inserted at index 0
D) The program automatically resizes the stack
Answer: ………………………………………………………………………………………

3. In the peep() function, what does pos = 1 represent?

A) Bottom element of the stack
B) Middle element of the stack
C) Top element of the stack
D) Invalid position
Answer: ………………………………………………………………………………………

4. Which function is used to modify the value at a given position from the top?

A) push()
B) pop()
C) change()
D) peep()
Answer: ………………………………………………………………………………………

5. If the stack is empty (top == -1), what will the display() function print?

A) "Stack Overflow!"
B) "Stack Underflow!"
C) "Stack is empty!"
D) Nothing will be printed
Answer: ………………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 2

Write a program to convert an infix expression into reverse polish (postfix) notation with parenthesis.

```
#include <stdio.h>
#include <ctype.h>   // for isalnum()
#include <string.h>  // for strlen()

#define SIZE 100

char stack[SIZE];
int top = -1;

// Function to push onto stack
void push(char c) {
   if (top == SIZE - 1) {
      printf("Stack Overflow!\n");
   } else {
      stack[++top] = c;
   }
}

// Function to pop from stack
char pop() {
   if (top == -1) {
      return -1;  // stack empty
   } else {
      return stack[top--];
   }
}

// Function to return precedence of operators
int precedence(char c) {
   if (c == '^')
      return 3;
   else if (c == '*' || c == '/')
      return 2;
   else if (c == '+' || c == '-')
      return 1;
   else
      return -1;
}

// Function to convert infix to postfix
void infixToPostfix(char infix[]) {
   char postfix[SIZE];
```

```
    int i, k = 0;
    char ch;

    for (i = 0; i < strlen(infix); i++) {
        ch = infix[i];

        // If operand (a-z or A-Z or 0-9), add to postfix
        if (isalnum(ch)) {
            postfix[k++] = ch;
        }
        // If '(', push onto stack
        else if (ch == '(') {
            push(ch);
        }
        // If ')', pop until '('
        else if (ch == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[k++] = pop();
            }
            pop(); // remove '('
        }
        // If operator
        else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }

    // Pop remaining operators
    while (top != -1) {
        postfix[k++] = pop();
    }

    postfix[k] = '\0';  // null terminate string
    printf("Postfix Expression: %s\n", postfix);
}

// Main function
int main() {
    char infix[SIZE];

    printf("Enter an infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix);

    return 0;
}
```

Output:

MCQ Questions

1. What is the main data structure used in infix to postfix conversion?
A) Queue
B) Stack
C) Linked List
D) Array only
Answer: ………………………………………………………………………………………

2. In the program, what does the function precedence(char c) return for operator *?
A) 1
B) 2
C) 3
D) -1
Answer: ………………………………………………………………………………………

3. What happens in the algorithm when a closing parenthesis) is encountered?
A) Push it onto the stack
B) Pop all operators until an opening (is found
C) Ignore it
D) Exit the program
Answer: ………………………………………………………………………………………

4. If the input is (A+B)*(C-D), what will be the correct postfix expression?
A) AB+CD-*
B) AB+CD-
C) ABCD+-*
D) A+B*C-D
Answer: ………………………………………………………………………………………

5. Which library function is used in the program to check if a character is an operand (alphabet or digit)?
A) isalpha()
B) isdigit()
C) isalnum()
D) strlen()
Answer: ………………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 3

Write a program to solve the problem of Tower of Hanoi (Application of stack).

```
#include <stdio.h>
// Recursive function to solve Tower of Hanoi
void towerOfHanoi(int n, char source, char auxiliary, char destination) {
   // Base case: if only 1 disk
   if (n == 1) {
      printf("Move disk 1 from %c to %c\n", source, destination);
      return;
   }

   // Step 1: Move top (n-1) disks from source to auxiliary
   towerOfHanoi(n - 1, source, destination, auxiliary);

   // Step 2: Move the remaining disk to destination
   printf("Move disk %d from %c to %c\n", n, source, destination);

   // Step 3: Move (n-1) disks from auxiliary to destination
   towerOfHanoi(n - 1, auxiliary, source, destination);
}

// Main function
int main() {
   int n;

   printf("Enter number of disks: ");
   scanf("%d", &n);

   printf("\nSolution for Tower of Hanoi with %d disks:\n", n);
   towerOfHanoi(n, 'A', 'B', 'C');  // A=Source, B=Auxiliary, C=Destination

   return 0;
}
```

Output:

MCQ Questions

1. How many moves are required to solve the Tower of Hanoi problem with n disks?
A) n
B) n²
C) 2^n – 1
D) n!
Answer: ………………………………………………………………………………………

2. In the Tower of Hanoi problem, what is the role of the auxiliary rod?
A) It stores all disks permanently
B) It is used as a temporary storage to move disks
C) It holds only the largest disk
D) It is not necessary for the solution
Answer: ………………………………………………………………………………………

3. What is the base case in the recursive solution of Tower of Hanoi?
A) When there are 0 disks
B) When there is 1 disk
C) When all disks are on the destination rod
D) When the largest disk is moved
Answer: ………………………………………………………………………………………

4. If there are 3 disks in Tower of Hanoi, what is the minimum number of moves required?
A) 3
B) 5
C) 7
D) 9
Answer: ………………………………………………………………………………………

5. Which of the following concepts is mainly used in solving Tower of Hanoi?
A) Iteration
B) Recursion and Stack
C) Sorting
D) Searching
Answer: ………………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 4

Write a menu driven program to perform the following operations on the QUEUE using an array.
1. Insert
2. Delete
3. Search
4. Change
5. Display the contents
6. Exit

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 5   // Maximum size of Queue

int queue[SIZE];
int front = -1, rear = -1;

// Function to insert element
void insert(int x) {
    if (rear == SIZE - 1) {
        printf("Queue Overflow! Cannot insert %d\n", x);
    } else {
        if (front == -1) front = 0; // first element
        queue[++rear] = x;
        printf("%d inserted into queue\n", x);
    }
}

// Function to delete element
void delete() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow! Nothing to delete\n");
    } else {
        printf("%d deleted from queue\n", queue[front++]);
        if (front > rear) {  // reset queue
            front = rear = -1;
        }
    }
}

// Function to search element
void search(int val) {
    if (front == -1) {
        printf("Queue is empty!\n");
        return;
    }
    int found = 0;
    for (int i = front; i <= rear; i++) {
        if (queue[i] == val) {
            printf("%d found at position %d\n", val, i - front + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("%d not found in the queue\n", val);
    }
}

// Function to change element at given position
void change(int pos, int val) {
    if (front == -1) {
        printf("Queue is empty!\n");
        return;
    }
    if (pos <= 0 || pos > (rear - front + 1)) {
        printf("Invalid position!\n");
    } else {
        queue[front + pos - 1] = val;
        printf("Element at position %d changed to %d\n", pos, val);
    }
}

// Function to display queue contents
void display() {
    if (front == -1) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue contents: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

// Main function
int main() {
    int choice, val, pos;

    while (1) {
```

```
printf("\n--- QUEUE MENU ---\n");
printf("1. Insert\n");
printf("2. Delete\n");
printf("3. Search\n");
printf("4. Change\n");
printf("5. Display\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
case 1:
    printf("Enter value to insert: ");
    scanf("%d", &val);
    insert(val);
    break;

case 2:
    delete();
    break;

case 3:
    printf("Enter value to search: ");
    scanf("%d", &val);
    search(val);
    break;

case 4:
    printf("Enter position to change: ");
    scanf("%d", &pos);
    printf("Enter new value: ");
    scanf("%d", &val);
    change(pos, val);
    break;

case 5:
    display();
    break;

case 6:
    printf("Exiting program...\n");
    exit(0);

default:
    printf("Invalid choice! Try again.\n");
}
```

```
    }
    return 0;
}
```

Output:

MCQ Questions

1. In a linear queue implemented using an array, what happens when rear == SIZE - 1?

A) Queue is empty
B) Queue Overflow occurs
C) Queue Underflow occurs
D) Rear is reset to 0
Answer: ………………………………………………………………………………………

2. What condition indicates that the queue is empty in the given program?

A) rear == SIZE - 1
B) front == rear
C) front == -1
D) rear == 0
Answer: ………………………………………………………………………………………

3. In the delete() function, when the last element is removed, why do we reset front = rear = -1?

A) To free memory
B) To indicate queue is empty again
C) To avoid infinite loop
D) To double the size of the queue
Answer: ………………………………………………………………………………………

4. If the queue currently has elements [10, 20, 30] and we call delete(), what will be displayed?

A) 10 deleted from queue
B) 30 deleted from queue
C) Queue is empty!
D) 20 deleted from queue
Answer: ………………………………………………………………………………………

5. In the given program, what does the change(pos, val) function do?

A) Deletes the element at the given position
B) Inserts the value at the given position
C) Replaces the value at the given position with a new value
D) Searches for the given value
Answer: ………………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 5

Write a menu driven program to perform the following operations on the CIRCULARQUEUE using an array.
1. Insert
2. Delete
3. Search
4. Change
5. Display the contents
6. Exit

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 5   // Maximum size of Circular Queue

int cq[SIZE];
int front = -1, rear = -1;

// Function to check if queue is full
int isFull() {
    return ((front == 0 && rear == SIZE - 1) || (front == rear + 1));
}

// Function to check if queue is empty
int isEmpty() {
    return (front == -1);
}

// Function to insert element
void insert(int x) {
    if (isFull()) {
        printf("Circular Queue Overflow! Cannot insert %d\n", x);
        return;
    }
    if (front == -1) { // first element
        front = rear = 0;
    } else if (rear == SIZE - 1 && front != 0) {
        rear = 0; // wrap around
    } else {
        rear++;
    }
    cq[rear] = x;
    printf("%d inserted into Circular Queue\n", x);
}

// Function to delete element
```

```
void delete() {
    if (isEmpty()) {
        printf("Circular Queue Underflow! Nothing to delete\n");
        return;
    }
    printf("%d deleted from Circular Queue\n", cq[front]);
    if (front == rear) { // only one element
        front = rear = -1;
    } else if (front == SIZE - 1) {
        front = 0; // wrap around
    } else {
        front++;
    }
}

// Function to search element
void search(int val) {
    if (isEmpty()) {
        printf("Circular Queue is empty!\n");
        return;
    }
    int i = front;
    int pos = 1;
    while (1) {
        if (cq[i] == val) {
            printf("%d found at position %d\n", val, pos);
            return;
        }
        if (i == rear) break;
        i = (i + 1) % SIZE;
        pos++;
    }
    printf("%d not found in Circular Queue\n", val);
}

// Function to change element at given position
void change(int pos, int val) {
    if (isEmpty()) {
        printf("Circular Queue is empty!\n");
        return;
    }
    int count = 1;
    int i = front;
    while (1) {
        if (count == pos) {
            cq[i] = val;
            printf("Element at position %d changed to %d\n", pos, val);
            return;
        }
        if (i == rear) break;
        i = (i + 1) % SIZE;
        count++;
    }
    printf("Invalid position!\n");
}

// Function to display queue contents
void display() {
    if (isEmpty()) {
        printf("Circular Queue is empty!\n");
        return;
    }
    printf("Circular Queue contents: ");
    int i = front;
    while (1) {
        printf("%d ", cq[i]);
        if (i == rear) break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

// Main function
int main() {
    int choice, val, pos;

    while (1) {
        printf("\n--- CIRCULAR QUEUE MENU ---\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Search\n");
        printf("4. Change\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &val);
            insert(val);
            break;

        case 2:
            delete();
            break;

        case 3:
            printf("Enter value to search: ");
            scanf("%d", &val);
            search(val);
            break;

        case 4:
            printf("Enter position to change: ");
            scanf("%d", &pos);
            printf("Enter new value: ");
            scanf("%d", &val);
            change(pos, val);
            break;

        case 5:
            display();
            break;

        case 6:
            printf("Exiting program...\n");
            exit(0);

        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Output:

MCQ Questions

1. In a circular queue, which condition indicates that the queue is full?

A) front == rear

B) (front == 0 && rear == SIZE-1) || (front == rear + 1)

C) rear == SIZE - 1

D) front == -1

Answer: ……………………………………………………………………………………

2. What happens to rear when it reaches the end of the array in a circular queue?

A) It overflows

B) It is reset to 0 (wrap-around)

C) It becomes equal to front

D) It stays at last position forever

Answer: ……………………………………………………………………………………

3. In the given program, what does the function isEmpty() check?

A) front == 0

B) rear == SIZE - 1

C) front == -1

D) rear == -1

Answer: ……………………………………………………………………………………

4. If a circular queue of size 5 currently has elements [20, 30, 40] (front=0, rear=2) and we call delete(), what will be displayed?

A) 20 deleted from Circular Queue

B) 40 deleted from Circular Queue

C) Queue is empty!

D) 30 deleted from Circular Queue

Answer: ……………………………………………………………………………………

5. What is the main advantage of a circular queue over a linear queue?

A) Uses less memory

B) Can handle multiple queues at once

C) Reuses freed-up space efficiently

D) Allows random access to elements

Answer: ……………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 6

Write a menu driven program to perform the following operations on a Singly Linked list.

1. Insert	**6. Search**
2. Insend	**7. Sort**
3. Insat	**8. Count**
4. Delete	**9. Display**
5. Reverse	**10. Exit**

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;

// Function to create a new node
struct Node* createNode(int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    return newNode;
}

// 1. Insert at beginning
void insertBeg(int val) {
    struct Node* newNode = createNode(val);
    newNode->next = head;
    head = newNode;
    printf("%d inserted at beginning\n", val);
}

// 2. Insert at end
void insertEnd(int val) {
    struct Node* newNode = createNode(val);
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("%d inserted at end\n", val);
}

// 3. Insert at given position
void insertAt(int pos, int val) {
    if (pos <= 0) {
        printf("Invalid position!\n");
        return;
    }
    struct Node* newNode = createNode(val);
    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        printf("%d inserted at position %d\n", val, pos);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position out of range!\n");
        free(newNode);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
        printf("%d inserted at position %d\n", val, pos);
    }
}

// 4. Delete element
void deleteNode(int val) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev = NULL;

    if (temp != NULL && temp->data == val) {
        head = temp->next;
        free(temp);
        printf("%d deleted from list\n", val);
        return;
    }
    while (temp != NULL && temp->data != val) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("%d not found in list!\n", val);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("%d deleted from list\n", val);
}

// 5. Reverse list
void reverse() {
    struct Node* prev = NULL, *curr = head, *next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
    printf("List reversed\n");
}

// 6. Search element
void search(int val) {
    struct Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == val) {
            printf("%d found at position %d\n", val, pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf("%d not found in list\n", val);
}

// 7. Sort list
void sort() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* i, *j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("List sorted\n");
}

// 8. Count nodes
void count() {
    int c = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        c++;
        temp = temp->next;
    }
    printf("Total nodes: %d\n", c);
}

// 9. Display list
void display() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    printf("List contents: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main menu
int main() {
    int choice, val, pos;
    while (1) {
        printf("\n--- SINGLY LINKED LIST MENU ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete\n");
        printf("5. Reverse\n");
        printf("6. Search\n");
        printf("7. Sort\n");
        printf("8. Count\n");
        printf("9. Display\n");
        printf("10. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter value: ");
            scanf("%d", &val);
            insertBeg(val);
            break;
        case 2:
            printf("Enter value: ");
            scanf("%d", &val);
            insertEnd(val);
            break;
        case 3:
            printf("Enter position: ");
            scanf("%d", &pos);
            printf("Enter value: ");
            scanf("%d", &val);
            insertAt(pos, val);
            break;
        case 4:
            printf("Enter value to delete: ");
            scanf("%d", &val);
            deleteNode(val);
            break;
        case 5:
            reverse();
            break;
        case 6:
            printf("Enter value to search: ");
            scanf("%d", &val);
            search(val);
            break;
        case 7:
            sort();
```

```
            break;
        case 8:
            count();
            break;
        case 9:
            display();
            break;
        case 10:
            printf("Exiting program...\n");
            exit(0);
        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Output:

MCQ Questions

1. In a singly linked list, each node contains:
A) Only data
B) Data and address of next node
C) Data and address of previous node
D) Only address of next node
Answer: ……………………………………………………………………………………

2. Which operation requires updating the head pointer in a singly linked list?
A) Insert at end
B) Delete at end
C) Insert at beginning
D) Traverse the list
Answer: ……………………………………………………………………………………

3. What will the reverse() function do in the given program?
A) Sort the list in descending order
B) Print the list in reverse order
C) Rearrange the nodes so that the list order is reversed
D) Delete all elements from the list
Answer: ……………………………………………………………………………………

4. If the linked list contains nodes 10 → 20 → 30 → NULL and we call deleteNode(20), what will be the resulting list?
A) 10 → 30 → NULL
B) 20 → 30 → NULL
C) 10 → 20 → NULL
D) 30 → NULL
Answer: ……………………………………………………………………………………

5. Which sorting method is implemented in the sort() function of the program?
A) Merge Sort
B) Quick Sort
C) Bubble Sort (by data swapping)
D) Insertion Sort
Answer: ……………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 7

Write a menu driven program to perform the following operations on a Doubly Linked list.

1. Insert
2. Insend
3. Insat
4. Delete
5. Display
6. Exit

