ESA LAB PROGRAMS

Bubble Sort (Ascending Order)

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    // Bubble Sort logic
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "Sorted array:\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
```

```
}
```

Insertion Sort

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    // Insertion Sort
    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--;
        }
        arr[j + 1] = key;
    }

    cout << "Sorted array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
```

```
}
```

Selection Sort

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    // Selection Sort
    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;
            }
        }

        // swap minimum element with first element
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }

    cout << "Sorted array:\n";
```

```
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Brute Force String Search

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string text, pattern;
    cout << "Enter text: ";
    cin >> text;
    cout << "Enter pattern: ";
    cin >> pattern;

    int n = text.length();
    int m = pattern.length();

    bool found = false;
    for (int i = 0; i <= n - m; i++) {
        int j;
        for (j = 0; j < m; j++) {
            if (text[i + j] != pattern[j])
                break;
        }
        if (j == m) {
            cout << "Pattern found at index " << i << endl;
            found = true;
        }
    }
```

```
    if (!found)
        cout << "Pattern not found";
    return 0;
}
```

Floyd's C++

```
#include <iostream>
using namespace std;

#define INF 99999
#define V 4
int main() {
    int dist[V][V] = {
        {0,   5,   INF, 10},
        {INF, 0,   3,   INF},
        {INF, INF, 0,   1},
        {INF, INF, INF, 0}
    };
    // Floyd's Algorithm
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    cout << "Shortest Path Matrix:\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
```

```
                cout << "INF ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Warshall's C++

```
#include <iostream>
using namespace std;

#define V 4
int main() {
    int reach[V][V] = {
        {0, 1, 0, 0},
        {0, 0, 1, 0},
        {0, 0, 0, 1},
        {0, 0, 0, 0}
    };

    // Warshall's Algorithm
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                reach[i][j] = reach[i][j] || (reach[i][k] &&
reach[k][j]);
            }
        }
    }
    cout << "Transitive Closure Matrix:\n";
```

```
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout << reach[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Quick Sort

```
#include <iostream>
using namespace std;

// Quick Sort function
void sort(int arr[], int low, int high) {
    if (low < high) {
        int p = partition(arr, low, high);
        sort(arr, low, p - 1);
        sort(arr, p + 1, high);
    }
}
// Partition function
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;
}
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    sort(arr, 0, n - 1);
    cout << "Sorted array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Merge Sort

```
#include <iostream>
using namespace std;

// Merge Sort function
void sort(int arr[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        sort(arr, low, mid);
        sort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

// Merge function
```

```
void merge(int arr[], int low, int mid, int high) {
    int i = low;
    int j = mid + 1;
    int k = 0;

    int temp[100];   // for exam simplicity

    while (i <= mid && j <= high) {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }

    while (i <= mid)
        temp[k++] = arr[i++];

    while (j <= high)
        temp[k++] = arr[j++];

    for (i = low, k = 0; i <= high; i++, k++)
        arr[i] = temp[k];
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    sort(arr, 0, n - 1);

    cout << "Sorted array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

BST IN SEPERATE PDF

1. **INSERT**
2. **DELETE**
3. **TRAVERSALS**

BFS(Breadth First Search)

```
#include <iostream>
using namespace std;

void bfs(int m[10][10], int v, int source) {
    int queue[20];
    int front = 0, rear = 0, u, i;
    int visited[10];

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

    queue[rear] = source;
    visited[source] = 1;

    cout << "The BFS Traversal is... \n";
```

```
    while (front <= rear) {
        u = queue[front];
        cout << u << "\t";
        front++;

        for (i = 0; i < v; i++) {
            if (m[u][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                rear++;
                queue[rear] = i;
            }
        }
    }
}

int main() {
    int v = 5;
    int m[10][10] = {{0,1,1,0,0}, {1,0,0,1,1},
        {1,0,0,0,1}, {0,1,0,0,0}, {0,1,1,0,0}};

    int source;
    cout << "Enter the source vertex: ";
    cin >> source;

    bfs(m, v, source);
    return 0;
}
```

DFS(Depth First Search)

```
#include <iostream>
using namespace std;

int v = 5;
int m[10][10] = {{0,1,1,0,0}, {1,0,0,1,1},
        {1,0,0,0,1}, {0,1,0,0,0}, {0,1,1,0,0}};
int visited[10];

void dfs(int m[10][10], int v, int source) {
    visited[source] = 1;
    for (int i = 0; i < v; i++) {
        if (m[source][i] == 1 && visited[i] == 0) {
            cout << i << "\t";
            dfs(m, v, i);
        }
    }
}

int main() {
    int source;
    for (int i = 0; i < v; i++)
        visited[i] = 0;

    cout << "Enter the source vertex: ";
    cin >> source;
    cout << "The DFS Traversal is... \n";
    cout << source << "\t";
    dfs(m, v, source);
    return 0;
}
```

Heap Sort

```
#include <iostream>
using namespace std;

// Function to heapify a subtree rooted at index i
void heapify(int arr[], int n, int i) {
    int largest = i;        // root
    int left = 2 * i + 1;   // left child
    int right = 2 * i + 2;  // right child

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

// Heap Sort function
void sort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Extract elements from heap
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);   // move max to end
        heapify(arr, i, 0);
```

```
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    sort(arr, n);

    cout << "Sorted array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Prim's

```
#include <iostream>
#include <climits>
using namespace std;

int minKey(int key[], bool mstSet[], int V) {
    int min = INT_MAX, minIndex;

    for (int v = 0; v < V; v++) {
        if (!mstSet[v] && key[v] < min) {
```

```
            min = key[v];
            minIndex = v;
        }
    }
    return minIndex;
}

void primMST(int graph[10][10], int V) {
    int parent[10];   // stores MST
    int key[10];      // min edge weight
    bool mstSet[10];  // included in MST

    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        mstSet[i] = false;
    }

    key[0] = 0;      // start from vertex 0
    parent[0] = -1; // root

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = true;

        for (int v = 0; v < V; v++) {
           if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
{
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
```

```
    cout << "Edges in Minimum Spanning Tree:\n";
    int totalWeight = 0;
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i
             << " : " << graph[i][parent[i]] << endl;
        totalWeight += graph[i][parent[i]];
    }

    cout << "Total weight of MST = " << totalWeight << endl;
}

int main() {
    int V;
    cout << "Enter number of vertices: ";
    cin >> V;

    int graph[10][10];
    cout << "Enter adjacency matrix (0 if no edge):\n";
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            cin >> graph[i][j];

    primMST(graph, V);

    return 0;
}
```

Expected Input

```
4
0 10 6 5
```

```
10 0 0 15
6 0 0 4
5 15 4 0
```

Kruskal

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Edge {
    int src, dest, weight;
};
int find(int parent[], int i) {
    if (parent[i] == i)
        return i;
    return find(parent, parent[i]);
}

void unionSet(int parent[], int x, int y) {
    int xroot = find(parent, x);
    int yroot = find(parent, y);
    parent[xroot] = yroot;
}
int main() {
    int V, E;
    cout << "Enter number of vertices and edges: ";
    cin >> V >> E;

    Edge edges[E];
    cout << "Enter edges (src dest weight):\n";
    for (int i = 0; i < E; i++) {
```

```
        cin >> edges[i].src >> edges[i].dest >> edges[i].weight;
    }

    // Sort edges by weight
    sort(edges, edges + E, [](Edge a, Edge b) {
        return a.weight < b.weight;
    });

    int parent[V];
    for (int i = 0; i < V; i++)
        parent[i] = i;

    cout << "\nEdges in Minimum Spanning Tree:\n";
    int mstWeight = 0;
    for (int i = 0; i < E; i++) {
        int x = find(parent, edges[i].src);
        int y = find(parent, edges[i].dest);

        if (x != y) {
            cout << edges[i].src << " - "
                 << edges[i].dest << " : "
                 << edges[i].weight << endl;
            mstWeight += edges[i].weight;
            unionSet(parent, x, y);
        }
    }

    cout << "Total weight of MST = " << mstWeight << endl;
    return 0;
}
```

Correct Input

```
4 5

0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
```

Dijkstra's Algorithm

```
#include <iostream>
#include <climits>
using namespace std;

int minDistance(int dist[], bool visited[], int n) {
    int min = INT_MAX, minIndex;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            minIndex = v;
        }
    }
    return minIndex;
}

void dijkstra(int graph[10][10], int n, int src) {
    int dist[10];
    bool visited[10];

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

```
        visited[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = true;

        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] != 0 &&
                dist[u] != INT_MAX &&
                dist[u] + graph[u][v] < dist[v]) {

                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    cout << "Vertex \t Distance from Source\n";
    for (int i = 0; i < n; i++)
        cout << i << "\t\t" << dist[i] << endl;
}

int main() {
    int n;
    cout << "Enter number of vertices: ";
    cin >> n;
    int graph[10][10];
    cout << "Enter adjacency matrix (0 if no edge):\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> graph[i][j];
```

```
    int src;
    cout << "Enter source vertex: ";
    cin >> src;
    dijkstra(graph, n, src);
    return 0;
}
```