

DAA – PRACTICALS

DHIRENDRA KUMAR PATEL

ROLL – 16027

bsc hONS cOMPUTER sCIENCE

Practicals List

# 1) Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

#include <iostream>

using namespace std;

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

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

        comparisons = 0; // reset comparisons for each iteration

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

            arr[j + 1] = arr[j];

            j = j - 1;

            comparisons++; // increment comparisons

        }

        arr[j + 1] = key;

    }

}

int main() {

    int arr[] = {10, 20, 7, 15, 5};

    int n = sizeof(arr) / sizeof(arr[0]);

    int comparisons = 0;

    insertionSort(arr, n, comparisons);

    for (int i = 0; i < n; i++)

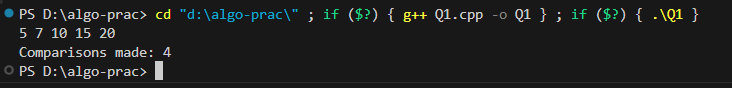
        cout << arr[i] << " ";

    cout << "\nComparisons made: " << comparisons << endl;

    return 0;

}

Output:



# 2) Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).

#include <iostream>

using namespace std;

void merge(int arr[], int l, int m, int r, int& comparisons) {

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2) {

        comparisons++; // increment comparisons

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(int arr[], int l, int r, int& comparisons) {

    if (l >= r) {

        return;

    }

    int m = l + (r - l) / 2;

    mergeSort(arr, l, m, comparisons);

    mergeSort(arr, m + 1, r, comparisons);

    merge(arr, l, m, r, comparisons);

}

int main() {

    int arr[] = {11, 30, 70, 15, 25};

    int n = sizeof(arr) / sizeof(arr[0]);

    int comparisons = 0;

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

    for (int i = 0; i < n; i++)

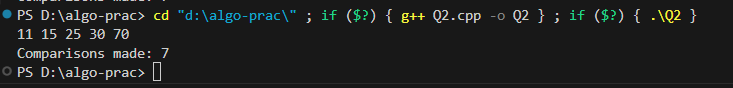
        cout << arr[i] << " ";

    cout << "\nComparisons made: " << comparisons << endl;

    return 0;

}

Output:



# 3) Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).

#include <iostream>

using namespace std;

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

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest]) {

        largest = l;

        comparisons++; // increment comparisons

    }

    if (r < n && arr[r] > arr[largest]) {

        largest = r;

        comparisons++; // increment comparisons

    }

    if (largest!= i) {

        swap(arr[i], arr[largest]);

        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]);

        heapify(arr, i, 0, comparisons);

    }

}

int main() {

    int arr[] = {33, 6, 1, 90, 4, 7};

    int n = sizeof(arr) / sizeof(arr[0]);

    int comparisons = 0;

    heapSort(arr, n, comparisons);

    for (int i = 0; i < n; i++)

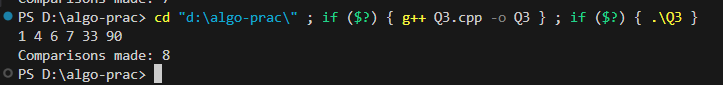
        cout << arr[i] << " ";

    cout << "\nComparisons made: " << comparisons << endl;

    return 0;

}

Output:



# 4) Write a program to sort the elements of an array using Quick Sort (The program should report the number of comparisons).

#include <iostream>

using namespace std;

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

    int pivot = arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {

        comparisons++; // increment comparisons

        if (arr[j] < pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    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 arr[] = {55, 34, 12, 32, 89, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    int comparisons = 0;

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

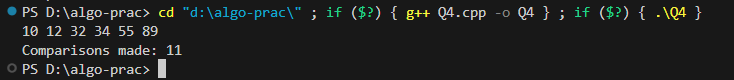
    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << "\nComparisons made: " << comparisons << endl;

    return 0;

}

Output: 

# 5) Write a program to multiply two matrices using the Strassen’s algorithm for matrix multiplication.

#include <iostream>

using namespace std;

const int N = 2;     // Matrix size

// N must be a power of 2 (N = 2^k)

// If N is not a power of 2, we add additional rows and columns as needed

// This program automatically does not add extra rows and columns

template<typename T>

void Strassen(int n, T A[][N], T B[][N], T C[][N]);

template<typename T>

void input(int n, T p[][N]);

template<typename T>

void output(int n, T C[][N]);

int main() {

    int A[N][N],B[N][N],C[N][N];

    cout<<"Enter A\n";

    input(N,A);

    cout<<"\nEnter B\n";

    input(N,B);

    cout<<"\nMatrix A is:\n";

    output(N, A);

    cout<<"\nMatrix B is:\n";

    output(N, B);

    //C=A\*B

    Strassen(N, A, B, C);

    output(N, C);

    return 0;

}

// Input

template<typename T>

void input(int n, T p[][N]) {

     for(int i=0; i<n; i++) {

        cout<<"Please Input Line "<<i+1<<endl;

        for(int j=0; j<n; j++) {

           cin>>p[i][j];

        }

     }

}

// Output

template<typename T>

void output(int n, T C[][N]) {

    cout<<"\n";

     for(int i=0; i<n; i++) {

        for(int j=0; j<n; j++) {

           cout<<C[i][j]<<"\t";

        }

        cout<<"\n";

     }

}

// Multiplication

template<typename T>

void Matrix\_Multiply(T A[][N], T B[][N], T C[][N]) {  // Calculating A\*B -> C

     //output(2,A);

     //output(2,B);

     for(int i=0; i<2; i++) {

        for(int j=0; j<2; j++) {

           C[i][j] = 0;

           for(int t=0; t<2; t++) {

              C[i][j] = C[i][j] + A[i][t]\*B[t][j];

           }

        }

     }

}

// Addition

template <typename T>

void Matrix\_Add(int n, T X[][N], T Y[][N], T Z[][N]) {

     //output(n,X);

     //output(n,Y);

     for(int i=0; i<n; i++) {

        for(int j=0; j<n; j++) {

           Z[i][j] = X[i][j] + Y[i][j];

        }

     }

}

// Under construction

template <typename T>

void Matrix\_Sub(int n, T X[][N], T Y[][N], T Z[][N]) {

     for(int i=0; i<n; i++) {

        for(int j=0; j<n; j++) {

           Z[i][j] = X[i][j] - Y[i][j];

        }

     }

}

//              STRASSEN

template <typename T>

void Strassen(int n, T A[][N], T B[][N], T C[][N]) {

     T A11[N][N], A12[N][N], A21[N][N], A22[N][N];

     T B11[N][N], B12[N][N], B21[N][N], B22[N][N];

     T C11[N][N], C12[N][N], C21[N][N], C22[N][N];

     T M1[N][N], M2[N][N], M3[N][N], M4[N][N], M5[N][N], M6[N][N], M7[N][N];

     T AA[N][N], BB[N][N];

     if(n == 2) {  //2-order

        Matrix\_Multiply(A, B, C);

     } else {

        for(int i=0; i<n/2; i++) {

           for(int j=0; j<n/2; j++) {

              A11[i][j] = A[i][j];

              A12[i][j] = A[i][j+n/2];

              A21[i][j] = A[i+n/2][j];

              A22[i][j] = A[i+n/2][j+n/2];

              B11[i][j] = B[i][j];

              B12[i][j] = B[i][j+n/2];

              B21[i][j] = B[i+n/2][j];

              B22[i][j] = B[i+n/2][j+n/2];

           }

        }

        cout<<"\n Dividing A and B :\n";

        cout<<"\n\tA11:\n";

        output(n/2, A11);

        cout<<"\n\tA12:\n";

        output(n/2, A12);

        cout<<"\n\tA21:\n";

        output(n/2, A21);

        cout<<"\n\tA22:\n";

        output(n/2, A22);

        cout<<"\n\tB11:\n";

        output(n/2, B11);

        cout<<"\n\tB12:\n";

        output(n/2, B12);

        cout<<"\n\tB21:\n";

        output(n/2, B21);

        cout<<"\n\tB22:\n";

        output(n/2, B22);

        cout<<"\n\n\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n---------------------------------------";

        // Calculation of M1 = (A0 + A3) \* (B0 + B3)

        cout<<"\n-------------------------------------";

        cout<<"\nM1 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A11);

        cout<<"++++++++";

        output(n/2, A22);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B11);

        cout<<"++++++++";

        output(n/2, B22);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, A11, A22, AA);

        Matrix\_Add(n/2, B11, B22, BB);

        Strassen(n/2, AA, BB, M1);

        // Calculation of M2 = (A2 + A3) \* B0

        cout<<"------------------------------------";

        cout<<"\nM2 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A21);

        cout<<"++++++++";

        output(n/2, A22);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B11);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, A21, A22, AA);

        Strassen(n/2, AA, B11, M2);

        // Calculation M3 = A0 \* (B1 - B3)

        cout<<"------------------------------------";

        cout<<"\nM3 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A11);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B12);

        cout<<"--------";

        output(n/2, B22);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Sub(n/2, B12, B22, BB);

        Strassen(n/2, A11, BB, M3);

        // Calculation M4 = A3 \* (B2 - B0)

        cout<<"------------------------------------";

        cout<<"\nM4 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A22);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B21);

        cout<<"--------";

        output(n/2, B11);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Sub(n/2, B21, B11, BB);

        Strassen(n/2, A22, BB, M4);

        // Calculation M5 = (A0 + A1) \* B3

        cout<<"------------------------------------";

        cout<<"\nM5 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A11);

        cout<<"++++++++";

        output(n/2, A12);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B22);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, A11, A12, AA);

        Strassen(n/2, AA, B22, M5);

        // Calculation M6 = (A2 - A0) \* (B0 + B1)

        cout<<"------------------------------------";

        cout<<"\nM6 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A21);

        cout<<"--------";

        output(n/2, A11);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B11);

        cout<<"++++++++";

        output(n/2, B12);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Sub(n/2, A21, A11, AA);

        Matrix\_Add(n/2, B11, B12, BB);

        Strassen(n/2, AA, BB, M6);

        // Calculation M7 = (A1 - A3) \* (B2 + B3)

        cout<<"------------------------------------";

        cout<<"\nM7 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, A12);

        cout<<"--------";

        output(n/2, A22);

        cout<<"\_\_\_\_\_\_\_\_\_\n\*\*\*\*\*\*\*\*\n\_\_\_\_\_\_\_\_\_";

        output(n/2, B21);

        cout<<"++++++++";

        output(n/2, B22);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Sub(n/2, A12, A22, AA);

        Matrix\_Add(n/2, B21, B22, BB);

        Strassen(n/2, AA, BB, M7);

        cout<<"\n\n\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n\_\_\_\_\_\_\_\_\_\nM1=";

        output(n/2, M1);

        cout<<"\_\_\_\_\_\_\_\_\_\nM2=";

        output(n/2, M2);

        cout<<"\_\_\_\_\_\_\_\_\_\nM3=";

        output(n/2, M3);

        cout<<"\_\_\_\_\_\_\_\_\_\nM4=";

        output(n/2, M4);

        cout<<"\_\_\_\_\_\_\_\_\_\nM5=";

        output(n/2, M5);

        cout<<"\_\_\_\_\_\_\_\_\_\nM6=";

        output(n/2, M6);

        cout<<"\_\_\_\_\_\_\_\_\_\nM7=";

        output(n/2, M7);

        cout<<"\n\n\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n---------------------------------------";

        // Calculation C0 = M1 + M4 - M5 + M7

        cout<<"\n------------------------------------";

        cout<<"\nC11 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, M1);

        cout<<"+++++++";

        output(n/2, M4);

        cout<<"-------";

        output(n/2, M5);

        cout<<"++++++++";

        output(n/2, M7);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, M1, M4, AA);

        Matrix\_Sub(n/2, M7, M5, BB);

        Matrix\_Add(n/2, AA, BB, C11);

        // Calculation C1 = M3 + M5

        cout<<"------------------------------------";

        cout<<"\nC12 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, M3);

        cout<<"+++++++";

        output(n/2, M5);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, M3, M5, C12);

        // Calculation C2 = M2 + M4

        cout<<"------------------------------------";

        cout<<"\nC21 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, M2);

        cout<<"+++++++";

        output(n/2, M4);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Add(n/2, M2, M4, C21);

        // Calculation C3 = M1 - M2 + M3 + M6

        cout<<"------------------------------------";

        cout<<"\nC22 =\n\_\_\_\_\_\_\_\_\_";

        output(n/2, M1);

        cout<<"-------";

        output(n/2, M2);

        cout<<"+++++++";

        output(n/2, M3);

        cout<<"++++++++";

        output(n/2, M6);

        cout<<"\_\_\_\_\_\_\_\_\_\n------------------------------------";

        Matrix\_Sub(n/2, M1, M2, AA);

        Matrix\_Add(n/2, M3, M6, BB);

        Matrix\_Add(n/2, AA, BB, C22);

        cout<<"\n\n\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n---------------------------------------";

        cout<<"\n\_\_\_\_\_\_\_\_\_\nC11=";

        output(n/2, C11);

        cout<<"\_\_\_\_\_\_\_\_\_\nC12=";

        output(n/2, C12);

        cout<<"\_\_\_\_\_\_\_\_\_\nC21=";

        output(n/2, C21);

        cout<<"\_\_\_\_\_\_\_\_\_\nC22=";

        output(n/2, C22);

        cout<<"\_\_\_\_\_\_\_\_\_";

        cout<<"\n\n\n---------------------------------------";

        cout<<"---------------------------------------";

        cout<<"\nFinal Answer:\n\n";

        for(int i=0; i<n/2; i++) {

           for(int j=0; j<n/2; j++) {

              C[i][j] = C11[i][j];

              C[i][j+n/2] = C12[i][j];

              C[i+n/2][j] = C21[i][j];

              C[i+n/2][j+n/2] = C22[i][j];

           }

        }

     }

}

Input:

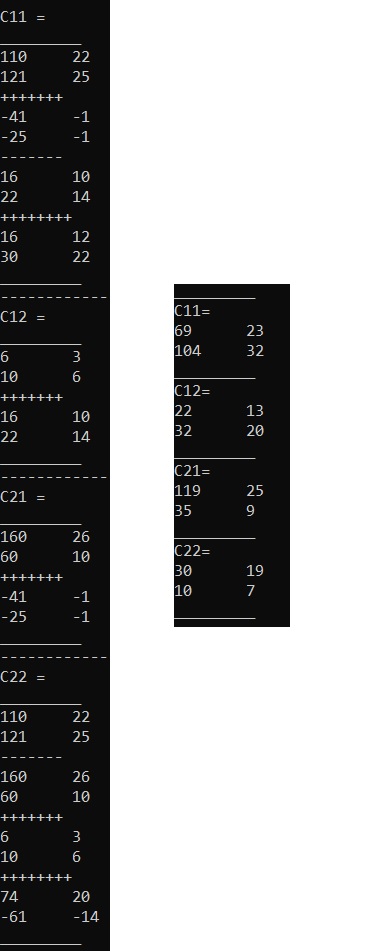
A screenshot of a computer screen

Description automatically generated Output:

A black screen with white text

Description automatically generated A black and white photo of a movie strip

Description automatically generated with medium confidence

 A screenshot of a computer

Description automatically generated

# 6) Write a program to sort the elements of an array using Count Sort.

#include <iostream>

using namespace std;

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

    int k = arr[0];

    for (int i = 0; i < n; i++)

        k = max(k, arr[i]);

    int count[k + 1] = {0};

    for (int i = 0; i < n; i++)

        count[arr[i]]++;

    for (int i = 1; i <= k; 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 arr[] = {35, 40, 11, 20, 78, 43, 51};

    int n = sizeof(arr) / sizeof(arr[0]);

    countSort(arr, n);

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    return 0;

}

Output: 

# 7) Display the data stored in a given graph using the Breadth-First Search algorithm.

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

class Graph {

public:

    vector<vector<int>> adj;

    Graph(int n) {

        adj.resize(n);

    }

    void addEdge(int u, int v) {

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    void BFS(int src) {

        vector<bool> visited(adj.size(), false);

        queue<int> q;

        q.push(src);

        visited[src] = true;

        while (!q.empty()) {

            int u = q.front();

            q.pop();

            cout << u << " ";

            for (auto v : adj[u]) {

                if (!visited[v]) {

                    q.push(v);

                    visited[v] = true;

                }

            }

        }

    }

};

int main() {

    Graph abc(13);

    abc.addEdge(0, 12);

    abc.addEdge(0, 2);

    abc.addEdge(1, 9);

    abc.addEdge(2, 6);

    abc.addEdge(2, 4);

    abc.addEdge(3, 10);

    abc.addEdge(4, 5);

    abc.BFS(0);

    return 0;

}

Output:



# 8) Display the data stored in a given graph using the Depth-First Search algorithm.

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

class Graph {

public:

    vector<vector<int>> adj;

    Graph(int n) {

        adj.resize(n);

    }

    void addEdge(int u, int v) {

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    void DFS(int src, vector<bool>& visited) {

        stack<int> s;

        s.push(src);

        visited[src] = true;

        while (!s.empty()) {

            int u = s.top();

            s.pop();

            cout << u << " ";

            for (auto v : adj[u]) {

                if (!visited[v]) {

                    s.push(v);

                    visited[v] = true;

                }

            }

        }

    }

};

int main() {

    Graph abc(13);

    abc.addEdge(0, 12);

    abc.addEdge(0, 2);

    abc.addEdge(1, 9);

    abc.addEdge(2, 6);

    abc.addEdge(2, 4);

    abc.addEdge(3, 10);

    abc.addEdge(4, 5);

    vector<bool> visited(abc.adj.size(), false);

    abc.DFS(0, visited);

    return 0;

}

Output:



# 9) Write a program to determine a minimum spanning tree of a graph using the Prim’s algorithm.

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

class Graph {

public:

    vector<vector<pair<int, int>>> adj;

    Graph(int n) {

        adj.resize(n);

    }

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

        adj[u].push\_back(make\_pair(v, w));

        adj[v].push\_back(make\_pair(u, w));

    }

    int prim() {

        int n = adj.size();

        vector<bool> visited(n, false);

        vector<int> key(n, 999999);  // Manual maximum value

        vector<int> parent(n, -1);

        priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

        key[0] = 0;

        pq.push(make\_pair(0, 0));

        int mstCost = 0;

        while (!pq.empty()) {

            int u = pq.top().second;

            pq.pop();

            if (visited[u])

                continue;

            visited[u] = true;

            mstCost += key[u];

            for (auto v : adj[u]) {

                if (!visited[v.first] && v.second < key[v.first]) {

                    key[v.first] = v.second;

                    parent[v.first] = u;

                    pq.push(make\_pair(v.second, v.first));

                }

            }

        }

        return mstCost;

    }

};

int main() {

    Graph abc(13);

    abc.addEdge(0, 1, 10);

    abc.addEdge(0, 2, 6);

    abc.addEdge(0, 3, 5);

    abc.addEdge(1, 3, 15);

    abc.addEdge(2, 3, 4);

    abc.addEdge(2, 4, 2);

    abc.addEdge(3, 4, 1);

    cout << "Minimum Spanning Tree Cost: " << abc.prim() << endl;

    return 0;

}

Output:



# 10) Write a program to solve the 0-1 knapsack problem

#include <iostream>

#include <vector>

using namespace std;

int knapSack(int W, vector<int> wt, vector<int> val, int n) {

    vector<vector<int>> K(n + 1, vector<int>(W + 1));

    for (int i = 0; i <= n; i++) {

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

            if (i == 0 || w == 0)

                K[i][w] = 0;

            else if (wt[i - 1] <= w)

                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);

            else

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

        }

    }

    return K[n][W];

}

int main() {

    vector<int> wt = {1, 2, 3, 12, 22, 30};

    vector<int> val = {40, 30, 60, 80, 90, 99};

    int W = 20;

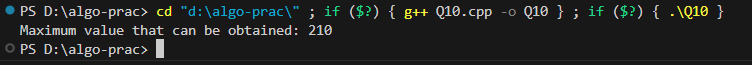
    int n = wt.size();

    cout << "Maximum value that can be obtained: " << knapSack(W, wt, val, n) << endl;

    return 0;

}

Output:



THANK YOU!

BY:

DHIRENDRA KUMAR PATEL

ROLL – 16027

bsc hONS cOMPUTER sCIENCE