MANIPAL UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
Lab Manual

Course Name: Data Structure and Algorithm Lab

Course Code: AIM2130

Credits: 1

Session: 2024-2025

Course Coordinator: Mr. Sanjay kumar Tehariya

Lab 1: Programs based on 1-D Array Operations

AIM:

To perform basic operations on one-dimensional arrays: insertion, deletion, traversal, and searching.

THEORY:

A 1-D array is a data structure that holds a fixed number of values of the same type. These values are stored in contiguous memory locations. Key operations:

- **Traversal**: Visit and process each element.
- **Insertion**: Add a new element at a given position.
- **Deletion**: Remove an element from a given position.
- **Searching**: Find the location of a specific element.

PROGRAM:

```
#include<stdio.h>
#define SIZE 100

int main() {
   int arr[SIZE], n, i, pos, elem;

   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter elements: ");
   for(i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   printf("\nEnter position to insert and element: ");
   scanf("%d%d", &pos, &elem);
   for(i = n; i >= pos; i--) arr[i] = arr[i-1];
   arr[pos-1] = elem;
   n++;

   printf("\nArray after insertion: ");
   for(i = 0; i < n; i++) printf("%d ", arr[i]);

   printf("\nEnter position to delete: ");
   scanf("%d", &pos);
   for(i = pos-1; i < n-1; i++) arr[i] = arr[i+1];
   n--;
```

```
    printf("\nArray after deletion: ");
    for(i = 0; i < n; i++) printf("%d ", arr[i]);

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Enter elements: 1 2 3 4 5
Enter position to insert and element: 3 99
Array after insertion: 1 2 99 3 4 5
Enter position to delete: 4
Array after deletion: 1 2 99 4 5
```

VIVA QUESTIONS:

1. What are the advantages and disadvantages of arrays?
2. How are elements accessed in an array?
3. What is the time complexity of insertion in an array?
4. Can we change the size of an array after declaration?
5. How does memory allocation work for arrays?

Lab 2: Programs based on 2-D Array Operations

AIM:

To perform addition of two matrices using 2D arrays.

THEORY:

A 2-D array stores data in row and column format. Matrix addition involves adding corresponding elements from two matrices of the same dimension.

PROGRAM:

```
#include<stdio.h>

int main() {
    int a[10][10], b[10][10], c[10][10], i, j, r, c1;
    printf("Enter rows and columns: ");
    scanf("%d%d", &r, &c1);

    printf("Enter first matrix:\n");
    for(i = 0; i < r; i++)
        for(j = 0; j < c1; j++)
            scanf("%d", &a[i][j]);

    printf("Enter second matrix:\n");
    for(i = 0; i < r; i++)
        for(j = 0; j < c1; j++)
            scanf("%d", &b[i][j]);

    printf("Sum of matrices:\n");
    for(i = 0; i < r; i++) {
        for(j = 0; j < c1; j++) {
            c[i][j] = a[i][j] + b[i][j];
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

Enter rows and columns: 2 2
Enter first matrix:
1 2
3 4
Enter second matrix:
5 6
7 8
Sum of matrices:
6 8
10 12

VIVA QUESTIONS:

1. What is a two-dimensional array?
2. What are the applications of 2-D arrays?
3. Can matrix addition be done for matrices of different sizes?
4. How is memory allocated in 2-D arrays?

Lab 3: Searching in 1-D Array

AIM:

To implement linear and binary search algorithms.

THEORY:

- **Linear Search**: Scan each element until the target is found.
- **Binary Search**: Divide and conquer on a sorted array.

PROGRAM (Linear & Binary Search):

```
#include<stdio.h>

int linearSearch(int arr[], int n, int key) {
    for(int i = 0; i < n; i++)
        if(arr[i] == key) return i;
    return -1;
}

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while(low <= high) {
        int mid = (low + high)/2;
        if(arr[mid] == key) return mid;
        else if(arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[100], n, key, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter sorted elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &key);

    int pos = linearSearch(arr, n, key);
    printf("Linear Search: %s\n", pos != -1 ? "Found" : "Not Found");
```

```
    pos = binarySearch(arr, n, key);
    printf("Binary Search: %s\n", pos != -1 ? "Found" : "Not Found");

    return 0;
}
```

VIVA QUESTIONS:

1. What is the difference between linear and binary search?
2. When is binary search preferable?
3. What is the time complexity of both methods?

Lab 4: Sorting Techniques

AIM:

To implement sorting using the Bubble Sort technique.

THEORY:

Sorting is the process of arranging elements in ascending or descending order.

- **Bubble Sort**: Repeatedly compares adjacent elements and swaps them if they are in the wrong order.

PROGRAM:

```
#include<stdio.h>

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]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int main() {
    int arr[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    bubbleSort(arr, n);

    printf("Sorted array: ");
    for(int i = 0; i < n; i++) printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Enter elements: 3 5 2 4 1
```

Sorted array: 1 2 3 4 5

VIVA QUESTIONS:

1. What is the time complexity of Bubble Sort?
2. Is Bubble Sort stable? Why?
3. Which sorting algorithm is more efficient than Bubble Sort?

Lab 5: Linked List Operations

AIM:

To implement a singly linked list and perform insertion and traversal operations.

THEORY:

A linked list is a dynamic data structure composed of nodes, where each node contains data and a pointer to the next node.

PROGRAM:

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

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

void display(struct Node* head) {
    while(head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL, *temp, *newNode;
    int n, data;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    for(int i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data: ");
        scanf("%d", &newNode->data);
        newNode->next = NULL;

        if(head == NULL)
            head = temp = newNode;
        else {
            temp->next = newNode;
            temp = newNode;
        }
    }
```

```
    display(head);
    return 0;
}
```

OUTPUT:
Enter number of nodes: 3
Enter data: 10
Enter data: 20
Enter data: 30
10 -> 20 -> 30 -> NULL

VIVA QUESTIONS:

1. What is the difference between an array and a linked list?

2. What are the types of linked lists?

3. How is memory allocated for nodes?

Lab 6: Stack and Recursion

AIM:

To implement a stack using an array and demonstrate recursion using factorial calculation.

THEORY:

- **Stack** is a LIFO structure. Basic operations: push, pop, display.
- **Recursion** is when a function calls itself.

PROGRAM (Stack using Array):

```
#include<stdio.h>
#define SIZE 100

int stack[SIZE], top = -1;

void push(int val) {
   if(top < SIZE - 1)
      stack[++top] = val;
   else
      printf("Stack Overflow\n");
}

void pop() {
   if(top >= 0)
      printf("Popped: %d\n", stack[top--]);
   else
      printf("Stack Underflow\n");
}

void display() {
   for(int i = top; i >= 0; i--)
      printf("%d ", stack[i]);
   printf("\n");
}

int main() {
   push(10); push(20); push(30);
   display();
   pop();
   display();
   return 0;
}
```

PROGRAM (Recursion Example - Factorial):

```
int factorial(int n) {
    if(n == 0) return 1;
    return n * factorial(n - 1);
}
```

VIVA QUESTIONS:

1. What is recursion?
2. What is the difference between iteration and recursion?
3. What are the applications of stacks?

Lab 7: Queue Implementation

AIM:

To implement a queue using an array.

THEORY:

A queue is a FIFO data structure that supports enqueue and dequeue operations.

PROGRAM:

```
#include<stdio.h>
#define SIZE 100

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

void enqueue(int val) {
    if(rear < SIZE-1) {
        if(front == -1) front = 0;
        queue[++rear] = val;
    } else printf("Queue Overflow\n");
}

void dequeue() {
    if(front == -1 || front > rear)
        printf("Queue Underflow\n");
    else
        printf("Dequeued: %d\n", queue[front++]);
}

void display() {
    for(int i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}

int main() {
    enqueue(10); enqueue(20); enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

VIVA QUESTIONS:

1. What is the difference between stack and queue?
2. What is a circular queue?
3. How can we implement a queue using a linked list?

Lab 8: Tree Operations

AIM:

To implement a binary tree and perform inorder traversal.

THEORY:

A **binary tree** is a non-linear data structure where each node can have at most two children.
Traversal refers to visiting all nodes in a specific order:

- **Inorder**: Left, Root, Right
- **Preorder**: Root, Left, Right
- **Postorder**: Left, Right, Root

PROGRAM (Inorder Traversal):

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* create(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

void inorder(struct Node* root) {
    if(root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    struct Node* root = create(1);
    root->left = create(2);
    root->right = create(3);
```

```
    root->left->left = create(4);
    root->left->right = create(5);

    printf("Inorder Traversal: ");
    inorder(root);
    return 0;
}
```

OUTPUT:
Inorder Traversal: 4 2 5 1 3

VIVA QUESTIONS:

1. What is a binary tree?
2. What are the different types of tree traversals?
3. What is a complete binary tree?
4. What is the difference between a binary tree and a binary search tree?

Lab 9: Binary Search Tree (BST)

AIM:

To implement insertion and inorder traversal in a Binary Search Tree (BST).

THEORY:

A **BST** is a binary tree where:

- The left child has values less than the root.
- The right child has values greater than the root.

This allows efficient searching, insertion, and deletion.

PROGRAM:

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

struct Node {
    int data;
    struct Node *left, *right;
};

struct Node* insert(struct Node* root, int key) {
    if(root == NULL) {
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = key;
        temp->left = temp->right = NULL;
        return temp;
    }
    if(key < root->data)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

void inorder(struct Node* root) {
    if(root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
```

```
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printf("BST Inorder Traversal: ");
    inorder(root);
    return 0;
}
```

OUTPUT:
BST Inorder Traversal: 20 30 40 50 60 70 80

VIVA QUESTIONS:

1. What is a BST?

2. What are the advantages of a BST?

3. What is the time complexity of search in BST?

4. Can BST contain duplicate values?

Lab 10: Graph Operations

AIM:

To implement a graph using adjacency matrix and perform DFS and BFS traversal.

THEORY:

A **graph** is a collection of vertices (nodes) and edges (connections).

- **BFS (Breadth First Search)**: Uses a queue to explore nodes level by level.
- **DFS (Depth First Search)**: Uses recursion (or stack) to explore as far as possible along each branch.

PROGRAM:

```
#include<stdio.h>
#define SIZE 10

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

void bfs(int a[SIZE][SIZE], int n, int start) {
    for(int i = 0; i < SIZE; i++) visited[i] = 0;
    front = rear = -1;
    queue[++rear] = start;
    visited[start] = 1;
    while(front != rear) {
        start = queue[++front];
        printf("%d ", start);
        for(int i = 0; i < n; i++) {
            if(a[start][i] && !visited[i]) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}

void dfs(int a[SIZE][SIZE], int n, int start) {
    printf("%d ", start);
    visited[start] = 1;
    for(int i = 0; i < n; i++) {
        if(a[start][i] && !visited[i])
            dfs(a, n, i);
    }
```

```
}

int main() {
    int a[SIZE][SIZE], n, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            scanf("%d", &a[i][j]);

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("BFS: ");
    bfs(a, n, start);

    for(int i = 0; i < SIZE; i++) visited[i] = 0;
    printf("\nDFS: ");
    dfs(a, n, start);

    return 0;
}
```

SAMPLE ADJACENCY MATRIX (4 Vertices):
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0

OUTPUT:
BFS: 0 1 2 3
DFS: 0 1 3 2

VIVA QUESTIONS:

1. What are directed and undirected graphs?

2. What is the difference between BFS and DFS?

3. What is an adjacency matrix?

4. Where are graphs used in real-world applications?