Q.1) a) Implement a stack using array with the following operations: push(), pop(), peek(), is\_empty().

#include <iostream.h>

#include <conio.h>

#define MAX 1000

class Stack {

private:

    int arr[MAX];

    int top;

public:

    Stack() {

        top = -1;

    }

    void push(int value) {

        if (top >= MAX - 1) {

            cout << "Overflow" << endl;

            return;

        }

        top++;

        arr[top] = value;

    }

    void pop() {

        if (top == -1) {

            cout << "Underflow" << endl;

            return;

        }

        top--;

    }

    void peek() {

        if (top == -1) {

            cout << "No Element Present in stack" << endl;

            return;

        }

        cout << arr[top] << endl;

    }

    void isEmpty() {

        if (top == -1) {

            cout << "Is EMPTY" << endl;

        } else {

            cout << "Not Empty" << endl;

        }

    }

    void display() {

        if (top == -1) {

            cout << "Stack is Empty" << endl;

            return;

    }

    for (int i = top; i >= 0; i--) {

        cout << arr[i] << "  ";

    }

    cout << endl;

    }

};

int main() {

    clrscr();

    cout<< "Shiv Arora"<< endl;

    Stack st;

    int choice, value;

    int flag = 0;

    cout << "\n1. Push\n2. Pop\n3. Peek\n4. Is Empty\n5. Exit\n6. Display" << endl;

    while (flag == 0) {

    cin >> choice;

    switch (choice) {

        case 1:

        cout << "Enter a number: ";

        cin >> value;

        st.push(value);

        break;

        case 2:

        st.pop();

                break;

            case 3:

                st.peek();

                break;

            case 4:

                st.isEmpty();

                break;

            case 5:

                flag = 1;

                break;

            case 6:

                st.display();

                break;

            default:

                cout << "Invalid Choice" << endl;

        }

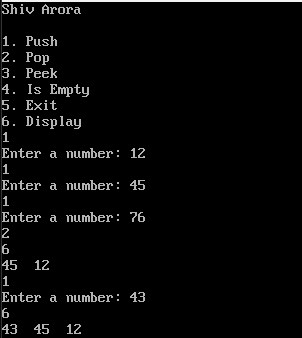
    }

    getch();

    return 0;

}

OUTPUT:



b) Implement a stack using linked list using same operations.

#include <iostream.h>

#include <conio.h>

#define MAX 1000

class Node {

public:

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = NULL;

    }

};

class Stack {

private:

    Node\* top;

public:

    Stack() {

        top = NULL;

    }

    void push(int value) {

        Node\* newNode = new Node(value);

        newNode->next = top;

        top = newNode;

    }

    void pop() {

        if (top == NULL) {

            cout << "Underflow" << endl;

            return;

        }

        Node\* temp = top;

        top = top->next;

        delete temp;

    }

    void peek() {

        if (top == NULL) {

            cout << "No Element Present in stack" << endl;

            return;

        }

        cout << top->data << endl;

    }

    void isEmpty() {

        if (top == NULL) {

            cout << "Is EMPTY" << endl;

        } else {

            cout << "Not Empty" << endl;

        }

    }

    void display() {

        Node\* temp = top;

        while (temp != NULL) {

            cout << temp->data << "  ";

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    clrscr();

    cout << "Shiv Arora" << endl;

    Stack st;

    int choice, value;

    int flag = 0;

    cout << "\n1. Push\n2. Pop\n3. Peek\n4. Is Empty\n5. Exit\n6. Display" << endl;

    while (flag == 0) {

    cin >> choice;

    switch (choice) {

            case 1:

                cout << "Enter a number: ";

                cin >> value;

                st.push(value);

                break;

            case 2:

                st.pop();

                break;

            case 3:

                st.peek();

                break;

            case 4:

                st.isEmpty();

                break;

            case 5:

                flag = 1;

                break;

            case 6:

                st.display();

                break;

            default:

                cout << "Invalid Choice" << endl;

        }

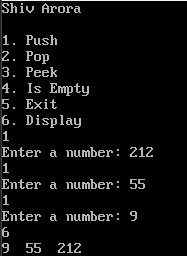
    }

    getch();

    return 0;

}

OUTPUT:



Q.2) a) Implement a queue using an array with the following operations: enqueue(), dequeue(), front(), is\_empty().

#include <iostream.h>

#include <conio.h>

#define MAX 1000

class Queue {

private:

    int front, rear;

    int queue[MAX];

public:

    Queue() {

        front = -1;

        rear = -1;

    }

    void enqueue(int value) {

        if (rear == MAX - 1) {

            cout << "Overflow" << endl;

            return;

        }

        if (front == -1) {

            front = 0;

        }

        rear++;

        queue[rear] = value;

    }

    void dequeue() {

        if (front == -1 || front > rear) {

            cout << "Underflow" << endl;

            return;

        }

        front++;

    }

    void frontElement() {

        if (front == -1 || front > rear) {

            cout << "No Element Present in queue" << endl;

            return;

        }

        cout << queue[front] << endl;

    }

    void isEmpty() {

        if (front == -1 || front > rear) {

            cout << "Is EMPTY" << endl;

        } else {

            cout << "Not Empty" << endl;

        }

    }

    void display() {

        if (front == -1 || front > rear) {

            cout << "Queue is Empty" << endl;

            return;

        }

        for (int i = front; i <= rear; i++) {

            cout << queue[i] << "  ";

        }

        cout << endl;

    }

};

int main() {

    Queue q;

    int choice, value;

    int flag = 0;

    clrscr();

    cout << "Shiv Arora"<< endl;

     cout << "\n1. Enqueue\n2. Dequeue\n3. Front\n4. Is\_Empty\n5. Display\n6. Exit" << endl;

    while (flag == 0) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter a number: ";

                cin >> value;

                q.enqueue(value);

                break;

            case 2:

                q.dequeue();

                break;

            case 3:

                q.frontElement();

                break;

            case 4:

                q.isEmpty();

                break;

            case 5:

                q.display();

                break;

            case 6:

                flag = 1;

                break;

            default:

                cout << "Invalid Choice" << endl;

        }

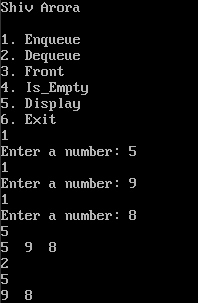
    }

    getch();

    return 0;

}

OUTPUT:



b) Implement a circular queue using an array.

#include <iostream.h>

#include <conio.h>

#define MAX 1000

class CircularQueue {

private:

    int front, rear;

    int queue[MAX];

public:

    CircularQueue() {

        front = -1;

        rear = -1;

    }

    void enqueue(int value) {

        if ((rear + 1) % MAX == front) {

            cout << "Overflow" << endl;

            return;

        }

        if (front == -1) {

            front = 0;

        }

        rear = (rear + 1) % MAX;

        queue[rear] = value;

    }

    void dequeue() {

        if (front == -1) {

            cout << "Underflow" << endl;

            return;

        }

        if (front == rear) {

            front = -1;

            rear = -1;

        } else {

            front = (front + 1) % MAX;

        }

    }

    void frontElement() {

        if (front == -1) {

            cout << "No Element Present in queue" << endl;

            return;

        }

        cout << queue[front] << endl;

    }

    void isEmpty() {

        if (front == -1) {

            cout << "Is EMPTY" << endl;

        } else {

            cout << "Not Empty" << endl;

        }

    }

    void display() {

        if (front == -1) {

            cout << "Queue is Empty" << endl;

            return;

        }

        for (int i = front; ; i = (i + 1) % MAX) {

            cout << queue[i] << "  ";

            if (i == rear) break;

        }

        cout << endl;

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    CircularQueue q;

    int choice, value;

    int flag = 0;

    cout << "\n1. Enqueue\n2. Dequeue\n3. Front\n4. Is\_Empty\n5. Display\n6. Exit" << endl;

    while (flag == 0) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter a number: ";

                cin >> value;

                q.enqueue(value);

                break;

            case 2:

                q.dequeue();

                break;

            case 3:

                q.frontElement();

                break;

            case 4:

                q.isEmpty();

                break;

            case 5:

                q.display();

                break;

            case 6:

                flag = 1;

                break;

            default:

                cout << "Invalid Choice" << endl;

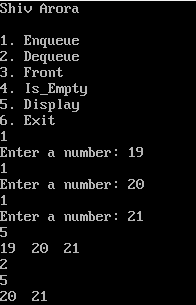
        }

    }

    getch();

}

OUTPUT:



Q.3) a) Implement a singly linked list with the following operations: insert\_at\_head(), insert\_at\_tail(), delete\_node(), search(), display()

#include <iostream.h>

#include <conio.h>

#define MAX 1000

class Node {

public:

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = NULL;

    }

};

class LinkedList {

private:

    Node\* head;

public:

    LinkedList() {

        head = NULL;

    }

    void insertAtHead(int value) {

        Node\* newNode = new Node(value);

        newNode->next = head;

        head = newNode;

    }

    void insertAtTail(int value) {

        Node\* newNode = new Node(value);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void deleteNode(int value) {

        if (head == NULL) return;

        if (head->data == value) {

            Node\* temp = head;

            head = head->next;

            delete temp;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL && temp->next->data != value) {

            temp = temp->next;

        }

        if (temp->next == NULL) return;

        Node\* toDelete = temp->next;

        temp->next = temp->next->next;

        delete toDelete;

    }

    void searchByValue(int value) {

        Node\* temp = head;

        while (temp != NULL) {

            if (temp->data == value) {

                cout << "Value found" << endl;

                return;

            }

            temp = temp->next;

        }

        cout << "Value not found" << endl;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

            cout << temp->data << " -> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    LinkedList list;

    int choice, value;

    int flag = 0;

    cout << "\n1. Insert at Head\n2. Insert at Tail\n3. Delete Node\n4. Search by Value\n5. Display\n6. Exit\n";

    while (flag == 0) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtHead(value);

                break;

            case 2:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtTail(value);

                break;

            case 3:

                cout << "Enter value to delete: ";

                cin >> value;

                list.deleteNode(value);

                break;

            case 4:

                cout << "Enter value to search: ";

                cin >> value;

                list.searchByValue(value);

                break;

            case 5:

                list.display();

                break;

            case 6:

                flag = 1;

                break;

            default:

                cout << "Invalid Choice" << endl;

        }

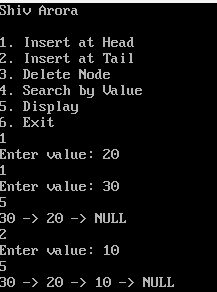
    }

    getch();

    return 0;

}

OUTPUT:



b) Extend your implementation to a doubly linked list with prevpointers.

#include <iostream.h>

#include <stddef.h>

#include <conio.h>

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int value) {

        data = value;

        next = NULL;

        prev = NULL;

    }

};

class LinkedList {

private:

    Node\* head;

public:

    LinkedList() {

        head = NULL;

    }

    void insertAtHead(int value) {

        Node\* newNode = new Node(value);

        newNode->next = head;

        if (head != NULL) {

            head->prev = newNode;

        }

        head = newNode;

    }

    void insertAtTail(int value) {

        Node\* newNode = new Node(value);

        if (head == NULL) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->prev = temp;

    }

    void deleteNode(int value) {

        if (head == NULL) return;

        if (head->data == value) {

            Node\* temp = head;

            head = head->next;

            if (head != NULL) {

                head->prev = NULL;

