Practical List

Sr. No.	Title of Practical
1	Implement a program to demonstrate use of variables, data types, and arrays
2	Implement Linear Search algorithm
3	Implement Binary Search algorithm
4	Implement Bubble Sort algorithm
5	Implement Insertion Sort algorithm
6	Implement Stack operations using array
7	Implement Queue operations using array
8	Implement Binary Tree and perform traversals
9	Implement Kruskal’s Algorithm for Minimum Spanning Tree
10	Implement Dynamic Programming (Fibonacci Sequence)
11	Implement Selection Sort
12	Implement Singly Linked List
13	Implement Doubly Linked List
14	Implement Circular Queue
15	Implement Postfix Evaluation
16	Implement Depth First Search (DFS)
17	Implement Breadth First Search (BFS)
18	Implement Quick Sort
19	Implement Fractional Knapsack (Greedy Method)
20	Implement Dijkstra’s Shortest Path

LAB PRACTICALS

Practical No. 1: Basic Data Types and Arrays

Objective: To understand variables, data types, and arrays.

Program:

```
#include <stdio.h>

int main() {
   int i, arr[5] = {10, 20, 30, 40, 50};
   float avg;
   int sum = 0;
   for(i = 0; i < 5; i++)
      sum += arr[i];
      avg = sum / 5.0;
   printf("Sum = %d\nAverage = %.2f\n", sum, avg);
   return 0;
}
```

Output:

Sum = 150
Average = 30.00

Practical No. 2: Linear Search

Objective: To search an element in an array using linear search.

Program:

```
#include <stdio.h>

int main() {
   int arr[5] = {5, 10, 15, 20, 25}, key, i, flag = 0;
   printf("Enter element to search: ");
   scanf("%d", &key);

   for(i = 0; i < 5; i++) {
      if(arr[i] == key) {
         printf("Element found at position %d\n", i+1);
         flag = 1;
         break;
      }
```

```
    }
    if(!flag)
        printf("Element not found\n");
    return 0;
}
```

Practical No. 3: Binary Search

Objective: To search a given element in a sorted array using binary search.

Program:

```
#include <stdio.h>
int main() {
    int arr[6] = {2, 4, 6, 8, 10, 12};
    int low = 0, high = 5, mid, key;
    printf("Enter element to search: ");
    scanf("%d", &key);
    while(low <= high) {
        mid = (low + high) / 2;
        if(arr[mid] == key) {
            printf("Element found at position %d\n", mid + 1);
            return 0;
        } else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    printf("Element not found\n");
    return 0;
}
```

Practical No. 4: Bubble Sort

Objective: To sort an array using bubble sort.

Program:

```
#include <stdio.h>
int main() {
   int arr[5] = {64, 34, 25, 12, 22}, i, j, temp;
   for(i = 0; i < 5-1; i++)
      for(j = 0; j < 5-i-1; j++)
         if(arr[j] > arr[j+1]) {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
   printf("Sorted array: ");
   for(i = 0; i < 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

Practical No. 5: Insertion Sort

Objective: To sort an array using insertion sort.

Program:

```
#include <stdio.h>
int main() {
   int arr[5] = {12, 11, 13, 5, 6};
   int i, j, key;

   for(i = 1; i < 5; i++) {
      key = arr[i];
      j = i - 1;
      while(j >= 0 && arr[j] > key) {
         arr[j+1] = arr[j];
         j--;
      }
```

```
        arr[j+1] = key;
    }
    printf("Sorted array: ");
    for(i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Practical No. 6: Stack Using Array

Objective: To implement stack operations using array.

Program:

```
#include <stdio.h>
#define MAX 5
int stack[MAX], top = -1;

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

void pop() {
    if(top == -1)
        printf("Stack Underflow\n");
    else
        printf("Popped element: %d\n", stack[top--]);
}

void display() {
    if(top == -1)
        printf("Stack Empty\n");
```

```
    else {
        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;
}
```

Practical No. 7: Queue Using Array

Objective: To implement queue operations using array.

Program:

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front = -1, rear = -1;

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

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

void display() {
   if(front == -1)
      printf("Queue Empty\n");
   else {
      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;
}
```

Practical No. 8: Binary Tree Traversals

Objective: To implement and traverse a binary tree.

Program:

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

```
struct node {
   int data;
   struct node *left, *right;
};

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

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

int main() {
   struct node* root = newNode(1);
   root->left = newNode(2);
   root->right = newNode(3);
   root->left->left = newNode(4);
   root->left->right = newNode(5);

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

Practical No. 9: Kruskal's Algorithm (MST)

Objective: To find a Minimum Spanning Tree using Kruskal's algorithm.

Program:

```
#include <stdio.h>

#define INF 999


int main() {
   int cost[4][4] = {
      {INF, 2, INF, 6},
      {2, INF, 3, 8},
      {INF, 3, INF, INF},
      {6, 8, INF, INF}
   };
   int n = 4, min, a, b, u, v, parent[4]={0}, ne = 1, mincost = 0;


   while(ne < n) {
      for(min = INF, i = 0; i < n; i++)
         for(j = 0; j < n; j++)
            if(cost[i][j] < min) {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }


      while(parent[u])
         u = parent[u];
      while(parent[v])
```

```
        v = parent[v];

      if(u != v) {
        printf("Edge %d: (%d, %d) cost:%d\n", ne++, a, b, min);
        mincost += min;
        parent[v] = u;
      }

      cost[a][b] = cost[b][a] = INF;
   }
   printf("Minimum cost = %d\n", mincost);
   return 0;
}
```

Practical No. 10: Dynamic Programming – Fibonacci

Objective: To find Fibonacci sequence using dynamic programming.
Program:

```
#include <stdio.h>
int main() {
   int n, i;
   printf("Enter n: ");
   scanf("%d", &n);
   int f[n+2];
   f[0]=0; f[1]=1;
   for(i=2;i<=n;i++)
      f[i]=f[i-1]+f[i-2];
   printf("Fibonacci(%d) = %d\n", n, f[n]);
   return 0;
}
```

Practical No. 11: Selection Sort

Objective: To sort an array using the selection sort algorithm.
Program:

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

```
int main() {
   int arr[5] = {64, 25, 12, 22, 11};
   int i, j, min, temp;
   for(i=0; i<4; i++) {
      min = i;
      for(j=i+1; j<5; j++)
         if(arr[j] < arr[min])
            min = j;
      temp = arr[i];
      arr[i] = arr[min];
      arr[min] = temp;
   }
   printf("Sorted array: ");
   for(i=0; i<5; i++) printf("%d ", arr[i]);
   return 0;
}
```

Practical No. 12: Singly Linked List Operations

Objective: To implement insertion and deletion in a singly linked list.

Program:

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

struct Node {
   int data;
```

```
   struct Node* next;
};

struct Node* head = NULL;
void insert(int val) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = val;
   newNode->next = head;
   head = newNode;
}

void delete() {
   if(head == NULL)
      printf("List is empty\n");
   else {
      struct Node* temp = head;
      head = head->next;
      free(temp);
   }
}

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

int main() {
   insert(10);
```

```
    insert(20);
    insert(30);
    display();
    delete();
    display();
    return 0;
}
```

Practical No. 13: Doubly Linked List

Objective: To implement a doubly linked list with insertion and traversal.

Program:

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

struct Node {
    int data;
    struct Node *prev, *next;
};
struct Node *head = NULL;

void insert(int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = head;
    newNode->prev = NULL;
    if(head != NULL) head->prev = newNode;
    head = newNode;
}

void display() {
    struct Node* temp = head;
    while(temp != NULL) {
```

```
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

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

Practical No. 14: Circular Queue

Objective: To implement circular queue operations.

Program:

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front = -1, rear = -1;

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

void dequeue() {
    if(front == -1)
        printf("Queue Empty\n");
```

```
    else {
        printf("Deleted: %d\n", queue[front]);
        if(front == rear) front = rear = -1;
        else front = (front + 1) % MAX;
    }
}

void display() {
    if(front == -1)
        printf("Queue Empty\n");
    else {
        int i = front;
        printf("Queue: ");
        while(i != rear) {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", queue[rear]);
    }
}

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

Practical No. 15: Evaluate Postfix Expression using Stack

Objective: To evaluate a postfix expression using stack.

Program:

```
#include <stdio.h>
#include <ctype.h>
int stack[20];
int top = -1;

void push(int x) { stack[++top] = x; }
int pop() { return stack[top--]; }

int main() {
   char exp[] = "53+82-*";
   int i;
   for(i=0; exp[i]!='\0'; i++) {
      if(isdigit(exp[i]))
         push(exp[i]-'0');
      else {
         int val2 = pop();
         int val1 = pop();
         switch(exp[i]) {
            case '+': push(val1+val2); break;
            case '-': push(val1-val2); break;
            case '*': push(val1*val2); break;
            case '/': push(val1/val2); break;
         }
      }
   }
   printf("Result = %d\n", pop());
   return 0;
}
```

Practical No. 16: Depth First Search (DFS)

Objective: To implement Depth First Search traversal on a graph.

Program:

```
#include <stdio.h>

int visited[10];

int adj[10][10];

int n;


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


int main() {
   int i, j, start;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         scanf("%d", &adj[i][j]);
   printf("Enter starting vertex: ");
   scanf("%d", &start);
   printf("DFS Traversal: ");
   dfs(start);
   return 0;
}
```

Practical No. 17: Breadth First Search (BFS)

Objective: To implement BFS traversal of a graph.

Program:

```
#include <stdio.h>
int adj[10][10], visited[10], queue[10];
int front = -1, rear = -1, n;

void enqueue(int v) {
   if(rear == 9) return;
   queue[++rear] = v;
   if(front == -1) front = 0;
}

int dequeue() {
   int v = queue[front];
   if(front == rear) front = rear = -1;
   else front++;
   return v;
}

void bfs(int start) {
   int i;
   visited[start] = 1;
   enqueue(start);
   while(front != -1) {
      int v = dequeue();
      printf("%d ", v);
      for(i=0; i<n; i++)
         if(adj[v][i] && !visited[i]) {
            enqueue(i);
            visited[i] = 1;
         }
   }
```

```
}

int main() {
   int i, j, start;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         scanf("%d", &adj[i][j]);
   printf("Enter starting vertex: ");
   scanf("%d", &start);
   printf("BFS Traversal: ");
   bfs(start);
   return 0;
}
```

Practical No. 18: Quick Sort (Divide and Conquer)

Objective: To implement quick sort using divide and conquer technique.

Program:

```
#include <stdio.h>
void swap(int *a, int *b) {
   int t = *a; *a = *b; *b = t;
}
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);
    }
}
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = 6;
    quickSort(arr, 0, n-1);
    printf("Sorted array: ");
    for(int i=0; i<n; i++) printf("%d ", arr[i]);
    return 0;
}
```

Practical No. 19: Greedy Method – Fractional Knapsack

Objective: To solve fractional knapsack using greedy method.

Program:

```
#include <stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
    float x[20], total = 0;
    int i, j, u;
    u = capacity;
```

```
    for(i = 0; i < n; i++)
        x[i] = 0.0;

    for(i = 0; i < n; i++) {
        if(weight[i] > u) break;
        else {
            x[i] = 1.0;
            u -= weight[i];
        }
    }
    if(i < n)
        x[i] = u / weight[i];

    for(i = 0; i < n; i++)
        total += x[i] * profit[i];

    printf("Maximum profit: %.2f\n", total);
}

int main() {
    float weight[3] = {10, 20, 30};
    float profit[3] = {60, 100, 120};
    float capacity = 50;
    knapsack(3, weight, profit, capacity);
    return 0;
}
```

Practical No. 20: Dijkstra's Shortest Path Algorithm

Objective: To find the shortest path using Dijkstra's algorithm.

Program:

```
#include <stdio.h>
#define INF 999
```

```
void dijkstra(int G[10][10], int n, int start) {
   int cost[10][10], distance[10], visited[10], count, min, next, i, j;
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         cost[i][j] = (G[i][j]==0)? INF : G[i][j];
   for(i=0;i<n;i++){
      distance[i]=cost[start][i];
      visited[i]=0;
   }
   distance[start]=0;
   visited[start]=1;
   count=1;
   while(count<n-1){
      min=INF;
      for(i=0;i<n;i++)
         if(distance[i]<min && !visited[i]){
            min=distance[i];
            next=i;
         }
      visited[next]=1;
      for(i=0;i<n;i++)
         if(!visited[i])
            if(min+cost[next][i]<distance[i])
               distance[i]=min+cost[next][i];
      count++;
   }
   printf("Vertex\tDistance from Source\n");
   for(i=0;i<n;i++)
      printf("%d\t\t%d\n", i, distance[i]);
}

int main(){
```

```
    int G[5][5]={{0,10,0,30,100},
                 {10,0,50,0,0},
                 {0,50,0,20,10},
                 {30,0,20,0,60},
                 {100,0,10,60,0}};
    dijkstra(G,5,0);
    return 0;
}
```