DSA PRACTICAL

1. Write a menu driven program to perform following

operations on singly linked list: Create, Insert, Delete, and Display.

```
#include <iostream>
using namespace std;
struct Node {
 int data;
 Node* next;
};
Node* head = NULL;
void insert(int x) {
 Node* temp = new Node();
 temp->data = x;
 temp->next = head;
 head = temp;
}
```

```
void Delete(int n) {
 Node* temp1 = head;
 if(n == 1) {
 head = temp1->next;
 delete temp1;
 return;
 }
 for(int i=0; i<n-2; i++) {
 temp1 = temp1->next;
 }
 Node* temp2 = temp1->next;
 temp1->next = temp2->next;
 delete temp2;
}
void display() {
 Node* temp = head;
 while(temp != NULL) {
 cout << temp->data << " ";
 temp = temp->next;
```

```
}
cout << endl;
}
int main() {
int choice, x, n;
while(1) {
cout << "1. Insert" << endl;
cout << "2. Delete" << endl;
cout << "3. Display" << endl;
cout << "4. Exit" << endl;
cout << "Enter your choice: ";
cin >> choice;
switch(choice) {
case 1: cout << "Enter the element: ";
cin >> x;
insert(x);
break;
case 2: cout << "Enter the element you want to
delete: ";
```

```
 cin >> n;
 Delete(n);
 break;
 case 3: display();
 break;
 case 4: exit(0);
 default: cout << "Invalid Input" << endl;
 }
 }
 return 0;
}
```

2.sequential search

```
#include <iostream>
using namespace std;
int sequentialSearch(int array[], int size, int key) {
 for (int i = 0; i < size; i++) {
 if (array[i] == key) {
 return i; // return the index of the key if found
```

```
 }
 }
 return -1; // return -1 if key is not found
}
int main() {
 int array[] = {1, 2, 3, 4, 5};
 int size = sizeof(array) / sizeof(array[0]);
 int key = 3;
 int index = sequentialSearch(array, size, key);
 if (index != -1) {
 cout << "Key found at index " << index << endl;
 } else {
 cout << "Key not found" << endl;
 }
 return 0;
}
```

3.Binary Search

```
#include <iostream>
using namespace std;
int binarySearch(int arr[], int n, int key) {
 int left = 0, right = n - 1;
 while (left <= right) {
 int mid = (left + right) / 2;
 if (arr[mid] == key) {
 return mid;
 }
 else if (arr[mid] < key) {
 left = mid + 1;
 }
 else {
 right = mid - 1;
 }
 }
 return -1;
}
int main() {
```

```
 int arr[] = {1, 2, 3, 4, 5};
 int n = sizeof(arr) / sizeof(arr[0]);
 int key = 3;
 int index = binarySearch(arr, n, key);
 if (index != -1) {
 cout << "Element found at index " << index <<
endl;
 }
 else {
 cout << "Element not found" << endl;
 }
 return 0;
}
```

4.Implement circular queue using arrays.

```
#include <iostream>
using namespace std;
class CircularQueue {
 int *queue, size, front, rear;
```

```
public:
 CircularQueue(int s) {
 size = s;
 queue = new int[size];
 front = rear = -1;
 }
 void enqueue(int x);
 int dequeue();
 void display();
};
void CircularQueue::enqueue(int x) {
 if ((front == 0 && rear == size - 1) || (front ==
rear + 1)) {
 cout << "Queue is full\n";
 return;
 }
 else if (front == -1) {
 front = rear = 0;
 }
```

```
 else if (rear == size - 1 && front != 0) {
 rear = 0;
 }
 else {
 rear++;
 }
 queue[rear] = x;
}
int CircularQueue::dequeue() {
 if (front == -1) {
 cout << "Queue is empty\n";
 return -1;
 }
 int x = queue[front];
 if (front == rear) {
 front = rear = -1;
 }
 else if (front == size - 1) {
 front = 0;
```

```
 }
 else {
 front++;
 }
 return x;
}
void CircularQueue::display() {
 if (front == -1) {
 cout << "Queue is empty\n";
 return;
 }
 if (rear >= front) {
 for (int i = front; i <= rear; i++)
 cout << queue[i] << " ";
 }
 else {
 for (int i = front; i < size; i++)
 cout << queue[i] << " ";
 for (int i = 0; i <= rear; i++)
```

```
 cout << queue[i] << " ";
 }
}
int main() {
 CircularQueue q(5);
 q.enqueue(1);
 q.enqueue(2);
 q.enqueue(3);
 q.enqueue(4);
 q.enqueue(5);
 q.enqueue(6);
 q.display();
 cout << endl;
 q.dequeue();
 q.dequeue();
 q.display();
 cout << endl;
 return 0;
}
```

5.Write a menu driven program to perform following operations on singly linked list: Create, reverse, search, count and Display

```
#include <iostream>
using namespace std;
struct Node {
 int data;
 Node* next;
};
class LinkedList {
private:
 Node* head;
 int count;
public:
 LinkedList() {
 head = NULL;
 count = 0;
 }
```

```
void create() {
int data;
cout << "Enter the data for the node: ";
cin >> data;
Node* newNode = new Node();
newNode->data = data;
newNode->next = head;
head = newNode;
count++;
}
void reverse() {
Node* prev = NULL;
Node* current = head;
Node* next = NULL;
while (current != NULL) {
next = current->next;
current->next = prev;
prev = current;
current = next;
```

```
}
head = prev;
}
int search(int key) {
Node* current = head;
int index = 0;
while (current != NULL) {
if (current->data == key) {
return index;
}
current = current->next;
index++;
}
return -1;
}
int countNodes() {
return count;
}
void display() {
```

```
 Node* current = head;
 while (current != NULL) {
 cout << current->data << " ";
 current = current->next;
 }
 cout << endl;
 }
};
int main() {
 int choice;
 LinkedList list;
 while (true) {
 cout << "1. Create Node" << endl;
 cout << "2. Reverse List" << endl;
 cout << "3. Search Element" << endl;
 cout << "4. Count Nodes" << endl;
 cout << "5. Display List" << endl;
 cout << "6. Exit" << endl;
 cout << "Enter your choice: ";
```

```
cin >> choice;
switch (choice) {
case 1:
list.create();
break;
case 2:
list.reverse();
break;
case 3: {
int key;
cout << "Enter the element to be searched: ";
cin >> key;
int index = list.search(key);
if (index == -1) {
cout << "Element not found." << endl;
} else {
cout << "Element found at index: " << index << endl;
}
```

```
 break;
 }
 case 4:
 cout << "Number of nodes: " <<
list.countNodes() << endl;
 break;
 case 5:
 list.display();
 break;
 case 6:
 return 0;
 default:
 cout << "Invalid choice. Please enter a valid
choice." << endl;
 }
 }
 return 0;
}
```

6. Create binary tree and perform recursive traversals.

```
#include <iostream>
using namespace std;
// Structure for a node of a binary tree
struct Node {
 int data;
 Node* left;
 Node* right;
};
// Function to create a new node and return its address
Node* getNewNode(int data) {
 Node* newNode = new Node();
 newNode->data = data;
 newNode->left = newNode->right = NULL;
 return newNode;
}
```

```
// Recursive function to do pre-order traversal of the binary tree
void preOrder(Node* root) {
 if (root == NULL) return;
 cout << root->data << " ";
 preOrder(root->left);
 preOrder(root->right);
}
// Recursive function to do in-order traversal of the binary tree
void inOrder(Node* root) {
 if (root == NULL) return;
 inOrder(root->left);
 cout << root->data << " ";
 inOrder(root->right);
}
// Recursive function to do post-order traversal of the binary tree
void postOrder(Node* root) {
```

```
 if (root == NULL) return;
 postOrder(root->left);
 postOrder(root->right);
 cout << root->data << " ";
}
int main() {
 Node* root = getNewNode(1);
 root->left = getNewNode(2);
 root->right = getNewNode(3);
 root->left->left = getNewNode(4);
 root->left->right = getNewNode(5);
 cout << "Pre-order traversal: ";
 preOrder(root);
 cout << endl;
 cout << "In-order traversal: ";
 inOrder(root);
 cout << endl;
 cout << "Post-order traversal: ";
 postOrder(root);
```

```
 cout << endl;
 return 0;
}
```

7.Implement Linked queue

```
#include <bits/stdc++.h>
using namespace std;
struct QNode {
 int data;
 QNode* next;
 QNode(int d)
 {
 data = d;
 next = NULL;
 }
};
struct Queue {
 QNode *front, *rear;
 Queue() { front = rear = NULL; }
```

```
void enQueue(int x)
{
// Create a new LL node
QNode* temp = new QNode(x);
// If queue is empty, then
// new node is front and rear both
if (rear == NULL) {
front = rear = temp;
return;
}
// Add the new node at
// the end of queue and change rear
rear->next = temp;
rear = temp;
}
// Function to remove
// a key from given queue q
void deQueue()
{
```

```
 // If queue is empty, return NULL.
 if (front == NULL)
 return;
 // Store previous front and
 // move front one node ahead
 QNode* temp = front;
 front = front->next;
 // If front becomes NULL, then
 // change rear also as NULL
 if (front == NULL)
 rear = NULL;
 delete (temp);
 }
};
// Driver code
int main()
{
 Queue q;
 q.enQueue(10);
```

```
 q.enQueue(20);
 q.deQueue();
 q.deQueue();
 q.enQueue(30);
 q.enQueue(40);
 q.enQueue(50);
 q.deQueue();
 cout << "Queue Front : " << (q.front)->data <<
endl;
 cout << "Queue Rear : " << (q.rear)->data;
}
```

8.Create binary tree. Find height of the tree and print leaf nodes. Find mirror image, print original and mirror image using level-wise printing.

```
#include <bits/stdc++.h>
using namespace std;
/* A binary tree node has data, pointer
```

```
to left child and a pointer to right child */
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};
struct Node* newNode(int data)
{
   struct Node* node
      = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return (node);
}
void mirror(struct Node* node)
{
   if (node == NULL)
      return;
```

```
    else {
        struct Node* temp;
        /* do the subtrees */
        mirror(node->left);
        mirror(node->right);
        /* swap the pointers in this node */
        temp = node->left;
        node->left = node->right;
        node->right = temp;
    }
}
/* Helper function to print
Inorder traversal.*/
void inOrder(struct Node* node)
{
    if (node == NULL)
        return;
    inOrder(node->left);
    cout << node->data << " ";
```

```
    inOrder(node->right);
}
// Driver Code
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);
    /* Print inorder traversal of the input tree */
    cout << "Inorder traversal of the constructed"
        << " tree is" << endl;
    inOrder(root);
    /* Convert tree to its mirror */
    mirror(root);
    /* Print inorder traversal of the mirror tree */
    cout << "\nInorder traversal of the mirror tree"
        << " is \n";
```

```
    inOrder(root);
    return 0;
}
```

9.Implement shortest path algorithm

```
#include <bits/stdc++.h>
using namespace std;
void printSolution(int dist[], int n) {
    cout << "Vertex Distance from Source\n";
    for (int i = 0; i < n; i++)
        cout << i << " \t\t " << dist[i] << endl;
}
void shortestPath(int graph[5][5], int src, int dest, int n) {
    int dist[5];
    bool sptSet[5];

    for (int i = 0; i < n; i++) {
        dist[i] = INT_MAX;
```

```
        sptSet[i] = false;
    }
    dist[src] = 0;
    for (int count = 0; count < n - 1; count++) {
        int u = -1;
        for (int i = 0; i < n; i++)
            if (sptSet[i] == false && dist[i] < INT_MAX)
                u = i;

        if (u == -1)
            break;
        sptSet[u] = true;
        for (int v = 0; v < n; v++)
            if (sptSet[v] == false && graph[u][v] != 0
&& dist[u] != INT_MAX && dist[u] + graph[u][v] <
dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist, n);
```

```
}
int main() {
    int graph[5][5] = { {0, 9, 75, 0, 0},
                        {9, 0, 95, 19, 0},
                        {75, 95, 0, 55, 15},
                        {0, 19, 55, 0, 25},
                        {0, 0, 15, 25, 0} };

    int src = 0;
    int dest = 4;
    int n = sizeof(graph) / sizeof(graph[0]);
    shortestPath(graph, src, dest, n);
    return 0;
}
```

10. Implement minimum cost spanning tree algorithm.

```
 #include <iostream>
 #include<bits/stdc++.h>
```

```
#include <cstring>
using namespace std;
// number of vertices in graph
#define V 7
// create a 2d array of size 7x7
//for adjacency matrix to represent graph

int main () {
  // create a 2d array of size 7x7
//for adjacency matrix to represent graph
  int G[V][V] = {
  {0,28,0,0,0,10,0},
{28,0,16,0,0,0,14},
{0,16,0,12,0,0,0},
{0,0,12,22,0,18},
{0,0,0,22,0,25,24},
{10,0,0,0,25,0,0},
{0,14,0,18,24,0,0}
};
```

```
  int edge;            // number of edge
  // create an array to check visited vertex
  int visit[V];
  //initialise the visit array to false
 for(int i=0;i<V;i++){
   visit[i]=false;
}
 // set number of edge to 0
  edge = 0;
  // the number of edges in minimum spanning
tree will be
  // always less than (V -1), where V is the
number of vertices in
  //graph
  // choose 0th vertex and make it true
  visit[0] = true;
  int x;          //  row number
  int y;          //  col number
  // print for edge and weight
```

```
  cout << "Edge" << " : " << "Weight";
  cout << endl;
  while (edge < V - 1) {//in spanning tree consist
the V-1 number of edges

  //For every vertex in the set S, find the all
adjacent vertices
  // calculate the distance from the vertex
selected.
  // if the vertex is already visited, discard it
otherwise
  //choose another vertex nearest to selected
vertex.
      int min = INT_MAX;
      x = 0;
      y = 0;
      for (int i = 0; i < V; i++) {
        if (visit[i]) {
            for (int j = 0; j < V; j++) {
```

```
            if (!visit[j] && G[i][j]) { // not in selected
and there is an edge
              if (min > G[i][j]) {
                min = G[i][j];
                x = i;
                y = j;
              }
            }
          }
        }
      }
      cout << x <<  " ---> " << y << " :  " << G[x][y];
      cout << endl;
      visit[y] = true;
      edge++;
    }

  return 0;
}
```