            }

            delete temp;

            return;

        }

        Node\* temp = head;

        while (temp != NULL && temp->data != value) {

            temp = temp->next;

        }

        if (temp == NULL) return;

        if (temp->next != NULL) {

            temp->next->prev = temp->prev;

        }

        if (temp->prev != NULL) {

            temp->prev->next = temp->next;

        }

        delete temp;

    }

    void searchByValue(int value) {

        Node\* temp = head;

        while (temp != NULL) {

            if (temp->data == value) {

                cout << "Value found" << endl;

                return;

            }

            temp = temp->next;

        }

        cout << "Value not found" << endl;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

            cout << temp->data << " -> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    LinkedList list;

    int choice, value;

    int flag = 0;

    cout << "\n1. Insert at Head\n2. Insert at Tail\n3. Delete Node\n4. Search by Value\n5. Display\n6. Exit\n";

    while (flag == 0) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtHead(value);

                break;

            case 2:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtTail(value);

                break;

            case 3:

                cout << "Enter value to delete: ";

                cin >> value;

                list.deleteNode(value);

                break;

            case 4:

                cout << "Enter value to search: ";

                cin >> value;

                list.searchByValue(value);

                break;

            case 5:

                list.display();

                break;

            case 6:

                flag = 1;

                break;

            default:

                cout << "Invalid Choice" << endl;

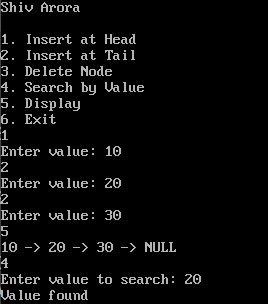
        }

    }

    getch();

}

OUTPUT:



Q.4) Implement a binary search tree.

#include <iostream.h>

#include <conio.h>

#include<stddef.h>

#include <stdlib.h>

class Node {

public:

    int data;

    Node\* left;

    Node\* right;

    Node(int value) {

    data = value;

    left = right = NULL;

    }

};

class BST {

private:

    Node\* root;

    Node\* insert(Node\* node, int value) {

    if (node == NULL) {

        return new Node(value);

    }

    if (value < node->data) {

        node->left = insert(node->left, value);

    } else {

        node->right = insert(node->right, value);

    }

    return node;

    }

    Node\* findMin(Node\* node) {

    while (node->left != NULL) {

        node = node->left;

    }

    return node;

    }

    Node\* deleteNode(Node\* node, int value) {

    if (node == NULL) return node;

    if (value < node->data) {

        node->left = deleteNode(node->left, value);

    } else if (value > node->data) {

        node->right = deleteNode(node->right, value);

    } else {

        if (node->left == NULL) {

        Node\* temp = node->right;

        delete node;

        return temp;

        } else if (node->right == NULL) {

        Node\* temp = node->left;

        delete node;

        return temp;

        }

        Node\* temp = findMin(node->right);

        node->data = temp->data;

        node->right = deleteNode(node->right, temp->data);

    }

    return node;

    }

    Node\* search(Node\* node, int value) {

    if (node == NULL || node->data == value) {

        return node;

    }

    if (value < node->data) {

        return search(node->left, value);

    }

    return search(node->right, value);

    }

    void inorder(Node\* node) {

    if (node != NULL) {

        inorder(node->left);

        cout << node->data << " ";

        inorder(node->right);

    }

    }

    void preorder(Node\* node) {

    if (node != NULL) {

        cout << node->data << " ";

        preorder(node->left);

        preorder(node->right);

    }

    }

    void postorder(Node\* node) {

    if (node != NULL) {

        postorder(node->left);

        postorder(node->right);

        cout << node->data << " ";

    }

    }

public:

    BST() {

    root = NULL;

    }

    void insert(int value) {

    root = insert(root, value);

    }

    void deleteNode(int value) {

    root = deleteNode(root, value);

    }

    int search(int value) {

    if(search(root, value) != NULL){

        return 1;

    }

    return 0;

    }

    void inorder() {

        inorder(root);

        cout << endl;

    }

    void preorder() {

        preorder(root);

        cout << endl;

    }

    void postorder() {

        postorder(root);

        cout << endl;

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    BST bst;

    int choice, value;

    cout << "\n1. Insert\n2. Delete\n3. Search\n4. Inorder\n5. Preorder\n6. Postorder\n7. Exit\n";

    while (1) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert: ";

                cin >> value;

                bst.insert(value);

                break;

            case 2:

                cout << "Enter value to delete: ";

                cin >> value;

                bst.deleteNode(value);

                break;

            case 3:

                cout << "Enter value to search: ";

                cin >> value;

                if (bst.search(value))

                    cout << "Found" << endl;

                else

                    cout << "Not Found" << endl;

                break;

            case 4:

                bst.inorder();

                break;

            case 5:

                bst.preorder();

                break;

            case 6:

                bst.postorder();

                break;

            case 7:

                exit(0);

            default:

                cout << "Invalid Choice" << endl;

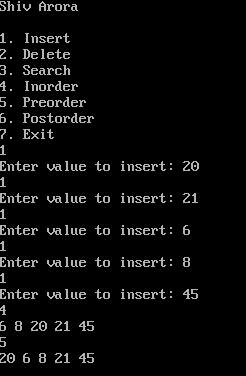
        }

    }

    getch();

}

OUTPUT:



Q.5) Implement a hash table using simple modulo based hash function and chaining for collision resolution.

#include <iostream.h>

#include <conio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

class Node {

public:

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = NULL;

    }

};

class HashTable {

private:

    Node\* table[TABLE\_SIZE];

    int hashFunction(int key) {

        return key % TABLE\_SIZE;

    }

public:

    HashTable() {

        for (int i = 0; i < TABLE\_SIZE; i++) {

            table[i] = NULL;

        }

    }

    void insert(int value) {

        int index = hashFunction(value);

        Node\* newNode = new Node(value);

        newNode->next = table[index];

        table[index] = newNode;

    }

    void deleteValue(int value) {

        int index = hashFunction(value);

        Node\* temp = table[index];

        Node\* prev = NULL;

        while (temp != NULL && temp->data != value) {

            prev = temp;

            temp = temp->next;

        }

        if (temp == NULL) return;

        if (prev == NULL) {

            table[index] = temp->next;

        } else {

            prev->next = temp->next;

        }

        delete temp;

    }

    int search(int value) {

    int index = hashFunction(value);

    Node\* temp = table[index];

    while (temp != NULL) {

        if (temp->data == value) return 1;

        temp = temp->next;

    }

    return 0;

    }

    void display() {

        cout << endl;

        for (int i = 0; i < TABLE\_SIZE; i++) {

            cout << i << " -> ";

            Node\* temp = table[i];

            while (temp != NULL) {

                cout << temp->data << " -> ";

                temp = temp->next;

            }

            cout << "NULL" << endl;

        }

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    HashTable ht;

    int choice, value;

    cout << "\n1. Insert\n2. Delete\n3. Search\n4. Display\n5. Exit\n";

    while (1) {

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert: ";

                cin >> value;

                ht.insert(value);

                break;

            case 2:

                cout << "Enter value to delete: ";

                cin >> value;

                ht.deleteValue(value);

                break;

            case 3:

                cout << "Enter value to search: ";

                cin >> value;

                if (ht.search(value))

                    cout << "Found" << endl;

                else

                    cout << "Not Found" << endl;

                break;

            case 4:

                ht.display();

                break;

            case 5:

                exit(0);

            default:

                cout << "Invalid Choice" << endl;

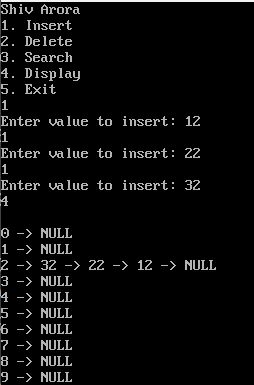
        }

    }

    getch();

}

OUTPUT:



Q.6) a) Implement graph using adjacency matrix and adjacency list representations.

#include <iostream.h>

#include <stdlib.h>

#include <conio.h>

#define MAX 10

class Node {

public:

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = NULL;

    }

};

class Graph {

private:

    int adjMatrix[MAX][MAX];

    Node\* adjList[MAX];

    int vertices;

public:

    Graph(int n) {

        vertices = n;

        initMatrix();

        initList();

    }

    void initMatrix() {

        for (int i = 0; i < vertices; i++) {

            for (int j = 0; j < vertices; j++) {

                adjMatrix[i][j] = 0;

            }

        }

    }

    void addEdgeMatrix(int u, int v) {

        adjMatrix[u][v] = 1;

        adjMatrix[v][u] = 1;

    }

    void displayMatrix() {

        cout << "Adjacency Matrix:\n";

        for (int i = 0; i < vertices; i++) {

            for (int j = 0; j < vertices; j++) {

                cout << adjMatrix[i][j] << " ";

            }

            cout << "\n";

        }

    }

    void initList() {

        for (int i = 0; i < vertices; i++) {

            adjList[i] = NULL;

        }

    }

    void addEdgeList(int u, int v) {

        Node\* newNode = new Node(v);

        newNode->next = adjList[u];

        adjList[u] = newNode;

        newNode = new Node(u);

        newNode->next = adjList[v];

        adjList[v] = newNode;

    }

    void displayList() {

        cout << "\nAdjacency List:\n";

        for (int i = 0; i < vertices; i++) {

            cout << i << " -> ";

            Node\* temp = adjList[i];

            while (temp != NULL) {

                cout << temp->data << " ";

                temp = temp->next;

            }

            cout << "\n";

        }

    }

};

void main() {

    clrscr();

    int n, edges, u, v;

    cout << "Shiv Arora"<< endl;

    cout << "Enter number of vertices: ";

    cin >> n;

    Graph g(n);

    cout << "Enter number of edges: ";

    cin >> edges;

    cout << "Enter edges (u v):\n";

    for (int i = 0; i < edges; i++) {

        cin >> u >> v;

        g.addEdgeMatrix(u, v);

        g.addEdgeList(u, v);

    }

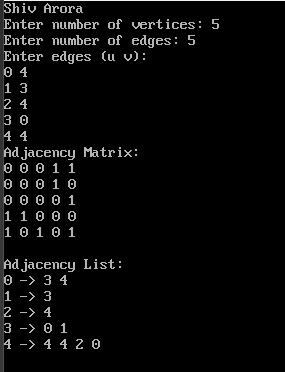
    g.displayMatrix();

    g.displayList();

    getch();

}

OUTPUT:



b) Implement breadth first search and depth first search.

#include <iostream.h>

#include <conio.h>

#define MAX 100

class Graph {

private:

    int n;

    int adjMatrix[MAX][MAX];

    int visited[MAX];

public:

    Graph(int size) {

        n = size;

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                adjMatrix[i][j] = 0;

            }

            visited[i] = 0;

        }

    }

    void addEdge(int u, int v) {

        adjMatrix[u][v] = 1;

        adjMatrix[v][u] = 1;

    }

    void bfs(int start) {

        int queue[MAX], front = 0, rear = 0;

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

        queue[rear++] = start;

        visited[start] = 1;

        cout << "BFS Traversal: ";

        while (front < rear) {

            int node = queue[front++];

            cout << node << " ";

            for (int i = 0; i < n; i++) {

                if (adjMatrix[node][i] == 1 && visited[i] == 0) {

                    queue[rear++] = i;

                    visited[i] = 1;

                }

            }

        }

        cout << "\n";

    }

    void dfs(int start) {

        int stack[MAX], top = -1;

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

        stack[++top] = start;

        cout << "DFS Traversal: ";

        while (top >= 0) {

            int node = stack[top--];

            if (!visited[node]) {

                cout << node << " ";

                visited[node] = 1;

            }

            for (int i = n - 1; i >= 0; i--) {

                if (adjMatrix[node][i] == 1 && visited[i] == 0) {

                    stack[++top] = i;

                }

            }

        }

        cout << "\n";

    }

    void executeGraph() {

        int edges, u, v, start;

        cout << "Enter number of edges: ";

        cin >> edges;

        cout << "Enter edges (u v):\n";

        for (int i = 0; i < edges; i++) {

            cin >> u >> v;

            addEdge(u, v);

        }

        cout << "Enter starting vertex for traversal: ";

        cin >> start;

        bfs(start);

        dfs(start);

    }

};

void main() {

    clrscr();

     cout << "Shiv Arora"<< endl;

    int n;

    cout << "Enter number of vertices: ";

    cin >> n;

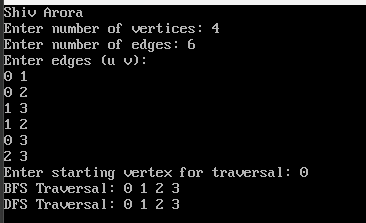
    Graph g(n);

    g.executeGraph();

    getch();

}

OUTPUT:



Q.7) Implement and compare sorting algorithms (bubble sort, Insertion sort, merge sort)

#include <iostream.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 10

#define ITERATIONS 1000

class Array {

public:

    int arr[SIZE];

    void generate() {

        srand(time(NULL));

    for (int i = 0; i < SIZE; i++) {

            arr[i] = rand() % 1000;

        }

    }

    void copy(int source[]) {

    for (int i = 0; i < SIZE; i++) {

            arr[i] = source[i];

        }

    }

    void print() {

    for (int i = 0; i < SIZE; i++) {

            cout << arr[i] << " ";

        }

        cout << endl;

    }

};

class Sorting {

public:

    void bubbleSort(int arr[], int size) {

    for (int i = 0; i < size - 1; i++) {

            for (int j = 0; j < size - i - 1; j++) {

                if (arr[j] > arr[j + 1]) {

                    int temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                }

            }

        }

    }

    void insertionSort(int arr[], int size) {

    for (int i = 1; i < size; i++) {

            int key = arr[i];

            int j = i - 1;

            while (j >= 0 && arr[j] > key) {

                arr[j + 1] = arr[j];

                j--;

            }

            arr[j + 1] = key;

        }

    }

    void merge(int arr[], int left, int mid, int right) {

        int n1 = mid - left + 1, n2 = right - mid;

        int L[SIZE], R[SIZE];

    for (int a = 0; a < n1; a++) L[a] = arr[left + a];

        for (int b = 0; b < n2; b++) R[b] = arr[mid + 1 + b];

        int i = 0, j = 0, k = left;

        while (i < n1 && j < n2) {

            arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

        }

        while (i < n1) arr[k++] = L[i++];

        while (j < n2) arr[k++] = R[j++];

    }

    void mergeSort(int arr[], int left, int right) {

        if (left < right) {

            int mid = left + (right - left) / 2;

            mergeSort(arr, left, mid);

            mergeSort(arr, mid + 1, right);

            merge(arr, left, mid, right);

        }

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora" << endl;

    Array original, temp;

    Sorting sorter;

    clock\_t start, end;

    double timeTaken;

    original.generate();

    cout << "Original Array: ";

    original.print();

    start = clock();

    for (int i = 0; i < ITERATIONS; i++) {

        temp.copy(original.arr);

        sorter.bubbleSort(temp.arr, SIZE);

    }

    end = clock();

    timeTaken = ((double)(end - start)) \* 1000 / CLOCKS\_PER\_SEC / ITERATIONS;

    cout << "\nBubble Sorted Array: ";

    temp.print();

    cout << "Avg Bubble Sort Time: " << timeTaken << " ms" << endl;

    start = clock();

    for (int j = 0; j < ITERATIONS; j++) {

    temp.copy(original.arr);

    sorter.insertionSort(temp.arr, SIZE);

    }

    end = clock();

    timeTaken = ((double)(end - start)) \* 1000 / CLOCKS\_PER\_SEC / ITERATIONS;

    cout << "\nInsertion Sorted Array: ";

    temp.print();

    cout << "Avg Insertion Sort Time: " << timeTaken << " ms" << endl;

    start = clock();

    for (int k = 0; k < ITERATIONS; k++) {

    temp.copy(original.arr);

    sorter.mergeSort(temp.arr, 0, SIZE - 1);

    }

    end = clock();

    timeTaken = ((double)(end - start)) \* 1000 / CLOCKS\_PER\_SEC / ITERATIONS;

    cout << "\nMerge Sorted Array: ";

    temp.print();

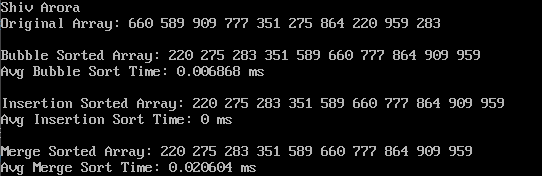
    cout << "Avg Merge Sort Time: " << timeTaken << " ms" << endl;

    getch();

    getch();

}

OUTPUT:



Q.8) Implement Dijkstra’s Algorithm to find the shortest path in a weighted graph.

#include <iostream.h>

#include <conio.h>

#include <limits.h>

#define MAX 10

class Graph {

private:

    int n;

    int adjMatrix[MAX][MAX];

public:

    Graph(int size) {

    n = size;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

        adjMatrix[i][j] = 0;

        }

    }

    }

    void addEdge(int u, int v, int weight) {

        adjMatrix[u][v] = weight;

        adjMatrix[v][u] = weight;

    }

    int minDistance(int dist[], int visited[]) {

        int min = INT\_MAX, minIndex = -1;

        for (int v = 0; v < n; v++) {

            if (visited[v] == 0 && dist[v] <= min) {

                min = dist[v];

                minIndex = v;

            }

        }

        return minIndex;

    }

    void dijkstra(int src) {

        int dist[MAX];

        int visited[MAX] = {0};

        for (int i = 0; i < n; i++) {

            dist[i] = INT\_MAX;

        }

        dist[src] = 0;

        for (int count = 0; count < n - 1; count++) {

            int u = minDistance(dist, visited);

            visited[u] = 1;

            for (int v = 0; v < n; v++) {

                if (!visited[v] && adjMatrix[u][v] && dist[u] != INT\_MAX && dist[u] + adjMatrix[u][v] < dist[v]) {

                    dist[v] = dist[u] + adjMatrix[u][v];

                }

            }

        }

        cout << "\nVertex \t Distance from Source\n";

    for (int j = 0; j < n; j++) {

        cout << j << " \t " << dist[j] << "\n";

        }

    }

    void executeGraph() {

        int edges, u, v, weight, start;

        cout << "Enter number of edges: ";

        cin >> edges;

        cout << "Enter edges (u v weight):\n";

        for (int i = 0; i < edges; i++) {

            cin >> u >> v >> weight;

            addEdge(u, v, weight);

        }

        cout << "Enter starting vertex for Dijkstra's algorithm: ";

        cin >> start;

        dijkstra(start);

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    int n;

    cout << "Enter number of vertices: ";

    cin >> n;

    Graph g(n);

    g.executeGraph();

    getch();

}

OUTPUT:

