1)Removal of Recursion

a)Write a program to implement removal of recursion for Finding maximum from array.

b)Binomial Coefficient B(n, m)= B(n-1, m-1)+B(n-1,m), B(n ,n)=B(n,0)=1

c) Searching element from array

****

a)Write a program to implement removal of recursion for Finding maximum from array.

```
#include <iostream>
using namespace std;
int findMaxIterative(int arr[], int n) {
    int maxVal = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxVal)
            maxVal = arr[i];
    }
    return maxVal;
}
int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) cin >> arr[i];

    cout << "Maximum element = " << findMaxIterative(arr, n) << endl;
    return 0;
}
```

b)Binomial Coefficient B(n, m)= B(n-1, m-1)+B(n-1,m), B(n ,n)=B(n,0)=1

```
**********************************************************************************
#include <iostream>
using namespace std;
int binomialCoeff(int n, int k) {
    int C[n+1][k+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, k); j++) {
            if (j == 0 || j == i) C[i][j] = 1;
            else C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }
    return C[n][k];
}

int main() {
    int n, k;
    cout << "Enter total number of items : ";
    cin >> n ;
    cout<<"Enter number of items we want to choose ";
    cin>> k;
    cout << "Binomial Coefficient = " << binomialCoeff(n, k) << endl;
    return 0;
}
**********************************************************************************
```

c) Searching element from array

```
#include <iostream>
using namespace std;
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++)
        if (arr[i] == key) return i;
    return -1;
}

int main() {
    int n, key;
    cout << "Enter size of array: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) cin >> arr[i];
    cout << "Enter element to search: ";
    cin >> key;

    int pos = linearSearch(arr, n, key);
    if (pos != -1) cout << "Element found at index " << pos << endl;
    else cout << "Element not found!" << endl;
    return 0;
}
```
****

2)Elementary Data Structure–Tree

a)Write a program for creating Max/Min. heap using INSERT.

b) Write a program for creating Max/Min. heap using ADJUST/HEAPIFY.

c) Write a program for sorting given array in ascending/descending order with n=1000, 2000, 3000.Find exact time of execution using Heap Sort.

d) Write a program to implement Weighted UNION and Collapsing FIND operations

****

a)Write a program for creating Max/Min. heap using INSERT.

```
#include <iostream>

using namespace std;


int heap[100]; // array for heap

int size = 0;  // current heap size


void insert(int val) {

    heap[size] = val;   // place at end

    int i = size;

    size++;


    while (i > 0 && heap[(i - 1) / 2] < heap[i]) {

        swap(heap[i], heap[(i - 1) / 2]);

        i = (i - 1) / 2;

    }

}

void printHeap() {

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

        cout << heap[i] << " ";

    cout << endl;

}
```

```
int main() {
    int arr[] = {30, 10, 20, 50, 40, 60};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Inserting elements: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
        insert(arr[i]);
    }
    cout << endl;

    cout << "Max Heap: ";
    printHeap();

    return 0;
}
```

****

b) Write a program for creating Max/Min. heap using ADJUST/HEAPIFY.

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

void minHeapify(int arr[], int n, int i) {
    int smallest = i;          // root
    int left = 2 * i + 1;      // left child
    int right = 2 * i + 2;     // right child

    if (left < n && arr[left] < arr[smallest])
        smallest = left;
        if (right < n && arr[right] < arr[smallest])
        smallest = right;

     if (smallest != i) {
        swap(arr[i], arr[smallest]);
        minHeapify(arr, n, smallest);
    }
}

void buildMinHeap(int arr[], int n) {
        for (int i = n / 2 - 1; i >= 0; i--)
        minHeapify(arr, n, i);
}

void printHeap(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {30, 10, 20, 50, 40, 60};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printHeap(arr, n);

    buildMinHeap(arr, n);

    cout << "Min Heap: ";
    printHeap(arr, n);

    return 0;
}
```
****

c) Write a program for sorting given array in ascending/descending order with n=1000, 2000, 3000.Find exact time of execution using Heap Sort.

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

void mySwap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

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

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

```
}

void heapSort(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n-1; i > 0; i--) {
        mySwap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

void testHeapSort(int n) {
    int *arr = new int[n];
    for (int i = 0; i < n; i++) arr[i] = rand() % 10000;

    clock_t start = clock();
    heapSort(arr, n);
    clock_t end = clock();

    cout << "n=" << n << " Ascending time: "
         << (double)(end - start) / CLOCKS_PER_SEC << " sec\n";

    delete[] arr;
}

int main() {
    srand(time(0));
    testHeapSort(1000);
    testHeapSort(2000);
    testHeapSort(3000);
    return 0;
}
```
****

d) Write a program to implement Weighted UNION and Collapsing FIND operations

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

const int N = 100;

int parent[N], sizeArr[N];

void makeSet(int n) {
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        sizeArr[i] = 1;      }
}

int findSet(int x) {
    if (parent[x] != x)
        parent[x] = findSet(parent[x]);
    return parent[x];
}

void unionSet(int a, int b) {
    int rootA = findSet(a);
    int rootB = findSet(b);
    if (rootA == rootB)
return;
        if (sizeArr[rootA] < sizeArr[rootB]) {
        parent[rootA] = rootB;
        sizeArr[rootB] += sizeArr[rootA];
    } else {
        parent[rootB] = rootA;
        sizeArr[rootA] += sizeArr[rootB];
    }
}

int main() {
    int n = 8;
    makeSet(n);

    unionSet(0, 1);
    unionSet(2, 3);
    unionSet(1, 2);
    unionSet(4, 5);

    cout << "Find(3) = " << findSet(3) << endl;
    cout << "Find(5) = " << findSet(5) << endl;
    cout << "Find(1) = " << findSet(1) << endl;

    return 0;
}
```

3) Divide and Conquer

a) Write a program for searching element form given array using search form =1000, 2000, 3000. Find exact time of execution.

b)Write a program to find minimum and maximum from a given array Using maxmin.

c) Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using –

d) Merge Sort

e) Quick Sort

f) Write a program for matrix multiplication using Strassen’s Matrix Multiplication

****

a) Write a program for searching element form given array using search form =1000, 2000, 3000. Find exact time of execution.

```
#include <iostream>

#include <ctime>

using namespace std;


int binSearch(int a[], int l, int r, int x) {

   if (l <= r) {

      int m = (l + r) / 2;

      if (a[m] == x) return m;

      if (a[m] > x) return binSearch(a, l, m - 1, x);

      return binSearch(a, m + 1, r, x);

   }

   return -1;

}


int main() {

   int n, x;

   cout << "Enter size (1000/2000/3000): ";
```

```
    cin >> n;
    int *a = new int[n];
    for (int i = 0; i < n; i++) a[i] = i + 1;

    cout << "Enter element to search: ";
    cin >> x;

    clock_t start = clock();
    int idx = binSearch(a, 0, n - 1, x);
    clock_t end = clock();

    if (idx != -1) cout << "Found at index " << idx << endl;
    else cout << "Not Found\n";

    cout << "Time = " << double(end - start) / CLOCKS_PER_SEC << " sec\n";
    delete[] a;
}
```
****

b)Write a program to find minimum and maximum from a given array Using maxmin.

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

struct Pair {
    int min, max;
};

Pair MAXMIN(int arr[], int low, int high) {
    Pair result, left, right;
    if (low == high) {
        result.min = result.max = arr[low];
        return result;
    }
    if (high == low + 1) {
        if (arr[low] < arr[high]) {
            result.min = arr[low];
            result.max = arr[high];
        } else {
            result.min = arr[high];
            result.max = arr[low];
        }
        return result;
    }
    int mid = (low + high) / 2;
    left  = MAXMIN(arr, low, mid);
    right = MAXMIN(arr, mid + 1, high);

    result.min = (left.min < right.min) ? left.min : right.min;
    result.max = (left.max > right.max) ? left.max : right.max;

    return result;
}

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

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

    Pair ans = MAXMIN(arr, 0, n - 1);

    cout << "Minimum element = " << ans.min << endl;
    cout << "Maximum element = " << ans.max << endl;
    return 0;
}
```

****

c) Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using –

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

// Merge function
void merge(int arr[], int l, int m, int r, bool ascending) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[10000], R[10000]; // temporary arrays

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        if ((ascending && L[i] <= R[j]) || (!ascending && L[i] >= R[j]))
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}

// Merge Sort function
void mergeSort(int arr[], int l, int r, bool ascending) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m, ascending);
        mergeSort(arr, m + 1, r, ascending);
        merge(arr, l, m, r, ascending);
    }
}

// Measure execution time
double measureTime(int n, bool ascending) {
    int arr[10000];
    for (int i = 0; i < n; i++)
```

```
        arr[i] = rand() % 10000 + 1;

    clock_t start = clock();
    mergeSort(arr, 0, n - 1, ascending);
    clock_t end = clock();

    return double(end - start) / CLOCKS_PER_SEC;
}

// Main function
int main() {
    srand(time(0));

    int sizes[] = {1000, 2000, 3000};

    cout << "Merge Sort Execution Time\n";
    cout << "----------------------------------\n";

    for (int i = 0; i < 3; i++) {
        int n = sizes[i];
        double asc_time = measureTime(n, true);
        double desc_time = measureTime(n, false);

        cout << "Array size: " << n << endl;
        cout << "  Ascending  : " << asc_time << " seconds" << endl;
        cout << "  Descending : " << desc_time << " seconds" << endl;
        cout << "----------------------------------\n";
    }
    return 0;
}
```

****

d) Merge sort

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

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int L[n1], R[n2];

    for(int i = 0; i < n1; i++) L[i] = arr[l+i];
    for(int i = 0; i < n2; i++) R[i] = arr[m+1+i];

    int i=0, j=0, k=l;
    while(i<n1 && j<n2) {
        if(L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while(i<n1) arr[k++] = L[i++];
    while(j<n2) arr[k++] = R[j++];
}
void mergeSort(int arr[], int l, int r) {
    if(l < r) {
        int m = l + (r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
int main() {
    srand(time(0));
    int n = 1000;
    int arr[n];
    for(int i=0;i<n;i++) arr[i] = rand()%10000;

    clock_t start = clock();
    mergeSort(arr, 0, n-1);
    clock_t end = clock();

    cout << "Sorted Array (first 20 elements): ";
    for(int i=0;i<20;i++) cout << arr[i] << " ";
    cout << "\nExecution time: " << double(end-start)/CLOCKS_PER_SEC << " seconds" <<
endl;

    return 0;
}
```

****

e) Quick sort

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

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;
}
void quickSort(int arr[], int low, int high) {
    if(low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
void fillRandom(int arr[], int n) {
    for(int i = 0; i < n; i++) arr[i] = rand() % 10000;
}
void printArray(int arr[], int n) {
    int limit = (n > 20) ? 20 : n;  // show only first 20 elements
    for(int i = 0; i < limit; i++) cout << arr[i] << " ";
    if(n > 20) cout << "...";
    cout << endl;
}

int main() {
    srand(time(0));
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    int arr[n];
    fillRandom(arr, n);
    cout << "Original array: ";
    printArray(arr, n);

    clock_t start = clock();
    quickSort(arr, 0, n - 1);
    clock_t end = clock();

    cout << "Sorted array (Ascending): ";
    printArray(arr, n);

    double duration = double(end - start) / CLOCKS_PER_SEC;
    cout << "Execution time: " << duration << " seconds" << endl;

    return 0;
}
```
****

F) Write a program for matrix multiplication using Strassen’s Matrix Multiplication

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

class Matrix {
    int A[2][2], B[2][2], result[2][2];

public:
    void Get();
    void Mult();
    void Put();
};

void Matrix::Get() {
    cout << "\nEnter the first 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> A[i][j];

    cout << "\nEnter the second 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> B[i][j];
}
void Matrix::Mult() {
    int p,q,r,s,t,u,v;
    p = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
    q = (A[1][0] + A[1][1]) * B[0][0];
    r = A[0][0] * (B[0][1] - B[1][1]);
    s = A[1][1] * (B[1][0] - B[0][0]);
    t = (A[0][0] + A[0][1]) * B[1][1];
    u = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
    v = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);

    result[0][0] = p + s - t + v;
    result[0][1] = r + t;
    result[1][0] = q + s;
    result[1][1] = p + r - q + u;
```

```
}
void Matrix::Put() {
    cout << "\nResult is:\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++)
            cout << result[i][j] << "\t";
        cout << "\n";
    }
}

int main() {
    Matrix m;
    m.Get();
    m.Mult();
    m.Put();
    return 0;
}
```
****

4) Greedy Technique:

a)Write a program to find solution of Fractional Knapsack instance.

b)Write a program to find Minimum Spanning Tree using Prim's algorithm.

c)Write a program to find Minimum Spanning tree using Kruskal's algorithm.

d)Write a program to find Single Source Shortest Path using Dijkstra's algorithm

****

a)Write a program to find solution of Fractional Knapsack instance.

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

int main()
{
    int n, W;
    cout << "Number of items: ";
    cin >> n;
    double value[n], weight[n];
    cout << "Enter value and weight of each item:\n";
    for (int i = 0; i < n; i++)
        cin >> value[i] >> weight[i];

    cout << "Knapsack capacity: ";
    cin >> W;

    double ratio[n], total = 0;
    bool taken[n] = {0};

    for (int i = 0; i < n; i++)
        ratio[i] = value[i] / weight[i];
    int remaining = W;
    while (remaining > 0) {
        int idx = -1;
        double maxRatio = 0;
        for (int i = 0; i < n; i++)
            if (!taken[i] && ratio[i] > maxRatio) {
                maxRatio = ratio[i];
                idx = i;
            }

        if (idx == -1) break;

        if (weight[idx] <= remaining) {
            total += value[idx];
            remaining -= weight[idx];
        } else {
            total += value[idx] * remaining / weight[idx];
            remaining = 0;
        }

        taken[idx] = true;
    }

    cout << "Maximum value in knapsack = " << total << endl;
    return 0;
}
```
****

b)Write a program to find Minimum Spanning Tree using Prim’s algorithm.

```
#include <iostream>
#include <climits>
```

```
using namespace std;

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

    int graph[n][n];
    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];

    bool selected[n] = {false};
    int parent[n];
    int key[n];

    for (int i = 0; i < n; i++)
        key[i] = INT_MAX;

    key[0] = 0;
    parent[0] = -1;

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

        for (int i = 0; i < n; i++)
            if (!selected[i] && (u == -1 || key[i] < key[u]))
                u = i;
        selected[u] = true;
        for (int v = 0; v < n; v++)
            if (graph[u][v] && !selected[v] && graph[u][v] < key[v]) {
                key[v] = graph[u][v];
                parent[v] = u;
            }
    }
    cout << "Edge \tWeight\n";
    for (int i = 1; i < n; i++)
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << "\n";

    return 0;
}
```

****

c)Write a program to find Minimum Spanning tree using Kruskal’s algorithm.

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

int parent[100];
int find(int x) {
    if (parent[x] == x) return x;
    return parent[x] = find(parent[x]);
}

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;
    cout << "Enter number of edges: ";
    cin >> e;
    int u[e], v[e], w[e];
    cout << "Enter edges (u v weight):\n";
    for (int i = 0; i < e; i++)
        cin >> u[i] >> v[i] >> w[i];

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


        for (int i = 0; i < e - 1; i++)
        for (int j = 0; j < e - i - 1; j++)
            if (w[j] > w[j + 1]) {
                swap(w[j], w[j + 1]);
                swap(u[j], u[j + 1]);
                swap(v[j], v[j + 1]);
            }
    cout << "Edges in MST:\n";
    int count = 0;
    for (int i = 0; i < e && count < n - 1; i++) {
        int set_u = find(u[i]);
        int set_v = find(v[i]);
        if (set_u != set_v) {
            cout << u[i] << " - " << v[i] << " : " << w[i] << "\n";
            parent[set_v] = set_u; // union
            count++;
        }
    }
    return 0;
}
```

****

d)Write a program to find Single Source Shortest Path using Dijkstra’s algorithm

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

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

    int graph[n][n];
    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;

    int dist[n];
    bool visited[n];

    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 = -1;
        for (int i = 0; i < n; i++)
            if (!visited[i] && (u == -1 || dist[i] < dist[u]))
                u = i;

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

    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < n; i++)
        cout << i << "\t" << dist[i] << "\n";

    return 0;
}
```
****

5) Dynamic Programming

a)Write a program to find solution of Knapsack Instance (0/1).

b)Write a program to find solution of Matrix Chain Multiplication.

c)Write a program to find shortest path using All Pair Shortest Path algorithm.

d)Write a program to Traverse Graph – Depth First Search, Breadth First Search

****

a)Write a program to find solution of Knapsack Instance (0/1)

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

int knapsack(int W, vector<int>& wt, vector<int>& val, int n) {
    vector<vector<int> > dp(n + 1, vector<int>(W + 1, 0));



    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 (wt[i - 1] <= w)
                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    return dp[n][W];
}
```

```
int main() {
    int n, W;
    cout << "Enter number of items: ";
    cin >> n;
    vector<int> val(n), wt(n);

    cout << "Enter values of items: ";
    for (int i = 0; i < n; i++) cin >> val[i];

    cout << "Enter weights of items: ";
    for (int i = 0; i < n; i++) cin >> wt[i];

    cout << "Enter maximum capacity of knapsack: ";
    cin >> W;

    int maxValue = knapsack(W, wt, val, n);
    cout << "Maximum value that can be carried: " << maxValue << endl;

    return 0;
}
```
****

b)Write a program to find solution of Matrix Chain Multiplication.

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

int matrixChainOrder(const vector<int>& dims) {
    int n = dims.size() - 1; // Number of matrices
    vector<vector<int> > dp(n, vector<int>(n, 0));

    for (int length = 2; length <= n; length++) { // chain length
        for (int i = 0; i <= n - length; i++) {
            int j = i + length - 1;
            dp[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int cost = dp[i][k] + dp[k + 1][j] + dims[i]*dims[k+1]*dims[j+1];
                if (cost < dp[i][j])
                    dp[i][j] = cost;
            }
        }
    }

    return dp[0][n-1];
}

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

    if (n <= 0) {
        cout << "Number of matrices must be positive." << endl;
        return 0;
    }
    vector<int> dims(n + 1);
    cout << "Enter dimensions of matrices (A1: dims[0]xdims[1], A2: dims[1]xdims[2], ...): ";
    for (int i = 0; i <= n; i++) cin >> dims[i];
    int minCost = matrixChainOrder(dims);
    cout << "Minimum number of multiplications: " << minCost << endl;
    return 0;
}
```
****

c)Write a program to find shortest path using All Pair Shortest Path algorithm.

```
#include <iostream>
using namespace std;
const int INF = 1e9;

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

    int dist[n][n];

    cout << "Enter adjacency matrix (-1 for no edge):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> dist[i][j];
            if (dist[i][j] == -1)
                dist[i][j] = INF;
        }
    }

    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            if (dist[i][k] < INF && dist[k][j] < INF)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

    cout << "\nAll-pairs shortest distances:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                cout << "INF" << " ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```
****

d)Write a program to Traverse Graph – Depth First Search, Breadth First Search

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

const int MAX = 20; // maximum number of vertices

void DFS(int adj[MAX][MAX], int visited[MAX], int n, int start) {
    cout << start << " ";
    visited[start] = 1;
    for (int i = 0; i < n; i++) {
        if (adj[start][i] != 0 && !visited[i])
            DFS(adj, visited, n, i);
    }
}

void BFS(int adj[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};
    queue<int> q;
    visited[start] = 1;
    q.push(start);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";
        for (int i = 0; i < n; i++) {
            if (adj[v][i] != 0 && !visited[i]) {
                visited[i] = 1;
                q.push(i);
            }
        }
    }
}

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

    int adj[MAX][MAX];
    cout << "Enter adjacency matrix (0 for no edge):\n";
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < n; j++)
            cin >> adj[i][j];

    int visited[MAX] = {0};
    int start;
    cout << "Enter starting vertex (0 to " << n-1 << "): ";
    cin >> start;

    cout << "\nDFS traversal: ";
    DFS(adj, visited, n, start);

    cout << "\nBFS traversal: ";
    BFS(adj, n, start);

    return 0;
}
```
****

6) Backtracking

a)Write a program to find all solutions for N-Queen problem using Backtracking.

b) Write a program for Graph Coloring using backtracking.

****

a)Write a program to find all solutions for N-Queen problem using Backtracking.

```
#include <iostream>

#include <cmath>

using namespace std;


int board[20], N, solution = 0;


bool safe(int row, int col) {

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

        if (board[i] == col || abs(board[i] - col) == row - i)

            return false;

    }

    return true;

}


void solve(int row) {

    if (row == N) {

        solution++;

        cout << "Solution " << solution << ":\n";

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

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

                cout << (board[i] == j ? "Q " : ". ");

            cout << endl;

        }

        cout << endl;
        return;
    }
    for (int col = 0; col < N; col++) {
        if (safe(row, col)) {
            board[row] = col;
            solve(row + 1);
        }
    }
}

int main() {
    cout << "Enter number of queens: ";
    cin >> N;
    solve(0);
    cout << "Total solutions: " << solution << endl;
    return 0;
}
```
****

b) Write a program for Graph Coloring using backtracking.

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

int graph[10][10], color[10], N, m;

bool safe(int v, int c) {
    for (int i = 0; i < N; i++)
        if (graph[v][i] && color[i] == c)
            return false;
    return true;
```

```
}

bool graphColoring(int v) {
    if (v == N) return true;
    for (int c = 1; c <= m; c++) {
        if (safe(v, c)) {
            color[v] = c;
            if (graphColoring(v + 1)) return true;
            color[v] = 0; // Backtrack
        }
    }
    return false;
}
int main() {
    cout << "Enter number of vertices: ";
    cin >> N;
    cout << "Enter adjacency matrix:\n";
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            cin >> graph[i][j];

    cout << "Enter number of colors: ";
    cin >> m;

    if (graphColoring(0)) {
        cout << "Solution exists. Colors:\n";
        for (int i = 0; i < N; i++)
            cout << "Vertex " << i << " --> Color " << color[i] << endl;
    } else {
        cout << "No solution exists.\n";
    }
    return 0;
}
```

****