**PRACTICAL-1**

CODE:

#include <stdio.h>

int main()

{

  int k = 0, arra[10], i, j, n, arr2;

  printf("Enter of values in the array: \n");

  scanf("%d", &n);

  printf("enter %d array value(s): \n", n);

  for (i = 0; i < n; i++)

    scanf("%d", &arra[i]);

  /\* Insertion Sort  \*/

  for (i = 1; i < n; i++)

  {

    arr2 = arra[i];

    j = i - 1;

    while (j >= 0 && arra[j] > arr2)

    {

      k++;

      arra[j + 1] = arra[j];

      j = j - 1;

    }

    arra[j + 1] = arr2;

  }

  printf("Sorted  Array: \n");

  for (i = 0; i < n; i++)

    printf("%d  \n", arra[i]);

  printf(" swap= %d", k);

}

#include<stdio.h>

void main(){

  int k=0,n,i,j,temp,arr[100];

  printf("Enter the n of the arr: ");

  scanf("%d",&n);

  printf("Enter %d integer values: ",n);

  for(i=0; i<n; i++)

      scanf("%d",&arr[i]);

  for(i=0; i<n; i++){

      for(j=i+1; j<n; j++){

           if(arr[i] > arr[j])

       {

               temp=arr[i];

               arr[i]=arr[j];

               arr[j]=temp;

                k++;

           }

      }

  }

  printf("array after sorting is: ");

  for(i=0; i<n; i++)

      printf(" %d",arr[i]);

printf(" swap= %d",k);

}

**Output:**

**PRACTICAL-2**

#include <stdio.h>

int main()

{

  int k = 0, arra[10], i, j, n, arr2;

  printf("Enter of values in the array: \n");

  scanf("%d", &n);

  printf("enter %d array value(s): \n", n);

  for (i = 0; i < n; i++)

    scanf("%d", &arra[i]);

  /\* Insertion Sort  \*/

  for (i = 1; i < n; i++)

  {

    arr2 = arra[i];

    j = i - 1;

    while (j >= 0 && arra[j] > arr2)

    {

      k++;

      arra[j + 1] = arra[j];

      j = j - 1;

    }

    arra[j + 1] = arr2;

  }

  printf("Sorted  Array: \n");

  for (i = 0; i < n; i++)

    printf("%d  \n", arra[i]);

  printf(" swap= %d", k);

}

#include<stdio.h>

void main(){

  int k=0,n,i,j,temp,arr[100];

  printf("Enter the n of the arr: ");

  scanf("%d",&n);

  printf("Enter %d integer values: ",n);

  for(i=0; i<n; i++)

      scanf("%d",&arr[i]);

  for(i=0; i<n; i++){

      for(j=i+1; j<n; j++){

           if(arr[i] > arr[j])

       {

               temp=arr[i];

               arr[i]=arr[j];

               arr[j]=temp;

                k++;

           }

      }

  }

  printf("array after sorting is: ");

  for(i=0; i<n; i++)

      printf(" %d",arr[i]);

printf(" swap= %d",k);

}

**Practical-3(recurrence relatiom)**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

    float a, b, c;

    float n0,n1,tn0,tn1;

    cout<<"Enter the condition n and tn :";

    cin>>n0>>tn0;

    cout<<"Enter the condition n and tn :";

    cin>>n1>>tn1;

    cout << "Enter the values of tn: " << endl;

    cin >> a;

    cout << "Enter the values of tn-1: " << endl;

    cin >> b;

    cout << "Enter the values of tn-2: " << endl;

    cin >> c;

float root ,root1 ,root2;

    float d = b \* b - 4 \* a \* c;

    if (d == 0)

    {

         root = -b / (2 \* a);

        cout << "First root: " << root << endl;

        cout << "Second root: " << root << endl;

    }

    else if (d > 0)

    {

         root1 = (-b + sqrt(d)) / (2 \* a);

         root2 = (-b - sqrt(d)) / (2 \* a);

        cout << "First root: " << root1 << endl;

        cout << "Second root: " << root2 << endl;

    }

    else

    {

        float real = -b / (2 \* a);

        float imaginary = sqrt(-d) / (2 \* a);

        cout << "First root: " << real << "+" << imaginary << "i" << endl;

        cout << "Second root: " << real << "-" << imaginary << "i" << endl;

    }

    float c1temp,c2temp,c1temp2,c2temp2;

    int a1=1, b1=1, c1=tn0, a2=root1, b2=root2, c2=tn1;

    cout << "Your system of equations is the following:" << endl;

    cout << a1 << "x+" << b1 << "y=" << c1 << endl;

    cout << a2 << "x+" << b2 << "y=" << c2 << endl;

int res\_x,res\_y;

if ((a1 \* b2) - (b1 \* a2) == 0){

    cout << "The system has no solution." << endl;

}

else{

    res\_x = ((c1\*b2) - (b1\*c2))/((a1\*b2)-(b1\*a2));

    res\_y = ((a1\*c2) - (c1\*a2)) / ((a1\*b2) - (b1\*a2));

    cout << "c1=" << res\_x << " c2=" << res\_y << endl;

}

cout<<res\_x<<" "<<root1<<"^n"<<"+"<<res\_y<<" "<<root2<<"^n";

}

**Practical 4(merge and quick)**

**Merge sort:**

#include <stdio.h>

int count=0;

int swap=0;

void merge(int arr[], int p, int q, int r) {

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2) {

if (L[i] <= M[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = M[j];

j++;

}

k++;

count++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

count++;

swap++;

}

while (j < n2) {

arr[k] = M[j];

j++;

k++;

count++;

}

}

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);

}

}

void printArray(int arr[], int size) {

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

printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[100],n;

printf("Enter number of Students:");

scanf("%d",&n);

printf("Enter Students RollNo: \n");

for(int i=0; i<n; i++) scanf("%d",&arr[i]);

mergeSort(arr, 0, n - 1);

printf("Sorted Array Using Merge Sort : ");

printArray(arr, n);

printf("number of iteration:%d \n And \nSwapping: %d",count,swap);

}

**Quick sort:**

#include<stdio.h>

#include <conio.h>

int count=0;

int swa=0;

void swap(int \*a, int \*b) {

swa++;

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int array[], int low, int high) {

int pivot = array[high];

int i = (low - 1);

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

count++;

if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int array[], int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

void printArray(int array[], int size) {

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

printf("%d ", array[i]);

}

printf("\n");

}

int main() {

int arr[100],n;

printf("Enter number of Students:\n");

scanf("%d",&n);

printf("Enter Students RollNo:\n");

for(int i=0; i<n; i++) scanf("%d",&arr[i]);

quickSort(arr, 0, n - 1);

printf("Sorted Array Using Quick Sort\n");

printArray(arr, n);

printf("number of iteration:%d\n",count);

printf("number of Swapping:%d",swa);

}

**Practical-5(heapsort)**

#include <iostream>

using namespace std;

int count=0;

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) {

count++;

swap(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--) {

count++;

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int arr[] = { 82,90,10,12,15,77,65,23};

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

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

heapify(arr,n,i);

}

cout << "After heap sort array is \n";

printArray(arr, n);

heapSort(arr, n);

cout << "Sorted array in ascending order \n";

printArray(arr, n);

cout<<count;

return 0;

}

Practical -6(knapsack)

#include<stdio.h>

void main ()

{

int n, m, w[100], p[100], ratio[100] , i, j, u, temp;

float xr, x[100], total\_profit=0, total\_weight=0;

//Reading number of items

printf ("Enter the number of items(n): ");

scanf ("%d", &n);

//Reading the capacity of the knapsack

printf ("Enter the capacity of the Knapsack(m): ");

scanf ("%d", &m);

//Initializing remaining capacity of Knapsack (u)

u = m;

//Initializing Solution Array x[]

for(i=0;i<n;i++)

{

x[i]=0;

}

//Reading the Weights

printf ("Enter the Weights of items: ");

for (i = 0; i < n; i++)

{

printf ("\n\tWeight %d = ", i + 1);

scanf ("%d", &w[i]);

}

//Reading the Profit values

printf ("\nEnter the Profit Values : ");

for (i = 0; i < n; i++)

{

printf ("\n\tProfit %d = ", i + 1);

scanf ("%d", &p[i]);

}

//Calculating Pi/Wi ratio of each item and storing in array ratio[]

for (i = 0; i < n; i++)

{

ratio[i] = p[i] / w[i];

}

//Sorting all the arrays based on the ratio in descending order

for (i = 0; i < n; i++)

{

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

{

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

{

temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

temp = w[i];

w[i] = w[j];

w[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

//PRINTING THE SORTED TABLE

printf("\n The Table After Sorting : \n");

//Printing Item numbers

printf("\nItem:\t\t");

for(i=0;i<n;i++)

{

printf("%d\t",i+1);

}

//Printing Profit Array

printf("\nProfit:\t\t");

for(i=0;i<n;i++)

{

printf("%d\t",p[i]);

}

//Printing Weight Array

printf("\nWeights:\t");

for(i=0;i<n;i++)

{

printf("%d\t",w[i]);

}

//Printing RATIO Array

printf ("\nRATIO:\t\t");

for (i = 0; i < n; i++)

{

printf ("%d\t", ratio[i]);

}

//Calculating Solution Array x

for(i=0;i<n;i++)

{

if(w[i]<=u)

{

x[i]=1; //Setting solution index as 1

u=u-w[i]; //updating remaining knapsack capacity

}

else if(w[i]>u)

{

break;

}

}

if(i<=n)

{

xr = (float)u/w[i]; //Calculating what fraction of that item will fit into the knapsack

x[i] = xr; //Setting this fraction to solution array

}

//Printing Solution Array x

printf("\n X = [");

for(i=0;i<n;i++)

{

printf("%.3f , ",x[i]);

}

printf("]");

//Calculating Total Profit & Total Weight

for(i=0;i<n;i++)

{

total\_profit += x[i]\*p[i];

total\_weight += x[i]\*w[i];

}

//Displaying Total Profit and Total Weight

printf("\nTotal Profit = %.2f \n Total Weight = %.2f ",total\_profit,total\_weight);

}

**Pracctical -7(travelling salesman)**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_CITIES 10

// Global variables

int numCities;

int distance[MAX\_CITIES][MAX\_CITIES];

int visited[MAX\_CITIES];

int bestRoute[MAX\_CITIES];

int minDistance = INT\_MAX;

// Function to calculate the distance between two cities

int calculateDistance(int city1, int city2)

{

return distance[city1][city2];

}

// Recursive function to find the shortest route

void tsp(int currentCity, int depth, int currentDistance, int route[])

{

if (depth == numCities) {

// All cities visited, check if it forms a better route

if (currentDistance + calculateDistance(currentCity, 0) < minDistance) {

minDistance = currentDistance + calculateDistance(currentCity, 0);

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

bestRoute[i] = route[i];

}

}

return;

}

visited[currentCity] = 1;

route[depth] = currentCity;

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

if (!visited[i]) {

tsp(i, depth + 1, currentDistance + calculateDistance(currentCity, i), route);

}

}

visited[currentCity] = 0;

}

// Function to solve the TSP

void solveTSP()

{

int route[MAX\_CITIES];

tsp(1, 1, 0, route); // Start from city 1

printf("Shortest possible route: ");

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

printf("%d ", bestRoute[i]);

}

printf("\n");

printf("Shortest distance: %d\n", minDistance);

}

// Main function

int main()

{

// Get the number of cities from the user

printf("Enter the number of cities: ");

scanf("%d", &numCities);

// Get the distances between cities from the user

printf("Enter the distance matrix:\n");

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

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

scanf("%d", &distance[i][j]);

}

}

solveTSP();

return 0;

}

**Practical-8(Floyd warshal shortest distance)**

**#include <iostream>**

**#include <vector>**

**#define INF 99999**

**using namespace std;**

**int main()**

**{**

**int dist[4][4] = {**

**{0, INF, 2, INF},**

**{4, 0, 3, INF},**

**{INF, INF, 0, 2},**

**{INF, 1, INF, 0}**

**};**

**cout<<"Input matrix:"<<endl;**

**cout<<"0\tINF\t2\tINF"<<endl;**

**cout<<"4\t0\t3\tINF"<<endl;**

**cout<<"INF\tINF\t0\t2"<<endl;**

**cout<<"INF\t1\tINF\t0"<<endl<<endl;**

**int n = 4;**

**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][k] + dist[k][j] < dist[i][j])**

**dist[i][j] = dist[i][k] + dist[k][j];**

**}**

**}**

**}**

**cout << "Shortest 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] == INF)**

**cout << "INF\t";**

**else**

**cout << dist[i][j] << "\t";**

**}**

**cout << endl;**

**}**

**return 0;**

**}**

**Practical 9(graph coloring)**

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#define V 6 // Number of vertices (states and territories in Australia)

enum Color {

RED,

GREEN,

BLUE

};

void printSolution(enum Color color[])

{

printf("Coloring scheme of Australia :\n");

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

char state[20];

switch (i) {

case 0:

strcpy(state, "\nWestern Australia");

break;

case 1:

strcpy(state, "\nQueensland");

break;

case 2:

strcpy(state, "\nNorthern Territory");

break;

case 3:

strcpy(state, "\nSouth Australia");

break;

case 4:

strcpy(state, "\nNew South Wales");

break;

case 5:

strcpy(state, "\nVictoria");

break;

}

printf("%s: ", state);

switch (color[i]) {

case RED:

printf("Red\n");

break;

case GREEN:

printf("Green\n");

break;

case BLUE:

printf("Blue\n");

break;

}

}

}

bool isSafe(int v, int graph[V][V], enum Color color[], enum Color c)

{

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

if (graph[v][i] && c == color[i])

return false;

}

return true;

}

bool graphColorUtil(int graph[V][V], int m, enum Color color[], int v)

{

if (v == V)

return true;

for (int c = RED; c <= m; c++) {

if (isSafe(v, graph, color, c)) {

color[v] = c;

if (graphColorUtil(graph, m, color, v + 1))

return true;

color[v] = 0;

}

}

return false;

}

bool graphColoring(int graph[V][V], int m)

{

enum Color color[V];

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

color[i] = RED;

if (!graphColorUtil(graph, m, color, 0)) {

printf("No solution exists.\n");

return false;

}

printSolution(color);

return true;

}

int main()

{

int graph[V][V] = {

{0, 1, 0, 0, 1, 1}, // 0 - Western Australia

{1, 0, 0, 0, 0, 1}, // 1 - Queensland

{0, 0, 0, 0, 0, 1}, // 2 - Northern Territory

{0, 0, 0, 0, 0, 1}, // 3 - South Australia

{1, 0, 0, 0, 0, 1}, // 4 - New South Wales

{1, 1, 1, 1, 1, 0} // 5 - Victoria

};

int m = BLUE; // Maximum color to be used (enum Color)

graphColoring(graph, m);

return 0;

}

TYPE2(adjacency matrix)

Code:

#include<stdio.h>

#include<conio.h>

static int m, n;

static int c=0;

static int count=0;

int g[50][50];

int x[50];

void nextValue(int k);

void GraphColoring(int k);

void main()

{

int i, j;

int temp;

printf("\nEnter the number of nodes: " );

scanf("%d", &n);

/\*

printf("\nIf edge exists then enter 1 else enter 0 \n");

for(i=1; i<=n; i++)

{

x[i]=0;

for(j=1; j<=n; j++)

{

if(i==j)

g[i][j]=0;

else

{

printf("%d -> %d: " , i, j);

scanf("%d", &temp);

g[i][j]=g[j][i]=temp;

}

}

}

\*/

printf("\nEnter Adjacency Matrix:\n");

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

{

scanf("%d", &g[i][j]);

}

}

printf("\nPossible Solutions are\n");

for(m=1; m<=n; m++)

{

if(c==1)

{

break;

}

GraphColoring(1);

}

printf("\nThe chromatic number is %d", m-1);

//in for loop, m gets incremented first and then the condition is checked

//so it is m minus 1

printf("\nThe total number of solutions is %d", count);

getch();

}

void GraphColoring(int k)

{

int i;

while(1)

{

nextValue(k);

if(x[k]==0)

{

return;

}

if(k==n)

{

c=1;

for(i=1; i<=n; i++)

{

printf("%d ", x[i]);

}

count++;

printf("\n");

}

else

GraphColoring(k+1);

}

}

void nextValue(int k)

{

int j;

while(1)

{

x[k]=(x[k]+1)%(m+1);

if(x[k]==0)

{

return;

}

for(j=1; j<=n; j++)

{

if(g[k][j]==1&&x[k]==x[j])

break;

}

if(j==(n+1))

{

return;

}

}

}

**Practical 10**

#include <bits/stdc++.h>

using namespace std;

int countt=0;

// A function to print a solution

void print(int board[][8]){

for(int i=0;i<8;i++){

for(int j=0;j<8;j++){

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

}

cout<<endl;

}

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

}

//Function to check whether a position is valid or not

bool isValid(int board[][8],int row,int col){

//loop to check horizontal positions

for(int i=col;i>=0;i--){

if(board[row][i])

return false;

}

int i=row,j=col;

//loop to check the upper left diagonal

while(i>=0&&j>=0){

if(board[i][j])

return false;

i--;

j--;

}

i=row;

j=col;

//loop to check the lower left diagonal

while(i<8&&j>=0){

if(board[i][j])

return false;

i++;

j--;

}

return true;

}

//function to check all the possible solutions

void ninjaQueens(int board[][8],int currentColumn){

if(currentColumn>=8)

return;

//loop to cover all the columns

for(int i=0;i<8;i++){

if(isValid(board,i,currentColumn)){

board[i][currentColumn]=1;

if(currentColumn==7){

print(board);

countt++;

}

//recursively calling the function

ninjaQueens(board,currentColumn+1);

//backtracking

board[i][currentColumn]=0;

}

}

}

int main() {

//initial board situation

int board[8][8]={{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0}};

ninjaQueens(board,0);

/\* In total, 92 solutions exist for 8x8 board. This statement will verify our code\*/

cout<<countt<<endl;

return 0;

}