```
#include <stdio.h>

#include <stdlib.h>


// Doubly linked list node structure

struct Node {

   int data;

   struct Node* prev;

   struct Node* next;

};


struct Node* head = NULL;


// Function to create a new node

struct Node* createNode(int val) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   newNode->data = val;

   newNode->prev = NULL;

   newNode->next = NULL;

   return newNode;

}


// 1. Insert at beginning

void insertBeg(int val) {

   struct Node* newNode = createNode(val);
```

```
    if (head == NULL) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
    printf("%d inserted at beginning\n", val);
}

// 2. Insert at end
void insertEnd(int val) {
    struct Node* newNode = createNode(val);
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
    printf("%d inserted at end\n", val);
}

// 3. Insert at position
void insertAt(int pos, int val) {
    if (pos <= 0) {
        printf("Invalid position!\n");
        return;
    }
    struct Node* newNode = createNode(val);

    if (pos == 1) {  // Insert at head
        insertBeg(val);
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Position out of range!\n");
        free(newNode);
    } else {
        newNode->next = temp->next;
        newNode->prev = temp;
        if (temp->next != NULL) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
        printf("%d inserted at position %d\n", val, pos);
    }
}

// 4. Delete a node
void deleteNode(int val) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;

    while (temp != NULL && temp->data != val) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("%d not found in list!\n", val);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        head = temp->next; // deleting head
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("%d deleted from list\n", val);
}

// 5. Display list
```

```
void display() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    printf("List contents: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main menu
int main() {
    int choice, val, pos;
    while (1) {
        printf("\n--- DOUBLY LINKED LIST MENU ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
```

```
    printf("Enter value: ");
    scanf("%d", &val);
    insertBeg(val);
    break;
case 2:
    printf("Enter value: ");
    scanf("%d", &val);
    insertEnd(val);
    break;
case 3:
    printf("Enter position: ");
    scanf("%d", &pos);
    printf("Enter value: ");
    scanf("%d", &val);
    insertAt(pos, val);
    break;
case 4:
    printf("Enter value to delete: ");
    scanf("%d", &val);
    deleteNode(val);
    break;
case 5:
    display();
    break;
case 6:
    printf("Exiting program...\n");
    exit(0);
default:
    printf("Invalid choice! Try again.\n");
}
```

```
    }
    return 0;
}
```

Output:

MCQ Questions

1. In a doubly linked list, each node contains:
A) Only data
B) Data and pointer to next node
C) Data, pointer to previous node, and pointer to next node
D) Data and pointer to previous node only
Answer: …………………………………………………………………………………

2. Which pointer needs to be updated when inserting a node at the beginning of a doubly linked list?
A) Only next of new node
B) Only prev of head
C) Both next of new node and prev of old head
D) No pointer needs updating
Answer: …………………………………………………………………………………

3. If the list contains 10 ⇔ 20 ⇔ 30 and we call deleteNode(20), what will be the resulting list?
A) 10 ⇔ 30
B) 20 ⇔ 30
C) 10 ⇔ 20
D) 30
Answer: …………………………………………………………………………………

4. What happens when you try to delete a value not present in the doubly linked list?
A) Program crashes
B) First node gets deleted
C) Last node gets deleted
D) Message displayed that value not found
Answer: …………………………………………………………………………………

5. Which of the following is an advantage of doubly linked list over singly linked list?
A) Requires less memory
B) Can be traversed in both directions
C) Insertion at beginning is faster
D) Deletion does not require extra pointer
Answer: …………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 8

Write a program to implement Searching Algorithms

1. **Sequential search**
2. **Binary search**

```
#include <stdio.h>

// 1. Sequential (Linear) Search
int sequentialSearch(int arr[], int n, int key) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == key)
         return i;  // return index if found
   }
   return -1;  // not found
}

// 2. Binary Search (works only on sorted arrays)
int binarySearch(int arr[], int n, int key) {
   int low = 0, high = n - 1, mid;
   while (low <= high) {
      mid = (low + high) / 2;
      if (arr[mid] == key)
         return mid; // found
      else if (arr[mid] < key)
         low = mid + 1; // search right half
      else
         high = mid - 1; // search left half
   }
   return -1; // not found
}
```

```
// Main program
int main() {
    int arr[50], n, choice, key, pos;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements (sorted if using binary search):\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    while (1) {
        printf("\n--- SEARCHING MENU ---\n");
        printf("1. Sequential Search\n");
        printf("2. Binary Search\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter value to search: ");
            scanf("%d", &key);
            pos = sequentialSearch(arr, n, key);
            if (pos != -1)
                printf("%d found at position %d\n", key, pos + 1);
            else
                printf("%d not found in array\n", key);
            break;
        case 2:
            printf("Enter value to search: ");
            scanf("%d", &key);
            pos = binarySearch(arr, n, key);
            if (pos != -1)
                printf("%d found at position %d\n", key, pos + 1);
            else
                printf("%d not found in array\n", key);
            break;

        case 3:
            printf("Exiting program...\n");
            return 0;

        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

Output:

MCQ Questions

1. In Sequential (Linear) Search, the time complexity in the worst case is:
A) O(1)
B) O(log n)
C) O(n)
D) O(n log n)
Answer: ………………………………………………………………………………………

2. Binary search requires the array to be:
A) Unsorted
B) Sorted in ascending or descending order
C) Randomly shuffled
D) Containing only unique elements
Answer: ………………………………………………………………………………………

3. If we search for element 50 in array [10, 20, 30, 40, 50] using binary search, which mid-value is checked first?
A) 10
B) 20
C) 30
D) 50
Answer: ………………………………………………………………………………………

4. In Sequential Search, if the element is at the last position, how many comparisons are made (for array of size n)?
A) 0
B) 1
C) n
D) n – 1
Answer: ………………………………………………………………………………………

5. Which statement is true about Binary Search compared to Sequential Search?
A) Binary search is slower than sequential search.
B) Binary search works only on sorted arrays.
C) Binary search requires checking every element.
D) Sequential search requires sorted input.
Answer: ………………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 9

Write a program to implement following sorting algorithms

1.Selection sort

2.Bubble sort

3.Merge sort

4.Quick sort

```
#include <stdio.h>
#include <stdlib.h>

// Utility function to swap two numbers
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// 1. Selection Sort
void selectionSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      int min = i;
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[min])
            min = j;
      }
      swap(&arr[i], &arr[min]);
   }
   printf("Array sorted using Selection Sort.\n");
}
```

```
// 2. Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
        }
    }
    printf("Array sorted using Bubble Sort.\n");
}

// 3. Merge Sort
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

// 4. Quick Sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Function to print array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Main function
int main() {
    int arr[50], n, choice;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    while (1) {
        printf("\n--- SORTING MENU ---\n");
        printf("1. Selection Sort\n");
        printf("2. Bubble Sort\n");
        printf("3. Merge Sort\n");
        printf("4. Quick Sort\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);


int temp[50];
for (int i = 0; i < n; i++) temp[i] = arr[i]; // copy original array


switch (choice) {
case 1:
   selectionSort(temp, n);
   printArray(temp, n);
   break;
case 2:
   bubbleSort(temp, n);
   printArray(temp, n);
   break;
case 3:
   mergeSort(temp, 0, n - 1);
   printf("Array sorted using Merge Sort.\n");
   printArray(temp, n);
   break;
case 4:
   quickSort(temp, 0, n - 1);
   printf("Array sorted using Quick Sort.\n");
   printArray(temp, n);
   break;
case 5:
   printf("Exiting program...\n");
   return 0;
default:
   printf("Invalid choice! Try again.\n");
}    } }
```

Output:

MCQ Questions

1. Which of the following sorting algorithms is based on the “divide and conquer” approach?

A) Bubble Sort
B) Selection Sort
C) Merge Sort
D) Insertion Sort
Answer: ……………………………………………………………………………………

2. In Bubble Sort, how many passes are required (in the worst case) to sort an array of n elements?

A) n^2
B) n – 1
C) log n
D) n/2
Answer: ……………………………………………………………………………………

3. What is the time complexity of Quick Sort in the best case?

A) $O(n^2)$
B) O(n log n)
C) O(n)
D) O(log n)
Answer: ……………………………………………………………………………………

4. In Selection Sort, after the first pass (iteration), which element is placed in the correct position?

A) The largest element
B) The smallest element
C) The middle element
D) Random element
Answer: ……………………………………………………………………………………

5. Which sorting algorithm is considered the most efficient for large datasets?

A) Bubble Sort
B) Selection Sort
C) Quick Sort
D) Linear Search
Answer: ……………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 10

Write a program to implement breadth first search (BFS) graph traversal algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20

int queue[MAX], front = -1, rear = -1;
int visited[MAX];

// Function to enqueue an element
void enqueue(int v) {
   if (rear == MAX - 1) {
       printf("Queue overflow!\n");
       return;
   }
   if (front == -1) front = 0;
   queue[++rear] = v;
}

// Function to dequeue an element
int dequeue() {
   if (front == -1 || front > rear) {
       return -1;
   }
   return queue[front++];
}

// BFS function
void BFS(int adj[MAX][MAX], int n, int start) {
   for (int i = 0; i < n; i++) visited[i] = 0;
```

```
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int node = dequeue();
        printf("%d ", node);

        for (int j = 0; j < n; j++) {
            if (adj[node][j] == 1 && visited[j] == 0) {
                enqueue(j);
                visited[j] = 1;
            }
        }
    }
    printf("\n");
}

int main() {
    int n, start;
    int adj[MAX][MAX];

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);

    BFS(adj, n, start);

    return 0;
}
```

Output:

MCQ Questions

1. BFS uses which data structure for traversal?

A) Stack

B) Queue

C) Linked List

D) Heap

Answer: ……………………………………………………………………………………

2. In BFS, nodes are visited in what order?

A) Depth-wise

B) Random order

C) Level by level

D) Reverse order

Answer: ……………………………………………………………………………………

3. If a graph has V vertices and E edges, the time complexity of BFS is:

A) O(V + E)

B) $O(V^2)$

C) $O(E^2)$

D) O(log V)

Answer: ……………………………………………………………………………………

4. For the adjacency matrix representation of a graph, the space complexity is:

A) O(V)

B) O(E)

C) $O(V^2)$

D) O(V + E)

Answer: ……………………………………………………………………………………

5. Which of the following is a correct application of BFS?

A) Shortest path in an unweighted graph

B) Topological sorting

C) Detecting cycles in a directed graph

D) Binary tree inorder traversal

Answer: ……………………………………………………………………………………

Faculty Signature	
Date and Grade	

PRACTICAL 11

Write a program to implement depth first search (DFS) graph traversal algorithm.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 20

int visited[MAX];

// DFS function
void DFS(int adj[MAX][MAX], int n, int start) {
    printf("%d ", start);
    visited[start] = 1;

    for (int j = 0; j < n; j++) {
        if (adj[start][j] == 1 && visited[j] == 0) {
            DFS(adj, n, j);
        }
    }
}

int main() {
    int n, start;
    int adj[MAX][MAX];

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    for (int i = 0; i < n; i++) visited[i] = 0;

    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);

    printf("DFS Traversal: ");
    DFS(adj, n, start);
    printf("\n");

    return 0;
}
```

Output:

MCQ Questions

1. DFS uses which data structure (implicitly or explicitly)?

A) Queue
B) Stack
C) Linked List
D) Heap
Answer: ……………………………………………………………………………………

2. In DFS, nodes are visited in what manner?

A) Level by level
B) Depth-wise (go as far as possible before backtracking)
C) Random order
D) By shortest path first
Answer: ……………………………………………………………………………………

3. The time complexity of DFS for a graph with V vertices and E edges is:

A) $O(V^2)$
B) $O(E^2)$
C) $O(V + E)$
D) $O(\log V)$
Answer: ……………………………………………………………………………………

4. Which of the following is NOT an application of DFS?

A) Detecting cycles in a graph
B) Solving mazes/puzzles
C) Topological sorting
D) Finding shortest path in an unweighted graph
Answer: ……………………………………………………………………………………

5. If DFS is implemented using recursion, the system internally uses:

A) Queue
B) Priority Queue
C) Stack (call stack)
D) Binary Heap
Answer: ……………………………………………………………………………………

Faculty Signature	
Date and Grade	

NOTES