Assignment 4 DSA

Q1: Write short note on the following:(i) BST.  
(ii) Time Complexity, Big O notation.  
(iii) Merge Sort.  
(iv) Depth First Traversal.

(i) **Binary Search Tree (BST):** A Binary Search Tree (BST) is a binary tree data structure where each node has at most two children, referred to as the left child and the right child. The key property of a BST is that the value of each node in the left subtree is less than or equal to the node's value, and the value of each node in the right subtree is greater than or equal to the node's value. This ordering property makes it efficient for searching, insertion, and deletion operations.

(ii) **Time Complexity, Big O notation:** Time complexity is a measure that describes the amount of time an algorithm takes to complete as a function of the size of the input data. Big O notation is used to express the upper bound of this time complexity. Big O notation allows programmers and analysts to analyze the efficiency and scalability of algorithms without being concerned with constant factors or lower-order terms.

(iii) **Merge Sort:** Merge Sort is a sorting algorithm that follows the divide-and-conquer paradigm. The basic idea is to divide the unsorted list into n sublists, each containing one element, and then repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining, which is the sorted list. It is a stable sorting algorithm, meaning that the relative order of equal elements remains unchanged.

(iv) **Depth First Traversal:** Depth First Traversal is a method used to traverse or search tree or graph data structures. It starts from the root node and explores as far as possible along each branch before backtracking. There are different variants of depth-first traversal, such as Preorder, Inorder, and Postorder for binary trees. In Preorder traversal, the root is visited first, then the left subtree is traversed, followed by the right subtree. In Inorder traversal, the left subtree is traversed first, followed by the root, and then the right subtree. In Postorder traversal, the left and right subtrees are traversed before the root.

Sort an array using insertion and selection sort.

Code:

#include <iostream>

using namespace std;

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

for (int i = 1; i < n; ++i) {

int key = arr[i];

int j = i - 1;

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

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

j = j - 1;

}

arr[j + 1] = key;

}

}

void selectionSort(int arr[], int n) {

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

int minIndex = i;

for (int j = i + 1; j < n; ++j) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

void printArray(int arr[], int n) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

const int n = 5;

int arr[n] = { 12, 11, 13, 5, 6 };

cout << "Original array: ";

printArray(arr, n);

insertionSort(arr, n);

cout << "Array after Insertion Sort: ";

printArray(arr, n);

int arr2[n] = { 12, 11, 13, 5, 6 };

selectionSort(arr2, n);

cout << "Array after Selection Sort: ";

printArray(arr2, n);

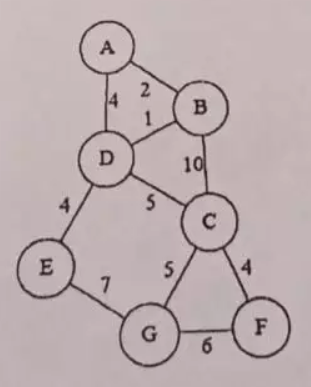
return 0;

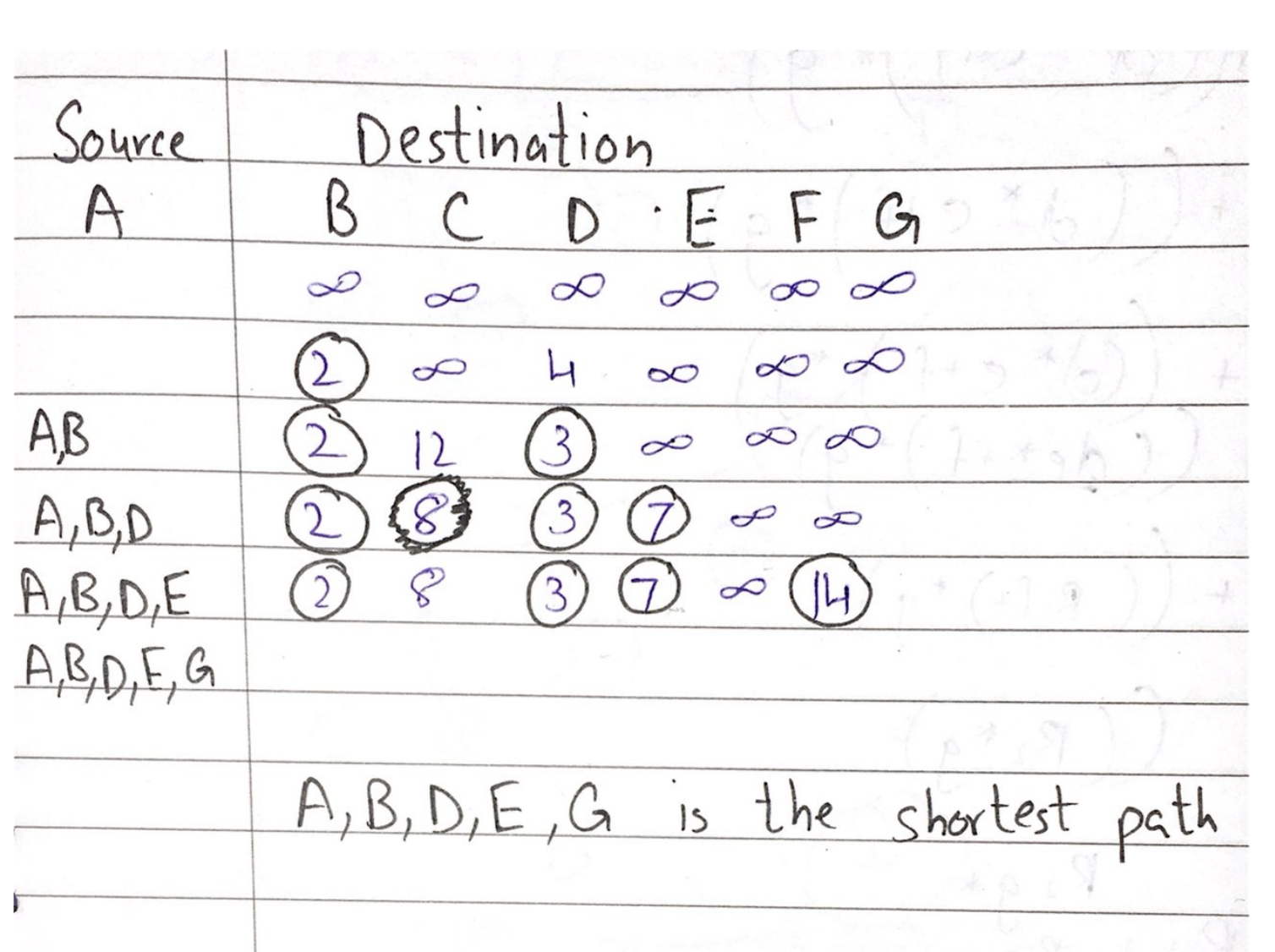
}

A screenshot of a computer

Description automatically generated

Q2: Using Dijkstra algorithm, find the shortest path on this graph between A and G, showing your work:





Q3: What are the real-life applications of graphs?

1. **Network Routing:**
   * **Application:** Internet Routing
   * **Description:** In computer networks, nodes can represent routers or devices, and edges represent communication links. Graphs are used to model and optimize the routing of data packets through the network, ensuring efficient and reliable communication.
2. **File System Structure:**
   * **Application:** File Allocation and Storage
   * **Description:** File systems can be represented as hierarchical graphs, where nodes are directories or folders, and edges connect them based on containment relationships. This graph structure helps in organizing and navigating files, optimizing file access, and managing storage space.
3. **Database Management:**
   * **Application:** Relational Database Systems
   * **Description:** In relational databases, tables and their relationships can be modeled as graphs. Nodes represent entities (tables), and edges represent relationships between entities. Graphs assist in designing and querying databases efficiently, supporting tasks such as data retrieval, integrity enforcement, and normalization.

Q4: Given a weighted, undirected, and connected graph of **V** vertices and **E** edges. The task is to find the sum of weights of the edges of the Minimum Spanning Tree. Given adjacency list adj as input parameters. Here adj[i] contains vectors of size 2, where the first integer in that vector denotes the end of the edge and the second integer denotes the edge weight.

Code:

#include <iostream>

using namespace std;

const int MAX\_VERTICES = 100;

int primMST(int adj[MAX\_VERTICES][MAX\_VERTICES], int V) {

bool visited[MAX\_VERTICES] = { false };

int weights[MAX\_VERTICES];

const int INF = 1e9;

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

weights[i] = INF;

}

int mstSum = 0;

weights[0] = 0;

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

int minVertex = -1;

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

if (!visited[v] && (minVertex == -1 || weights[v] < weights[minVertex])) {

minVertex = v;

}

}

visited[minVertex] = true;

mstSum += weights[minVertex];

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

if (!visited[v] && adj[minVertex][v] != -1 && adj[minVertex][v] < weights[v]) {

weights[v] = adj[minVertex][v];

}

}

}

return mstSum;

}

int main() {

int adj[MAX\_VERTICES][MAX\_VERTICES] = { {5, 8, 3}, {4, 5, 4}, {3, 4, 3} };

int V = 3;

int result = primMST(adj, V);

cout << "Sum of weights of the Minimum Spanning Tree: " << result << endl;

return 0;

}

Output:

