// INSERTION SORT

#include <iostream>

using namespace std;

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

    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;

            comparisons++;

        }

        arr[j + 1] = key;

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    insertionSort(arr, n, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

// SELECTION SORT

#include <iostream>

using namespace std;

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

    for (int i = 0; i < n - 1; i++) {

        int min\_index = i;

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

            if (arr[j] < arr[min\_index]) {

                min\_index = j;

            }

            comparisons++;

        }

        swap(arr[min\_index], arr[i]);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    selectionSort(arr, n, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//QUICK SORT

#include <iostream>

using namespace std;

void swap(int &a, int &b) {

    int temp = a;

    a = b;

    b = temp;

}

int partition(int arr[], int low, int high, int &comparisons) {

    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]);

        }

        comparisons++;

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

void quickSort(int arr[], int low, int high, int &comparisons) {

    if (low < high) {

        int pi = partition(arr, low, high, comparisons);

        quickSort(arr, low, pi - 1, comparisons);

        quickSort(arr, pi + 1, high, comparisons);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    quickSort(arr, 0, n - 1, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//BUBBLE SORT

#include <iostream>

using namespace std;

void swap(int &a, int &b) {

    int temp = a;

    a = b;

    b = temp;

}

void bubbleSort(int arr[], int n, int &comparisons) {

    for (int i = 0; i < n - 1; i++) {

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

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

                swap(arr[j], arr[j + 1]);

            }

            comparisons++;

        }

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    bubbleSort(arr, n, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//MERGE SORT

#include <iostream>

using namespace std;

// Function to merge two subarrays of arr[]

void merge(int arr[], int left, int middle, int right, int &comparisons) {

    int n1 = middle - left + 1;

    int n2 = right - middle;

    // Create temporary arrays

    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]

    for (int i = 0; i < n1; i++) {

        L[i] = arr[left + i];

    }

    for (int j = 0; j < n2; j++) {

        R[j] = arr[middle + 1 + j];

    }

    // Merge the temporary arrays back into arr[]

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        comparisons++; // Increment comparison count

        k++;

    }

    // Copy the remaining elements of L[], if there are any

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    // Copy the remaining elements of R[], if there are any

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

// Function to perform merge sort

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

    if (left < right) {

        // Find the middle point

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

        // Sort first and second halves

        mergeSort(arr, left, middle, comparisons);

        mergeSort(arr, middle + 1, right, comparisons);

        // Merge the sorted halves

        merge(arr, left, middle, right, comparisons);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    mergeSort(arr, 0, n - 1, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//HEAP SORT

#include <iostream>

using namespace std;

void heapify(int arr[], int n, int i, int &comparisons) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    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]);

        comparisons++;

        heapify(arr, n, largest, comparisons);

    }

}

void heapSort(int arr[], int n, int &comparisons) {

    for (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i, comparisons);

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]);

        comparisons++;

        heapify(arr, i, 0, comparisons);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    int comparisons = 0;

    heapSort(arr, n, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//COUNT SORT

#include <iostream>

using namespace std;

void countingSort(int arr[], int n, int max\_value, int &comparisons) {

    int count[max\_value + 1] = {0};

    for (int i = 0; i < n; i++) {

        count[arr[i]]++;

        comparisons++; // Increment comparison count

    }

    for (int i = 1; i <= max\_value; i++) {

        count[i] += count[i - 1];

    }

    int output[n];

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

        output[count[arr[i]] - 1] = arr[i];

        count[arr[i]]--;

    }

    for (int i = 0; i < n; i++) {

        arr[i] = output[i];

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    int max\_value = 0;

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

        max\_value = max(max\_value, arr[i]);

    }

    int comparisons = 0;

    countingSort(arr, n, max\_value, comparisons);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << "\nNumber of comparisons made: " << comparisons << endl;

    return 0;

}

//RADIX SORT

#include <iostream>

using namespace std;

int getMax(int arr[], int n) {

    int max = arr[0];

    for (int i = 1; i < n; i++) {

        if (arr[i] > max) {

            max = arr[i];

        }

    }

    return max;

}

void countingSort(int arr[], int n, int exp) {

    int output[n];

    int count[10] = {0};

    for (int i = 0; i < n; i++) {

        count[(arr[i] / exp) % 10]++;

    }

    for (int i = 1; i < 10; i++) {

        count[i] += count[i - 1];

    }

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

        output[count[(arr[i] / exp) % 10] - 1] = arr[i];

        count[(arr[i] / exp) % 10]--;

    }

    for (int i = 0; i < n; i++) {

        arr[i] = output[i];

    }

}

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

    int max = getMax(arr, n);

    for (int exp = 1; max / exp > 0; exp \*= 10) {

        countingSort(arr, n, exp);

    }

}

int main() {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    int arr[n];

    cout << "Enter " << n << " integers: ";

    for (int i = 0; i < n; i++) {

        cin >> arr[i];

    }

    radixSort(arr, n);

    cout << "Sorted array: ";

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

//BFS

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

class Graph

{

public:

    int V;                   /

    vector<vector<int>> adj;

    bool isDirected;

    // Constructor

    Graph(int numVertices, bool directed = true)

    {

        V = numVertices;

        isDirected = directed;

        adj.resize(V);

    }

    // Function to add an edge between vertices u and v

    void addEdge(int u, int v)

    {

        adj[u].push\_back(v);

        if (!isDirected)

        {

            adj[v].push\_back(u);

        }

    }

    void BFS(int start)

    {

        vector<bool> visited(V, false); // Visited array to keep track of visited vertices

        queue<int> q;                   // Queue for BFS traversal

        visited[start] = true; // Mark the start node as visited

        q.push(start);         // Enqueue the start node

        while (!q.empty())

        {

            int current = q.front(); // Dequeue a vertex from the queue

            q.pop();

            cout << current << " "; // Print the dequeued vertex

            // Visit all adjacent vertices of the dequeued vertex

            for (int i = 0; i < adj[current].size(); ++i)

            {

                int adjacent = adj[current][i];

                if (!visited[adjacent])

                {

                    visited[adjacent] = true;

                    q.push(adjacent); // Enqueue the adjacent vertex if not visited

                }

            }

        }

    }

};

int main()

{

    char choice;

    bool isDirected;

    cout << "Is the graph directed? (y/n): ";

    cin >> choice;

    if (choice == 'y' || choice == 'Y')

        isDirected = true;

    else

        isDirected = false;

    int numVertices;

    cout << "Enter the number of vertices: ";

    cin >> numVertices;

    Graph graph(numVertices, isDirected);

    int numEdges;

    cout << "Enter the number of edges: ";

    cin >> numEdges;

    int u, v;

    cout << "Enter the edges (format: u v):" << endl;

    for (int i = 0; i < numEdges; ++i)

    {

        cin >> u >> v;

        graph.addEdge(u, v);

    }

    int startVertex;

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

    cin >> startVertex;

    cout << "BFS Traversal starting from vertex " << startVertex << ": ";

    graph.BFS(startVertex);

    cout << endl;

    return 0;

}

//DFS

#include <iostream>

using namespace std;

#define MAX\_NODES 100 // Define maximum number of nodes

int graph[MAX\_NODES][MAX\_NODES]; // Adjacency matrix representation of the graph

bool visited[MAX\_NODES]; // Array to keep track of visited nodes

void dfs(int node, int n) {

    cout << node << " "; // Print the current node

    visited[node] = true; // Mark the current node as visited

    // Traverse all adjacent nodes of the current node

    for (int i = 0; i < n; ++i) {

        if (graph[node][i] && !visited[i]) {

            dfs(i, n); // Recursive call to visit adjacent nodes

        }

    }

}

int main() {

    int n, e; // Number of nodes and edges

    cout << "Enter the number of nodes and edges: ";

    cin >> n >> e;

    cout << "Enter the edges (node1 node2):" << endl;

    for (int i = 0; i < e; ++i) {

        int node1, node2;

        cin >> node1 >> node2;

        graph[node1][node2] = 1;

        graph[node2][node1] = 1; // For undirected graph

    }

    int startNode;

    cout << "Enter the starting node for DFS: ";

    cin >> startNode;

    cout << "DFS traversal starting from node " << startNode << ": ";

    dfs(startNode, n); // Perform DFS traversal

    cout << endl;

    return 0;

}

//STRASSEN ALGORITHM

#include <iostream>

using namespace std;

void strassenMultiplication(int A[2][2], int B[2][2], int C[2][2]) {

    int M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

    int M2 = (A[1][0] + A[1][1]) \* B[0][0];

    int M3 = A[0][0] \* (B[0][1] - B[1][1]);

    int M4 = A[1][1] \* (B[1][0] - B[0][0]);

    int M5 = (A[0][0] + A[0][1]) \* B[1][1];

    int M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

    int M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

    C[0][0] = M1 + M4 - M5 + M7;

    C[0][1] = M3 + M5;

    C[1][0] = M2 + M4;

    C[1][1] = M1 - M2 + M3 + M6;

}

int main() {

    int A[2][2], B[2][2], C[2][2];

    cout << "Enter the elements of matrix A (2x2):" << endl;

    for (int i = 0; i < 2; i++) {

        for (int j = 0; j < 2; j++) {

            cin >> A[i][j];

        }

    }

    cout << "Enter the elements of matrix B (2x2):" << endl;

    for (int i = 0; i < 2; i++) {

        for (int j = 0; j < 2; j++) {

            cin >> B[i][j];

        }

    }

    strassenMultiplication(A, B, C);

    cout << "Result of multiplication:" << endl;

    for (int i = 0; i < 2; i++) {

        for (int j = 0; j < 2; j++) {

            cout << C[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

//FRIM'S ALGORITHM

#include <bits/stdc++.h>

using namespace std;

#define V 4

int minKey(int key[], bool mstSet[])

{

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

    return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

    cout << "Edge \tWeight\n";

    for (int i = 1; i < V; i++)

        cout << parent[i] << " - " << i << " \t"

            << graph[i][parent[i]] << " \n";

}

void primMST(int graph[V][V])

{

    int parent[V];

    int key[V];

    // key[] holds the key values used to pick the minimum weight edge.

    bool mstSet[V];

    // mstSet[] keeps track of vertices included in the MST

    for (int i = 0; i < V; i++)

        key[i] = INT\_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

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

        int u = minKey(key, mstSet);

        mstSet[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == false

                && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph);

}

int main()

{

    int graph[V][V] = { { 0, 2, 0, 6},

                        { 2, 0, 3, 8},

                        { 0, 3, 0, 0},

                        { 6, 8, 0, 0} };

    primMST(graph);

    return 0;

}

//0-1 KNAPSACK

#include <iostream>

using namespace std;

const int MAX = 100;

int weight[MAX], value[MAX];

int n, W;

int knapsack() {

    int dp[MAX][MAX];

    for (int i = 0; i <= n; ++i) {

        for (int w = 0; w <= W; ++w) {

            if (i == 0 || w == 0)

                dp[i][w] = 0;

            else if (weight[i - 1] <= w)

                dp[i][w] = max(value[i - 1] + dp[i - 1][w - weight[i - 1]], dp[i - 1][w]);

            else

                dp[i][w] = dp[i - 1][w];

        }

    }

    return dp[n][W];

}

int main() {

    cout << "Enter the number of items: ";

    cin >> n;

    cout << "Enter the weight and value of each item:" << endl;

    for (int i = 0; i < n; ++i) {

        cin >> weight[i] >> value[i];

    }

    cout << "Enter the maximum capacity of the knapsack: ";

    cin >> W;

    int maxValue = knapsack();

    cout << "The maximum value that can be obtained: " << maxValue << endl;

    return 0;

}