Practical 1: Finding maximum from an array without recursion

```
#include <iostream>
using namespace std;
int findMax(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 arr[] = {5, 12, 3, 19, 8};
   int n = sizeof(arr) / sizeof(arr[0]);

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

Practical 2: Calculating Binomial Coefficient without recursion (Using formula: B(n, m) = B(n−1, m−1) + B(n−1, m), B(n,0)=B(n,n)=1)

```
#include<iostream>
using namespace std;
class Bino
{
 int k,S[30],add,top;
 public: int Binomial(int,int);
};
int Bino :: Binomial(int i,int j)
```

```
{
 top=-1;
 k=0;
 L1:if ((i!=j)&&(j!=0))
 {
  S[++top]=i-1;
  S[++top]=j-1;
  S[++top]=3;
       S[++top]=i-1;
  S[++top]=j;
  S[++top]=3;
 }
 else
     k++;
  if(top==-1)
       return(k);
    else
    {
     add=S[top--];
     j=S[top--];
     i=S[top--];
     if(add==3)
     {
      goto L1;
  }
 }
 return 0;
}
int main()
```

```
{
 Bino B;
 int a,b,val;
 cout<<"\n Enter two values:";
 cin>>a>>b;
 if (a>b)
 {
  val=B.Binomial(a,b);
  cout<<"\n Binomial coefficient of"<<a<<"&"<<b<<"is:"<<val;
 }
 else
 {
  cout<<"\n invalid input";
 }
}
```

Practical 3: Searching an element in an array without recursion

```
#include<iostream>
using namespace std;
class SearchEle
{
 int S[50],addr,top,A[50],n,i,no,j,k;
 public:
  SearchEle()
  {
   i=1;
  }
  void Get();
```

```
 void Search();
};
void SearchEle :: Get()
{
 cout<<"\n Enter the size of elements:";
 cin>>n;

 cout<<"\n Enter the elements:";
 for(int m=1;m<=n;m++)
 {
  cin>>A[m];
 }
 cout<<"\n Enter the element to be searched:";
 cin>>no;
}
void SearchEle :: Search()
{
 int j,k,top=0;
 L1:if(i<n)
 {
  S[++top]=i;
  S[++top]=2;
  i++;
  goto L1;

  L2:j=S[top--];
  if(A[j]==no)
  {
   k=j;
```

```
  cout<<"\n Element is found at position:"<<k;
  return;
 }
 else
 {
  k=0;
 }
}
if(top==0 && k==0)
{
 cout<<"\n Element is not found:";
}
else
{
 addr=S[top--];
 if(addr==2)
  goto L2;
}
}
int main()
{
 SearchEle S;
 int val;

 S.Get();
 S.Search();
}
```

Practical 4: Create Max/Min Heap using INSERT operation

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

class InsertMaxHeap {
   int a[20], n;
public:
   void Insert(int);
   void Get();
   void Show();
};

void InsertMaxHeap::Get() {
   cout << "\nEnter the size of heap: ";
   cin >> n;
   cout << "Enter the elements:\n";
   for(int i = 1; i <= n; i++) {
      cin >> a[i];
   }

   cout << "\nBefore building heap:\n";
   Show();

   for(int i = 2; i <= n; i++) {
      Insert(i);
   }
}

void InsertMaxHeap::Insert(int index) {
```

```
    int i = index;
    int item = a[i];
    while(i > 1 && a[i / 2] < item) {
        a[i] = a[i / 2];
        i = i / 2;
    }
    a[i] = item;
}

void InsertMaxHeap::Show() {
    for(int i = 1; i <= n; i++) {
        cout << a[i] << "\t";
    }
    cout << endl;
}

int main() {
    InsertMaxHeap heap;
    heap.Get();
    cout << "\nAfter building max heap:\n";
    heap.Show();
    return 0;
}
```

Practical 5: Create Max/Min Heap using ADJUST/HEAPIFY operation

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

class AdjustMinHeap {
    int a[10], n;
```

```
public:
   void Adjust(int i, int n);
   void Heapify(int n);
   void Get();
   void Show();
};

void AdjustMinHeap::Get() {
   cout << "\nEnter the size of heap: ";
   cin >> n;

   cout << "\nEnter the elements:\n";
   for(int i = 1; i <= n; i++) {
      cin >> a[i];
   }

   Heapify(n);
}

void AdjustMinHeap::Adjust(int i, int n) {
   int j = 2 * i;
   int item = a[i];
   while(j <= n) {
      if(j < n && a[j] > a[j + 1])
         j++;
      if(item <= a[j])
         break;
      a[j / 2] = a[j];
```

```
        j = 2 * j;
    }
    a[j / 2] = item;
}

void AdjustMinHeap::Heapify(int n) {
    for(int i = n / 2; i >= 1; i--)
        Adjust(i, n);
}

void AdjustMinHeap::Show() {
    cout << "\nMin Heap is:\n";
    for(int i = 1; i <= n; i++)
        cout << a[i] << "\t";
    cout << endl;
}

int main() {
    AdjustMinHeap heap;
    heap.Get();
    heap.Show();
    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////
#include<iostream>
using namespace std;

class AdjustMaxHeap {
    int a[10], n;
```

```
public:
   void Adjust(int i, int n);
   void Heapify(int n);
   void Get();
   void Show();
};

void AdjustMaxHeap::Get() {
   cout << "\nEnter the size of heap: ";
   cin >> n;

   cout << "\nEnter the elements:\n";
   for(int i = 1; i <= n; i++) {
      cin >> a[i];
   }

   Heapify(n);
}

void AdjustMaxHeap::Adjust(int i, int n) {
   int j = 2 * i;
   int item = a[i];
   while(j <= n) {
      if(j < n && a[j] < a[j + 1])
         j++;
      if(item >= a[j])
         break;
      a[j / 2] = a[j];
```

```
        j = 2 * j;
    }
    a[j / 2] = item;
}

void AdjustMaxHeap::Heapify(int n) {
    for(int i = n / 2; i >= 1; i--)
        Adjust(i, n);
}

void AdjustMaxHeap::Show() {
    cout << "\nMax Heap is:\n";
    for(int i = 1; i <= n; i++)
        cout << a[i] << "\t";
    cout << endl;
}

int main() {
    AdjustMaxHeap heap;
    heap.Get();
    heap.Show();
    return 0;
}
```

Practical 6: Sort a given array in Ascending/Descending order using Heap Sort with n = 1000, 2000, 3000 and measure exact execution time

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
```

```
#include <cstdlib>
using namespace std;
using namespace chrono;
// Heapify for ascending order
void heapifyAsc(vector<int> &arr, int n, int i) {
int largest = i; // root
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) {
      swap(arr[i], arr[largest]);
      heapifyAsc(arr, n, largest);
   }
}

// Heapify for descending order
void heapifyDesc(vector<int> &arr, int n, int i) {
   int smallest = i; // root
   int l = 2*i + 1;
   int r = 2*i + 2;

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

   if (smallest != i) {
      swap(arr[i], arr[smallest]);
      heapifyDesc(arr, n, smallest);
```

```
    }
}

// Heap Sort for ascending
void heapSortAsc(vector<int> &arr) {
    int n = arr.size();

    // Build max heap
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyAsc(arr, n, i);

    // Extract elements
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapifyAsc(arr, i, 0);
    }
}

// Heap Sort for descending
void heapSortDesc(vector<int> &arr) {
    int n = arr.size();

    // Build min heap
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyDesc(arr, n, i);

    // Extract elements
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
```

```
        heapifyDesc(arr, i, 0);
    }
}

// Function to generate random array
vector<int> generateArray(int n) {
    vector<int> arr(n);
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 10000; // Random values from 0 to 9999
    return arr;
}

// Function to measure execution time
void measureHeapSort(int n) {
vector<int> arr = generateArray(n);
vector<int> arrAsc = arr;
vector<int> arrDesc = arr;
cout << "\nArray Size: " << n;
auto startAsc = high_resolution_clock::now();
heapSortAsc(arrAsc);
auto stopAsc = high_resolution_clock::now();
auto durationAsc = duration_cast<microseconds>(stopAsc - startAsc);
cout << "\n→ Ascending Sort Time: " << durationAsc.count() << " microseconds";
auto startDesc = high_resolution_clock::now();
heapSortDesc(arrDesc);
auto stopDesc = high_resolution_clock::now();
auto durationDesc = duration_cast<microseconds>(stopDesc - startDesc);
cout << "\n→ Descending Sort Time: " << durationDesc.count() << " microseconds\n";
}
```

```
int main() {
srand(time(0)); // Seed for random number generation
measureHeapSort(1000);
measureHeapSort(2000);
measureHeapSort(3000);
return 0;
}
```

Practical 7: Implement Weighted UNION and Collapsing FIND operations (Disjoint Set)

```
#include <iostream>
using namespace std;
int parent[10], size[10];
int find(int x) {
return parent[x] = (parent[x] == x) ? x : find(parent[x]);
}
void unionSet(int x, int y) {
int rx = find(x), ry = find(y);
if (rx == ry) return;
if (size[rx] < size[ry]) swap(rx, ry);
parent[ry] = rx;
size[rx] += size[ry];
}
int main() {
for (int i = 0; i < 10; i++) parent[i] = i, size[i] = 1;
unionSet(1, 2);
unionSet(2, 3);
unionSet(4, 5);
unionSet(5, 6);
unionSet(3, 6);
cout << "Find(6): " << find(6) << "\n";
```

```
cout << "Parent: ";
for (int i = 0; i < 10; i++) cout << parent[i] << " ";
}
```

Practical 8: Search an element from a given array using Binary Search

```
#include<iostream>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
using namespace std;

class BSearch {
public:
   int A[100], Size;
   int Get();
   void sort();
   int Search(int, int, int);
   void show(int);
};

int BSearch::Get() {
   cout << "\n Enter the Size of List: ";
   cin >> Size;

   cout << "\n The elements of List are:\n";
   for(int i = 1; i <= Size; i++) {
      A[i] = rand() % 100;
      cout << A[i] << endl;
   }
```

```
    sort();

    cout << "\n After sorting:\n";
    for(int i = 1; i <= Size; i++) {
        cout << A[i] << endl;
    }

    return 0;
}

void BSearch::sort() {
    for(int i = 1; i <= Size; i++) {
        for(int j = i + 1; j <= Size; j++) {
            if(A[i] > A[j]) {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
        }
    }
}

int BSearch::Search(int i, int j, int x) {
    int mid;
    if(j < i)
        return 0;

    mid = (i + j) / 2;
```

```
    if(x == A[mid])
        return mid;
    else if(x < A[mid])
        return Search(i, mid - 1, x);
    else
        return Search(mid + 1, j, x);
}

void BSearch::show(int x) {
    int t = Search(1, Size, x);
    if(t == 0)
        cout << "\n Element is not found";
    else
        cout << "\n Element is found at location " << t;
}

int main() {
    BSearch b;
    int No;
    //clrscr(); // Clears screen (Turbo C++ style)

    int start = clock();

    b.Get();
    cout << "\n Enter element to search: ";
    cin >> No;
    b.show(No);

    int end = clock();
```

```
   cout << "\n The execution time is: " << (end - start) / CLK_TCK << " seconds";


  //  getch(); // Waits for key press
    return 0;
}
```

Practical 9: Write a program to find minimum and maximum from a given array using MAXMIN

```
#include<iostream>
using namespace std;
int main()
{
 int no;
 cout<<"Enter the size of array:";
 cin>>no;
 int a[no];
 for(int i=0;i<no;i++)
 {
  cin>>a[i];
 }
 int max=a[0],min=a[0];
 for(int i=1;i<no;i++)
 {
  if(a[i]>max)
   max=a[i];
  if(a[i]<min)
   min=a[i];
 }
 cout<<"Maximum Element"<<max;
 cout<<"Minimum Element"<<min;
```

```
 return 0;
}
```

Practical 10: Sort a given array in using Merge Sort

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

class Number {
   int a[50], n;

public:
   void getdata();
   void mergesort(int low, int high);
   void merge(int low, int mid, int high);
};

void Number::getdata() {
   cout << "\nNumber of Elements: ";
   cin >> n;

   cout << "\nEnter the Elements:\n";
   for(int i = 0; i < n; i++) {
      cin >> a[i];
   }

   cout << "\nYour Array is:\n";
   for(int i = 0; i < n; i++) {
```

```
        cout << a[i] << "\t";
    }

    mergesort(0, n - 1);

    cout << "\nThe array after sorting:\n";
    for(int i = 0; i < n; i++) {
        cout << a[i] << "\t";
    }
    cout << endl;
}

void Number::mergesort(int low, int high) {
    if(low < high) {
        int mid = (low + high) / 2;
        mergesort(low, mid);
        mergesort(mid + 1, high);
        merge(low, mid, high);
    }
}

void Number::merge(int low, int mid, int high) {
    int b[50];
    int h = low, i = low, j = mid + 1;

    while(h <= mid && j <= high) {
        if(a[h] <= a[j]) {
            b[i] = a[h];
            h++;
```

```
        } else {
            b[i] = a[j];
            j++;
        }
        i++;
    }

    while(h <= mid) {
        b[i] = a[h];
        h++;
        i++;
    }

    while(j <= high) {
        b[i] = a[j];
        j++;
        i++;
    }

    for(int k = low; k <= high; k++) {
        a[k] = b[k];
    }
}

int main() {
    Number obj;
    clock_t start, end;

    start = clock();
```

```
    obj.getdata();
    end = clock();

    double time_taken = double(end - start) / CLOCKS_PER_SEC;
    cout << "\nThe execution time is: " << time_taken << " seconds\n";

    return 0;
}i]
```

Practical 11: Quick Sort

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
using namespace std;

class Number {
    int a[50], n;

public:
    void getdata();
    void quicksort(int low, int high);
    int partition(int low, int high);
};

void Number::getdata() {
    cout << "\nNumber of Elements: ";
    cin >> n;

    cout << "\nEnter the Elements:\n";
```

```
   for(int i = 0; i < n; i++) {
      cin >> a[i];
   }

   cout << "\nYour Array is:\n";
   for(int i = 0; i < n; i++) {
      cout << a[i] << "\t";
   }

   quicksort(0, n - 1);

   cout << "\nThe array after sorting:\n";
   for(int i = 0; i < n; i++) {
      cout << a[i] << "\t";
   }
   cout << endl;
}

void Number::quicksort(int low, int high) {
   if(low < high) {
      int pivotIndex = partition(low, high);
      quicksort(low, pivotIndex - 1);
      quicksort(pivotIndex + 1, high);
   }
}

int Number::partition(int low, int high) {
   int pivot = a[low];
   int i = low + 1;
```

```
    int j = high;

    while(true) {
        while(i <= high && a[i] <= pivot)
            i++;
        while(j >= low && a[j] > pivot)
            j--;
        if(i < j)
            swap(a[i], a[j]);
        else
            break;
    }
    swap(a[low], a[j]);
    return j;
}

int main() {
    Number obj;
    clock_t start, end;

    //clrscr(); // Turbo C++ style screen clear

    start = clock();
    obj.getdata();
    end = clock();

    cout << "\nThe execution time is: " << (end - start) / CLK_TCK << " seconds";

   // getch(); // Wait for key press
```

```
    return 0;
}
```

Practical 13: Write a program to find solution of Fractional Knapsack instance.

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

struct Item {
    int weight;
    int value;
};

// Comparator to sort items by value-to-weight ratio
bool compare(Item a, Item b) {
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

double fractionalKnapsack(Item items[], int n, int capacity) {
    sort(items, items + n, compare);

    double totalValue = 0.0;
    int currentWeight = 0;

    for(int i = 0; i < n; i++) {
        if(currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
```

```
        } else {
            int remain = capacity - currentWeight;
            totalValue += items[i].value * ((double)remain / items[i].weight);
            break;
        }
    }

    return totalValue;
}

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

    Item items[n];
    cout << "Enter value and weight of each item:\n";
    for(int i = 0; i < n; i++) {
        cout << "Item " << i + 1 << " - Value: ";
        cin >> items[i].value;
        cout << "Item " << i + 1 << " - Weight: ";
        cin >> items[i].weight;
    }

    cout << "Enter knapsack capacity: ";
    cin >> capacity;

    double maxValue = fractionalKnapsack(items, n, capacity);
    cout << "\nMaximum value in knapsack = " << maxValue << endl;
```

```
    return 0;
}
```

1 4 Minimum Spanning Tree using Kruskal's Algorithm

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

struct Edge {
    int u, v, w;
};

int findParent(int u, vector<int> &parent) {
    if(parent[u] == u) return u;
    return parent[u] = findParent(parent[u], parent);
}

void unionSet(int u, int v, vector<int> &parent) {
    parent[findParent(u, parent)] = findParent(v, parent);
}

int main() {
    int n, e;
    cout << "Enter number of vertices and edges: ";
    cin >> n >> e;
    vector<Edge> edges(e);
```

```
    cout << "Enter edges (u v weight):\n";
    for(int i = 0; i < e; i++)
        cin >> edges[i].u >> edges[i].v >> edges[i].w;

    sort(edges.begin(), edges.end(), [](Edge a, Edge b) {
        return a.w < b.w;
    });

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

    cout << "Edges in MST:\n";
    for(auto &edge : edges) {
        if(findParent(edge.u, parent) != findParent(edge.v, parent)) {
            cout << edge.u << " - " << edge.v << " : " << edge.w << endl;
            unionSet(edge.u, edge.v, parent);
        }
    }

    return 0;
}
```

15: Write a program to find Minimum Spanning Tree using Prim's algorithm.

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

int main() {
```

```
int n;
cout << "Enter number of vertices: ";
cin >> n;

vector<vector<int>> graph(n, vector<int>(n));
cout << "Enter adjacency matrix:\n";
for(int i = 0; i < n; i++)
   for(int j = 0; j < n; j++)
      cin >> graph[i][j];

vector<int> key(n, INT_MAX);
vector<int> parent(n, -1);
vector<bool> mstSet(n, false);

key[0] = 0;

for(int count = 0; count < n - 1; count++) {
   int u = -1;
   for(int i = 0; i < n; i++) {
      if(!mstSet[i] && (u == -1 || key[i] < key[u]))
         u = i;
   }

   mstSet[u] = true;

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

```
            }
        }
    }

    cout << "\nEdges in MST:\n";
    for(int i = 1; i < n; i++) {
        if(parent[i] != -1)
            cout << parent[i] << " - " << i << " : " << graph[i][parent[i]] << endl;
    }

    return 0;
}
```

16: Write a program to find Single Source Shortest Path using Dijkstra'salgorithm

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

int findMinVertex(vector<int>& distance, vector<bool>& visited, int n) {
    int minVertex = -1;
    for(int i = 0; i < n; i++) {
        if(!visited[i] && (minVertex == -1 || distance[i] < distance[minVertex]))
            minVertex = i;
    }
    return minVertex;
}

void dijkstra(vector<vector<int> > &graph, int source) {
    int n = graph.size();
```

```
    vector<int> distance(n, INT_MAX);
    vector<bool> visited(n, false);

    distance[source] = 0;

    for(int i = 0; i < n - 1; i++) {
        int u = findMinVertex(distance, visited, n);
        visited[u] = true;

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

    cout << "\nShortest distances from source vertex " << source << ":\n";
    for(int i = 0; i < n; i++) {
        cout << "To vertex " << i << " : " << distance[i] << endl;
    }
}

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

```
    vector<vector<int> > graph(n, vector<int>(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];

    cout << "Enter source vertex (0 to " << n - 1 << "): ";
    cin >> source;

    dijkstra(graph, source);

    return 0;
}
```

17:Write a program to find solution of Matrix Chain Multiplication.

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

// Function to compute minimum number of multiplications
int matrixChainOrder(vector<int>& dims, int n) {
    vector<vector<int> > dp(n, vector<int>(n, 0));

    // l is chain length
    for(int l = 2; l < n; l++) {
        for(int i = 1; i < n - l + 1; i++) {
            int j = i + l - 1;
            dp[i][j] = INT_MAX;
```

```
            for(int k = i; k < j; k++) {
                int cost = dp[i][k] + dp[k+1][j] + dims[i-1] * dims[k] * dims[j];
                if(cost < dp[i][j])
                    dp[i][j] = cost;
            }
        }
    }

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

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

    vector<int> dims(n + 1);
    cout << "Enter dimensions (P0 P1 ... Pn):\n";
    for(int i = 0; i <= n; i++)
        cin >> dims[i];

    int minMultiplications = matrixChainOrder(dims, n + 1);
    cout << "\nMinimum number of scalar multiplications = " << minMultiplications << endl;

    return 0;
}
```

18 : Write a program to find shortest path using All Pair Shortest Path algorithm.

```
#include <iostream>
#include <vector>
```

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

void floydWarshall(vector<vector<int> > &graph, int n) {
   vector<vector<int> > dist = graph;

   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] != INT_MAX && dist[k][j] != INT_MAX &&
               dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
            }
         }
      }
   }

   cout << "\nShortest distances between every pair of vertices:\n";
   for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
         if(dist[i][j] == INT_MAX)
            cout << "INF\t";
         else
            cout << dist[i][j] << "\t";
      }
      cout << endl;
   }
}
```

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

   vector<vector<int> > graph(n, vector<int>(n));
   cout << "Enter adjacency matrix (use 99999 for INF):\n";
   for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++) {
         cin >> graph[i][j];
         if(graph[i][j] == 99999)
            graph[i][j] = INT_MAX;
      }

   floydWarshall(graph, n);

   return 0;
}
```

19: Write a program to Traverse Graph – Depth First Search, Breadth First Search

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

class Graph {
   int V;
   vector<vector<int> > adj;

public:
```

```
Graph(int V) {
    this->V = V;
    adj.resize(V);
}

void addEdge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u); // For undirected graph
}

void DFSUtil(int v, vector<bool>& visited) {
    visited[v] = true;
    cout << v << " ";

    for(int u : adj[v]) {
        if(!visited[u])
            DFSUtil(u, visited);
    }
}

void DFS(int start) {
    vector<bool> visited(V, false);
    cout << "\nDFS Traversal starting from vertex " << start << ": ";
    DFSUtil(start, visited);
    cout << endl;
}

void BFS(int start) {
    vector<bool> visited(V, false);
```

```
        queue<int> q;

        visited[start] = true;
        q.push(start);

        cout << "\nBFS Traversal starting from vertex " << start << ": ";
        while(!q.empty()) {
            int v = q.front();
            q.pop();
            cout << v << " ";

            for(int u : adj[v]) {
                if(!visited[u]) {
                    visited[u] = true;
                    q.push(u);
                }
            }
        }
        cout << endl;
    }
};

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

    Graph g(V);
    cout << "Enter edges (u v):\n";
```

```
    for(int i = 0; i < E; i++) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }

    int start;
    cout << "Enter starting vertex for traversal: ";
    cin >> start;

    g.DFS(start);
    g.BFS(start);

    return 0;
}
```

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

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

void printBoard(vector<int>& board, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(board[i] == j)
                cout << "Q ";
            else
                cout << ". ";
        }
```

```
        cout << endl;
    }
    cout << "------------------\n";
}

bool isSafe(vector<int>& board, int row, int col) {
    for(int i = 0; i < row; i++) {
        if(board[i] == col || abs(board[i] - col) == abs(i - row))
            return false;
    }
    return true;
}

void solveNQueens(vector<int>& board, int row, int n, int& count) {
    if(row == n) {
        count++;
        printBoard(board, n);
        return;
    }

    for(int col = 0; col < n; col++) {
        if(isSafe(board, row, col)) {
            board[row] = col;
            solveNQueens(board, row + 1, n, count);
        }
    }
}

int main() {
```

```
    int n;
    cout << "Enter value of N (size of board): ";
    cin >> n;

    vector<int> board(n, -1);
    int count = 0;

    cout << "\nAll possible solutions for " << n << "-Queen problem:\n";
    solveNQueens(board, 0, n, count);

    cout << "\nTotal solutions found: " << count << endl;
    return 0;
}
```

21: Write a program for Graph Coloring using backtracking.

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

bool isSafe(int v, vector<vector<int> >& graph, vector<int>& color, int c, int V) {
    for(int i = 0; i < V; i++) {
        if(graph[v][i] && color[i] == c)
            return false;
    }
    return true;
}

bool graphColoringUtil(vector<vector<int> >& graph, int m, vector<int>& color, int v, int V) {
    if(v == V)
        return true;
```

```
   for(int c = 1; c <= m; c++) {
      if(isSafe(v, graph, color, c, V)) {
         color[v] = c;
         if(graphColoringUtil(graph, m, color, v + 1, V))
            return true;
         color[v] = 0; // backtrack
      }
   }
   return false;
}

bool graphColoring(vector<vector<int> >& graph, int m, int V) {
   vector<int> color(V, 0);
   if(graphColoringUtil(graph, m, color, 0, V)) {
      cout << "\nSolution Exists: Following are the assigned colors:\n";
      for(int i = 0; i < V; i++)
         cout << "Vertex " << i << " ---> Color " << color[i] << endl;
      return true;
   } else {
      cout << "\nNo solution exists with " << m << " colors.\n";
      return false;
   }
}

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

```
    vector<vector<int> > graph(V, vector<int>(V));
    cout << "Enter adjacency matrix:\n";
    for(int i = 0; i < V; i++)
        for(int j = 0; j < V; j++)
            cin >> graph[i][j];

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

    graphColoring(graph, m, V);

    return 0;
}
```