Q1) Binary Search Implementation:

#include <iostream.h>

#include <conio.h>

class BinarySearch {

    int arr[100], size, comparisons;

public:

    BinarySearch() {

        comparisons = 0;

        size = 0;

    }

    void input() {

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

        cin >> size;

        cout << "Enter " << size << " sorted elements:\n";

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

            cin >> arr[i];

        }

    }

    void iterativeSearch() {

        int key;

        comparisons = 0;

        cout << "\nEnter element to search (Iterative): ";

        cin >> key;

        int low = 0, high = size - 1, mid, index = -1;

        while (low <= high) {

            comparisons++;

            mid = (low + high) / 2;

            if (arr[mid] == key) {

                index = mid;

                break;

            } else if (arr[mid] < key) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        if (index == -1)

            cout << "Element not found.\n";

        else

            cout << "Element found at index: " << index << "\n";

        cout << "Comparisons made: " << comparisons << "\n";

    }

    void recursiveSearchWrapper() {

        int key;

        comparisons = 0;

        cout << "\nEnter element to search (Recursive): ";

        cin >> key;

        int index = recursiveSearch(key, 0, size - 1);

        if (index == -1)

            cout << "Element not found.\n";

        else

            cout << "Element found at index: " << index << "\n";

        cout << "Comparisons made: " << comparisons << "\n";

    }

    int recursiveSearch(int key, int low, int high) {

        if (low > high)

            return -1;

        comparisons++;

        int mid = (low + high) / 2;

        if (arr[mid] == key)

            return mid;

        else if (arr[mid] < key)

            return recursiveSearch(key, mid + 1, high);

        else

            return recursiveSearch(key, low, mid - 1);

    }

    void run() {

        input();

        iterativeSearch();

        recursiveSearchWrapper();

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

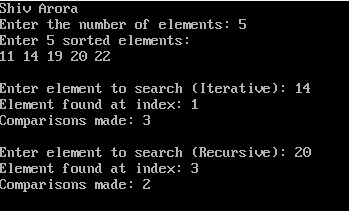
    BinarySearch bs;

    bs.run();

    getch();

}

OUTPUT:



Q2) Quick Sort Implementation:

#include <iostream.h>

#include <conio.h>

class QuickSort {

    int arr[100], n;

    int cmpFirst, swpFirst;

    int cmpMid, swpMid;

    int cmpLast, swpLast;

public:

    void input() {

        cout << "Enter number of elements: ";

        cin >> n;

        cout << "Enter " << n << " elements:\n";

        for (int i = 0; i < n; i++) {

            cin >> arr[i];

        }

    }

    void displayArray(int a[]) {

        for (int i = 0; i < n; i++) {

            cout << a[i] << " ";

        }

        cout << "\n";

    }

    void copyArray(int src[], int dest[]) {

        for (int i = 0; i < n; i++) {

            dest[i] = src[i];

        }

    }

    void quickSortFirst(int a[], int low, int high) {

        if (low < high) {

            int p = partitionFirst(a, low, high);

            quickSortFirst(a, low, p - 1);

            quickSortFirst(a, p + 1, high);

        }

    }

    int partitionFirst(int a[], int low, int high) {

        int pivot = a[low];

        int i = low + 1;

        int j = high;

        while (i <= j) {

            while (i <= high && a[i] <= pivot) {

                cmpFirst++;

                i++;

            }

            while (j >= low && a[j] > pivot) {

                cmpFirst++;

                j--;

            }

            if (i < j) {

                int temp = a[i];

                a[i] = a[j];

                a[j] = temp;

                swpFirst++;

            }

        }

        int temp = a[low];

        a[low] = a[j];

        a[j] = temp;

        swpFirst++;

        return j;

    }

    void quickSortMiddle(int a[], int low, int high) {

        if (low < high) {

            int p = partitionMiddle(a, low, high);

            quickSortMiddle(a, low, p - 1);

            quickSortMiddle(a, p + 1, high);

        }

    }

    int partitionMiddle(int a[], int low, int high) {

        int mid = (low + high) / 2;

        int temp = a[low];

        a[low] = a[mid];

        a[mid] = temp;

        swpMid++;

        int pivot = a[low];

        int i = low + 1;

        int j = high;

        while (i <= j) {

            while (i <= high && a[i] <= pivot) {

                cmpMid++;

                i++;

            }

            while (j >= low && a[j] > pivot) {

                cmpMid++;

                j--;

            }

            if (i < j) {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

                swpMid++;

            }

        }

        temp = a[low];

        a[low] = a[j];

        a[j] = temp;

        swpMid++;

        return j;

    }

    void quickSortLast(int a[], int low, int high) {

        if (low < high) {

            int p = partitionLast(a, low, high);

            quickSortLast(a, low, p - 1);

            quickSortLast(a, p + 1, high);

        }

    }

    int partitionLast(int a[], int low, int high) {

        int temp = a[low];

        a[low] = a[high];

        a[high] = temp;

        swpLast++;

        int pivot = a[low];

        int i = low + 1;

        int j = high;

        while (i <= j) {

            while (i <= high && a[i] <= pivot) {

                cmpLast++;

                i++;

            }

            while (j >= low && a[j] > pivot) {

                cmpLast++;

                j--;

            }

            if (i < j) {

                temp = a[i];

                a[i] = a[j];

                a[j] = temp;

                swpLast++;

            }

        }

        temp = a[low];

        a[low] = a[j];

        a[j] = temp;

        swpLast++;

        return j;

    }

    void sortAndDisplayAll() {

        int tempArr[100];

*// First Element Pivot*

        cmpFirst = swpFirst = 0;

        copyArray(arr, tempArr);

        quickSortFirst(tempArr, 0, n - 1);

        cout << "\nSorted Array using First Element as Pivot:\n";

        displayArray(tempArr);

        cout << "Comparisons: " << cmpFirst << ", Swaps: " << swpFirst << "\n";

*// Middle Element Pivot*

        cmpMid = swpMid = 0;

        copyArray(arr, tempArr);

        quickSortMiddle(tempArr, 0, n - 1);

        cout << "\nSorted Array using Middle Element as Pivot:\n";

        displayArray(tempArr);

        cout << "Comparisons: " << cmpMid << ", Swaps: " << swpMid << "\n";

*// Last Element Pivot*

        cmpLast = swpLast = 0;

        copyArray(arr, tempArr);

        quickSortLast(tempArr, 0, n - 1);

        cout << "\nSorted Array using Last Element as Pivot:\n";

        displayArray(tempArr);

        cout << "Comparisons: " << cmpLast << ", Swaps: " << swpLast << "\n";

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    QuickSort qs;

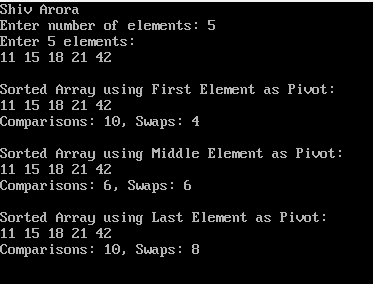
    qs.input();

    qs.sortAndDisplayAll();

    getch();

}

OUTPUT:



Q3) Strassen’s matrix multiplication:

#include <iostream.h>

#include <conio.h>

#define MAX 32 *// Maximum matrix size*

class Strassen {

    int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

    int n;

public:

    void run() {

        cout << "Enter matrix size (2^n, max " << MAX << "): ";

        cin >> n;

        if (n != 1 && n != 2 && n != 4 && n != 8) {

            cout << "Only 2^n sizes (1, 2, 4, 8, 16, 32) allowed.\n";

            return;

        }

        cout << "Enter elements of Matrix A:\n";

        for (int i = 0; i < n; i++)

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

                cin >> A[i][j];

        cout << "Enter elements of Matrix B:\n";

    for (int s = 0; s < n; s++)

        for (int t = 0; t < n; t++)

        cin >> B[s][t];

        strassenMultiply(A, B, C, n);

        cout << "\nResultant Matrix C = A \* B:\n";

        display(C, n);

        getch();

    }

    void display(int M[MAX][MAX], int size) {

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

            for (int j = 0; j < size; j++)

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

            cout << "\n";

        }

    }

    void add(int A[MAX][MAX], int B[MAX][MAX], int result[MAX][MAX], int size) {

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

            for (int j = 0; j < size; j++)

                result[i][j] = A[i][j] + B[i][j];

    }

    void subtract(int A[MAX][MAX], int B[MAX][MAX], int result[MAX][MAX], int size) {

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

            for (int j = 0; j < size; j++)

                result[i][j] = A[i][j] - B[i][j];

    }

    void strassenMultiply(int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX], int size) {

        if (size == 1) {

            C[0][0] = A[0][0] \* B[0][0];

            return;

        }

        int newSize = size / 2;

        int A11[MAX][MAX], A12[MAX][MAX], A21[MAX][MAX], A22[MAX][MAX];

        int B11[MAX][MAX], B12[MAX][MAX], B21[MAX][MAX], B22[MAX][MAX];

        int C11[MAX][MAX], C12[MAX][MAX], C21[MAX][MAX], C22[MAX][MAX];

        int M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX], M4[MAX][MAX];

        int M5[MAX][MAX], M6[MAX][MAX], M7[MAX][MAX];

        int T1[MAX][MAX], T2[MAX][MAX];

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

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

                A11[i][j] = A[i][j];

                A12[i][j] = A[i][j + newSize];

                A21[i][j] = A[i + newSize][j];

                A22[i][j] = A[i + newSize][j + newSize];

                B11[i][j] = B[i][j];

                B12[i][j] = B[i][j + newSize];

                B21[i][j] = B[i + newSize][j];

                B22[i][j] = B[i + newSize][j + newSize];

            }

        }

        add(A11, A22, T1, newSize);

        add(B11, B22, T2, newSize);

        strassenMultiply(T1, T2, M1, newSize);

        add(A21, A22, T1, newSize);

        strassenMultiply(T1, B11, M2, newSize);

        subtract(B12, B22, T1, newSize);

        strassenMultiply(A11, T1, M3, newSize);

        subtract(B21, B11, T1, newSize);

        strassenMultiply(A22, T1, M4, newSize);

        add(A11, A12, T1, newSize);

        strassenMultiply(T1, B22, M5, newSize);

        subtract(A21, A11, T1, newSize);

        add(B11, B12, T2, newSize);

        strassenMultiply(T1, T2, M6, newSize);

        subtract(A12, A22, T1, newSize);

        add(B21, B22, T2, newSize);

        strassenMultiply(T1, T2, M7, newSize);

        add(M1, M4, T1, newSize);

        subtract(T1, M5, T2, newSize);

        add(T2, M7, C11, newSize);

        add(M3, M5, C12, newSize);

        add(M2, M4, C21, newSize);

        subtract(M1, M2, T1, newSize);

        add(T1, M3, T2, newSize);

        add(T2, M6, C22, newSize);

    for (int q = 0; q < newSize; q++) {

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

        C[q][j] = C11[q][j];

        C[q][j + newSize] = C12[q][j];

        C[q + newSize][j] = C21[q][j];

        C[q + newSize][j + newSize] = C22[q][j];

        }

        }

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

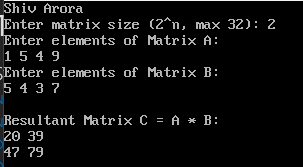
    Strassen s;

    s.run();

    getch();

}

OUTPUT:



Q4) Greedy Method Algorithm:

#include <iostream.h>

#include <conio.h>

#define MAX 20

#define INF 9999

class MST {

private:

    int V, E;

    int adj[MAX][MAX]; *// For Prim*

    int edges[MAX][3]; *// For Kruskal: u, v, weight*

public:

    MST() {

        for (int i = 0; i < MAX; i++)

            for (int j = 0; j < MAX; j++)

                adj[i][j] = INF;

    }

    void readGraph() {

        cout << "Enter number of vertices and edges: ";

        cin >> V >> E;

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

        for (int i = 0; i < E; i++) {

            int u, v, w;

            cin >> u >> v >> w;

            edges[i][0] = u;

            edges[i][1] = v;

            edges[i][2] = w;

            adj[u][v] = w;

            adj[v][u] = w;

        }

    }

    void kruskalMST() {

        int parent[MAX];

        int i, j, u, v;

        int count = 0, total = 0;

        for (i = 0; i < V; i++)

            parent[i] = i;

*// Simple Bubble Sort by weight*

        for (i = 0; i < E - 1; i++) {

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

                if (edges[j][2] > edges[j + 1][2]) {

                    int temp0 = edges[j][0];

                    int temp1 = edges[j][1];

                    int temp2 = edges[j][2];

                    edges[j][0] = edges[j + 1][0];

                    edges[j][1] = edges[j + 1][1];

                    edges[j][2] = edges[j + 1][2];

                    edges[j + 1][0] = temp0;

                    edges[j + 1][1] = temp1;

                    edges[j + 1][2] = temp2;

                }

            }

        }

        cout << "\nKruskal's MST:\n";

        for (i = 0; i < E && count < V - 1; i++) {

            u = find(parent, edges[i][0]);

            v = find(parent, edges[i][1]);

            if (u != v) {

                cout << edges[i][0] << " - " << edges[i][1] << " : " << edges[i][2] << "\n";

                total += edges[i][2];

                unionSet(parent, u, v);

                count++;

            }

        }

        cout << "Total weight: " << total << "\n";

    }

    void primMST() {

        int visited[MAX] = {0};

        int min, u = 0, v = 0, total = 0;

        int count = 0;

        visited[0] = 1;

        cout << "\nPrim's MST:\n";

        while (count < V - 1) {

            min = INF;

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

                if (visited[i]) {

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

                        if (!visited[j] && adj[i][j] < min) {

                            min = adj[i][j];

                            u = i;

                            v = j;

                        }

                    }

                }

            }

            cout << u << " - " << v << " : " << adj[u][v] << "\n";

            visited[v] = 1;

            total += adj[u][v];

            count++;

        }

        cout << "Total weight: " << total << "\n";

    }

    int find(int parent[], int i) {

        while (parent[i] != i)

            i = parent[i];

        return i;

    }

    void unionSet(int parent[], int u, int v) {

        parent[u] = v;

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    MST mst;

    mst.readGraph();

    int choice;

    cout << "\nChoose MST Algorithm:\n1. Kruskal\n2. Prim\nEnter choice: ";

    cin >> choice;

    if (choice == 1)

        mst.kruskalMST();

    else if (choice == 2)

        mst.primMST();

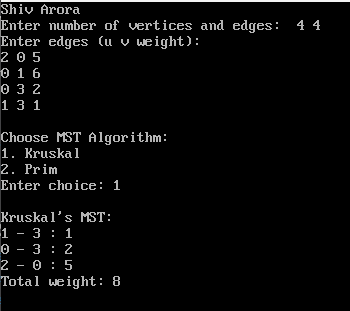
    else

        cout << "Invalid choice";

    getch();

}

OUTPUT:



Q5) Knapsack Problem:

#include <iostream.h>

#include <conio.h>

#define MAX 100

class Knapsack {

    float weights[MAX], values[MAX], ratio[MAX];

    int n;

    float capacity;

public:

    void run() {

        cout << "Enter number of items (max " << MAX << "): ";

        cin >> n;

        cout << "Enter weights of items:\n";

        for (int i = 0; i < n; i++)

            cin >> weights[i];

    cout << "Enter values of items:\n";

    for (int j = 0; j < n; j++)

        cin >> values[j];

    cout << "Enter capacity of knapsack: ";

    cin >> capacity;

    for (int k = 0; k < n; k++)

        ratio[k] = values[k] / weights[k];

        sortItems();

        float maxValue = fillKnapsack();

        cout << "\nMaximum value in knapsack = " << maxValue;

    }

    void sortItems() {

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

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

                if (ratio[i] < ratio[j]) {

                    float temp;

                    temp = ratio[i];

                    ratio[i] = ratio[j];

                    ratio[j] = temp;

                    temp = weights[i];

                    weights[i] = weights[j];

                    weights[j] = temp;

                    temp = values[i];

                    values[i] = values[j];

                    values[j] = temp;

                }

            }

        }

    }

    float fillKnapsack() {

        float totalValue = 0.0;

        float currWeight = 0.0;

        for (int i = 0; i < n; i++) {

            if (currWeight + weights[i] <= capacity) {

                currWeight += weights[i];

                totalValue += values[i];

            } else {

                float remain = capacity - currWeight;

                totalValue += ratio[i] \* remain;

                break;

            }

        }

        return totalValue;

    }

};

void main() {

    clrscr();

    cout << "Shiv Arora"<< endl;

    Knapsack k;

    k.run();

    getch();

}

OUTPUT:

