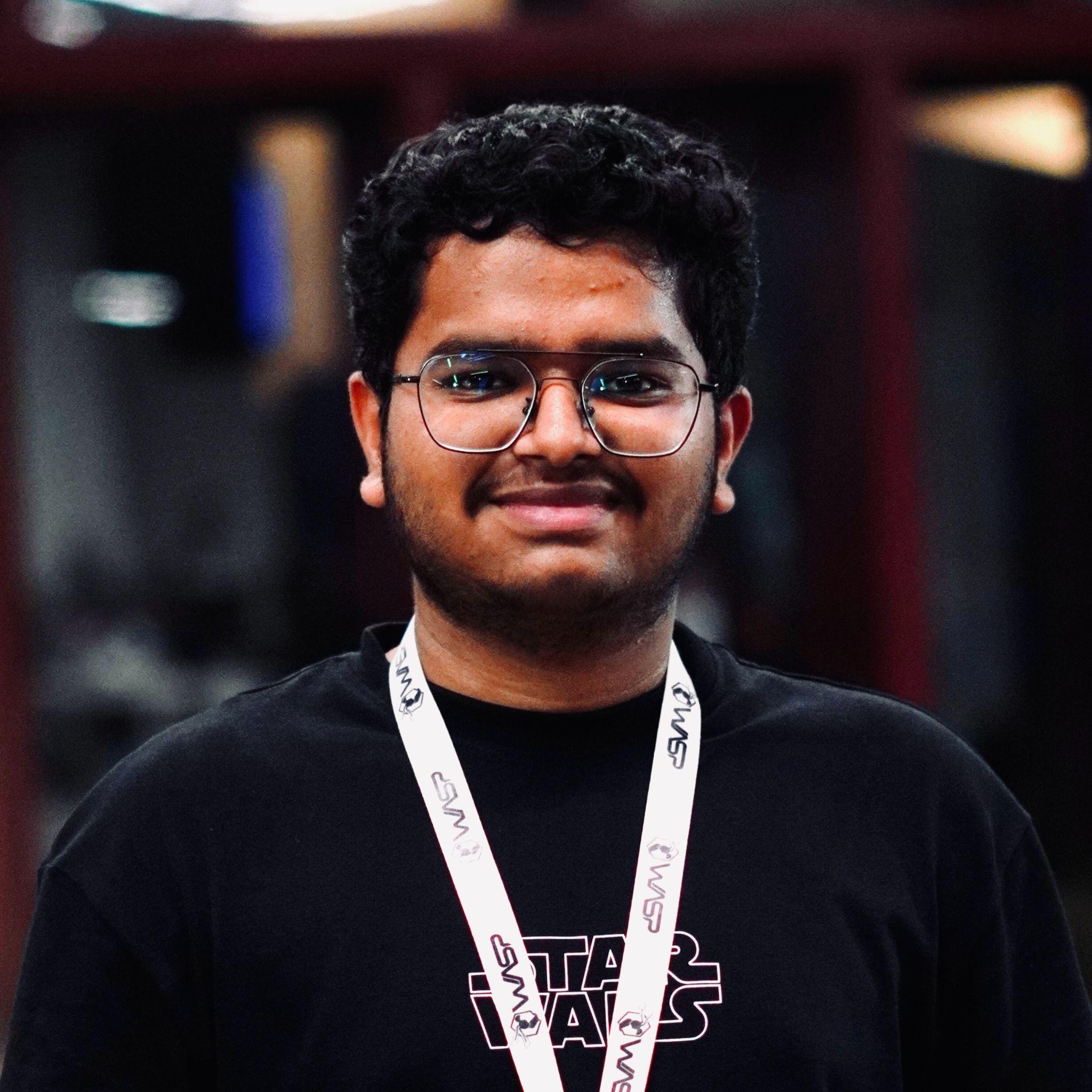
Name: Dhruv Mittal

Roll no: 1024030364

Subgroup: 2C24



Q-1 : /\*

1) Implement the binary search algorithm, regarded as a fast search algorithm

with run-time complexity of Ο(log n) in comparison to the Linear Search.

\*/

#include <iostream>

using namespace std;

int binary\_Search(int arr[] , int size , int target){

int low = 0;

int high = size-1 ;

while(low <= high){

int mid = low + (high-low) / 2 ;

if(arr[mid] == target){

return mid ;

}

else if(target < arr[mid]){

high = mid - 1;

}

else{

low = mid + 1 ;

}

}

return -1 ;

}

int main(){

int n ;

cout << "Enter the size of the sorted array : " ;

cin >> n ;

int arr[n];

cout << "Enter the sorted array : " ;

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

cin >> arr[i] ;

}

int target;

cout << "Enter the target : " ;

cin >> target ;

int index = binary\_Search(arr , n , target) ;

if(index != -1){

cout << "Target is at index : " << index ;

}

else{

cout << "Target NOT FOUND!" ;

}

return 0 ;

}

Q-2 : /\*

2) Bubble Sort is the simplest sorting algorithm that works by repeatedly

swapping the adjacent elements if they are in the wrong order. Code the Bubble sort

with the following elements: 64 34 25 12 22 11 90

\*/

#include <iostream>

using namespace std ;

void bubble\_Sort(int arr[] , int n){

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

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

if(arr[j] > arr[j+1]){

swap(arr[j] , arr[j+1]) ;

}

}

}

}

int main(){

int arr[7] = {64 , 34 , 25 , 12 , 22 , 11 , 90} ;

int n=7 ;

bubble\_Sort(arr , n) ;

cout << "Sorted Array : " ;

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

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

}

return 0 ;

}

Q-3 a: /\*

Design the Logic to Find a Missing Number in a Sorted Array. Given an array of

n-1 distinct integers in the range of 1 to n, find the missing number in it in a Sorted Array:

(a) Linear time

\*/

#include <iostream>

using namespace std ;

int linearsearch(int arr[], int n){

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

if(arr[i] != i+1){

return i+1;

}

}

return n;

}

int main() {

int arr[6] = {1,2,3,4,5,7};

int result = linearsearch(arr, 6);

cout << "The missing number is " << result;

return 0;

}

Q-3b : #include <iostream>

using namespace std;

int binarysearch(int arr[], int n) {

int start = 0, end = n - 2;

while (start <= end) {

int mid = start + (end - start) / 2;

if (arr[mid] == mid + 1) {

start = mid + 1;

} else {

end = mid - 1;

}

}

return start + 1;

}

int main() {

int arr[] = {1, 2, 3, 4, 5, 7};

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

int result = binarysearch(arr, n);

cout << "The missing number is : " << result << endl;

return 0;

}

Q-4a : /\*

4) String Related Programs

(a) Write a program to concatenate one string to another string.

\*/

#include<iostream>

#include<string>

using namespace std;

int main() {

string str1, str2;

cout << "Enter first string: ";

getline(cin, str1);

cout << "Enter second string: ";

getline(cin, str2);

str1 += str2; // Concatenate str2 to str1

cout << "Concatenated string: " << str1 << endl;

return 0;

}

Q-4b : //(b) Write a program to reverse a string.

#include <iostream>

#include <string>

using namespace std;

int main() {

string str;

cout << "Enter a string: ";

getline(cin, str);

int n = str.length();

for (int i = 0; i < n / 2; i++) {

char temp = str[i];

str[i] = str[n - i - 1];

str[n - i - 1] = temp;

}

cout << "Reversed string: " << str << endl;

return 0;

}

Q-4c : // (c) Write a program to delete all the vowels from the string.

#include<iostream>

#include<string>

using namespace std;

bool isVowel (char ch)

{

ch = tolower(ch);

return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');

}

int main() {

string str;

cout << "Enter a string: ";

getline(cin, str);

string result = "";

for (char ch : str) {

if (!isVowel(ch)) {

result += ch;

}

}

cout << "String after removing vowels: " << result << endl;

return 0;

}

Q-4d : // (d) Write a program to sort the strings in alphabetical order.

#include<iostream>

#include<string>

#include<algorithm> // For sort function

using namespace std;

int main() {

int n;

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

cin >> n;

cin.ignore(); // To ignore the newline character after reading n

string arr[100];

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

{

cout << "Enter string " << i + 1 << ": ";

getline(cin, arr[i]);

}

sort(arr, arr + n); // Sort the array of strings

cout << "Sorted strings: " << endl;

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

{

cout << arr[i] << endl;

}

return 0;

}

Q-4e : //(e) Write a program to convert a character from uppercase to lowercase.

#include <iostream>

using namespace std;

int main() {

char ch;

cout << "Enter an uppercase character: ";

cin >> ch;

if (ch >= 'A' && ch <= 'Z') {

ch = ch + 32; // ASCII conversion

cout << "Lowercase character: " << ch << endl;

} else {

cout << "Not an uppercase character!" << endl;

}

return 0;

}

Q-5 a : /\*

5) Space required to store any two-dimensional array is 𝑛𝑢𝑚𝑏𝑒𝑟 𝑜ƒ 𝑟𝑜𝑤𝑠 × 𝑛𝑢𝑚𝑏𝑒𝑟 𝑜ƒ

𝑐𝑜𝑙𝑢𝑚𝑛𝑠. Assuming an array is used to store elements of the following matrices,

implement an efficient way that reduces the space requirement.

(a) Diagonal Matrix.

\*/

#include <iostream>

using namespace std;

class DiagonalMatrix {

int \*arr, n;

public:

DiagonalMatrix(int size) {

n = size;

arr = new int[n]{0};

}

void set(int i, int j, int val) {

if (i == j) arr[i-1] = val;

}

int get(int i, int j) {

return (i == j) ? arr[i-1] : 0;

}

void display() {

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

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

cout << get(i,j) << " ";

cout << endl;

}

}

~DiagonalMatrix() { delete[] arr; }

};

int main() {

int n; cin >> n;

DiagonalMatrix d(n);

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

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

int val; cin >> val;

d.set(i,j,val);

}

d.display();

}

Q-5b : // (b) Tri-diagonal Matrix.

#include <iostream>

using namespace std;

class TriDiagonal {

int a[7]; // 3n - 2 = 7

public:

void set(int i,int j,int x){

if(i-j==1) a[i-1]=x;

else if(i==j) a[2+i]=x;

else if(j-i==1) a[5+i]=x;

}

int get(int i,int j){

if(i-j==1) return a[i-1];

else if(i==j) return a[2+i];

else if(j-i==1) return a[5+i];

else return 0;

}

void display(){

for(int i=0;i<3;i++){

for(int j=0;j<3;j++) cout<<get(i,j)<<" ";

cout<<endl;

}

}

};

int main(){

TriDiagonal t;

t.set(0,0,1); t.set(1,1,5); t.set(2,2,9);

t.set(0,1,2); t.set(1,2,6);

t.set(1,0,3); t.set(2,1,7);

t.display();

return 0 ;

}

Q-5c : // (c) Lower triangular Matrix.

#include <iostream>

using namespace std;

class LowerTriangular {

int \*arr, n;

public:

LowerTriangular(int size) {

n = size;

arr = new int[n\*(n+1)/2]{0};

}

void set(int i,int j,int val) {

if (i>=j) arr[(i\*(i-1))/2 + (j-1)] = val;

}

int get(int i,int j) {

if (i>=j) return arr[(i\*(i-1))/2 + (j-1)];

return 0;

}

void display() {

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

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

cout<<get(i,j)<<" ";

cout<<endl;

}

}

~LowerTriangular(){ delete[] arr; }

};

int main() {

int n; cin>>n;

LowerTriangular lt(n);

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

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

int val; cin>>val;

lt.set(i,j,val);

}

lt.display();

}

Q-5d : // (d) Upper triangular Matrix.

#include <iostream>

using namespace std;

class UpperTriangular {

int \*arr, n;

public:

UpperTriangular(int size) {

n = size;

arr = new int[n\*(n+1)/2]{0};

}

void set(int i,int j,int val) {

if (i<=j) arr[(i-1)\*n - (i-1)\*(i-2)/2 + (j-i)] = val;

}

int get(int i,int j) {

if (i<=j) return arr[(i-1)\*n - (i-1)\*(i-2)/2 + (j-i)];

return 0;

}

void display() {

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

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

cout<<get(i,j)<<" ";

cout<<endl;

}

}

~UpperTriangular(){ delete[] arr; }

};

int main() {

int n; cin>>n;

UpperTriangular ut(n);

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

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

int val; cin>>val;

ut.set(i,j,val);

}

ut.display();

}

Q-5e : // (e) Symmetric Matrix

#include <iostream>

using namespace std;

class SymmetricMatrix {

int \*arr, n;

public:

SymmetricMatrix(int size) {

n = size;

arr = new int[n\*(n+1)/2]{0};

}

void set(int i,int j,int val) {

if (i>=j) arr[(i\*(i-1))/2 + (j-1)] = val;

else arr[(j\*(j-1))/2 + (i-1)] = val; // since A[i][j]=A[j][i]

}

int get(int i,int j) {

if (i>=j) return arr[(i\*(i-1))/2 + (j-1)];

return arr[(j\*(j-1))/2 + (i-1)];

}

void display() {

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

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

cout<<get(i,j)<<" ";

cout<<endl;

}

}

~SymmetricMatrix(){ delete[] arr; }

};

int main() {

int n; cin>>n;

SymmetricMatrix sm(n);

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

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

int val; cin>>val;

sm.set(i,j,val);

}

sm.display();

}

Q-6 a : /\* 6) Write a program to implement the following operations on a Sparse Matrix,

assuming the matrix is represented using a triplet.

(a) Transpose of a matrix.

\*/

#include <iostream>

using namespace std;

#define MAX 100

void transpose(int a[][3], int t[][3]) {

int rows = a[0][0], cols = a[0][1], terms = a[0][2];

t[0][0] = cols;

t[0][1] = rows;

t[0][2] = terms;

if (terms > 0) {

int k = 1;

for (int c = 0; c < cols; c++) {

for (int i = 1; i <= terms; i++) {

if (a[i][1] == c) {

t[k][0] = a[i][1]; // new row = old col

t[k][1] = a[i][0]; // new col = old row

t[k][2] = a[i][2]; // value same

k++;

}

}

}

}

}

int main() {

int a[MAX][3], t[MAX][3];

int rows, cols, terms;

cout << "Enter rows, cols, non-zero terms: ";

cin >> rows >> cols >> terms;

a[0][0] = rows;

a[0][1] = cols;

a[0][2] = terms;

cout << "Enter triplet (row col value):\n";

for (int i = 1; i <= terms; i++)

cin >> a[i][0] >> a[i][1] >> a[i][2];

transpose(a, t);

cout << "\nTranspose in Triplet Form:\n";

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

cout << t[i][0] << " " << t[i][1] << " " << t[i][2] << endl;

return 0;

}

Q-6b : // (b) Addition of two matrices.

#include <iostream>

using namespace std;

#define MAX 100

void addSparse(int a[][3], int b[][3], int c[][3]) {

if (a[0][0] != b[0][0] || a[0][1] != b[0][1]) {

cout << "Addition not possible\n";

return;

}

int i=1, j=1, k=1;

c[0][0] = a[0][0];

c[0][1] = a[0][1];

while (i <= a[0][2] && j <= b[0][2]) {

if (a[i][0] < b[j][0] || (a[i][0] == b[j][0] && a[i][1] < b[j][1])) {

c[k][0] = a[i][0]; c[k][1] = a[i][1]; c[k][2] = a[i][2];

i++; k++;

}

else if (b[j][0] < a[i][0] || (a[i][0] == b[j][0] && b[j][1] < a[i][1])) {

c[k][0] = b[j][0]; c[k][1] = b[j][1]; c[k][2] = b[j][2];

j++; k++;

}

else { // same position

int sum = a[i][2] + b[j][2];

if (sum != 0) {

c[k][0] = a[i][0]; c[k][1] = a[i][1]; c[k][2] = sum;

k++;

}

i++; j++;

}

}

while (i <= a[0][2]) { c[k][0]=a[i][0]; c[k][1]=a[i][1]; c[k][2]=a[i][2]; i++; k++; }

while (j <= b[0][2]) { c[k][0]=b[j][0]; c[k][1]=b[j][1]; c[k][2]=b[j][2]; j++; k++; }

c[0][2] = k-1;

}

int main() {

int a[MAX][3], b[MAX][3], c[MAX][3];

int termsA, termsB;

cout << "Enter rows cols and non-zeros for A: ";

cin >> a[0][0] >> a[0][1] >> termsA;

a[0][2] = termsA;

cout << "Enter A (row col val):\n";

for (int i=1; i<=termsA; i++) cin >> a[i][0] >> a[i][1] >> a[i][2];

cout << "Enter rows cols and non-zeros for B: ";

cin >> b[0][0] >> b[0][1] >> termsB;

b[0][2] = termsB;

cout << "Enter B (row col val):\n";

for (int i=1; i<=termsB; i++) cin >> b[i][0] >> b[i][1] >> b[i][2];

addSparse(a, b, c);

cout << "\nResultant Matrix in Triplet Form:\n";

for (int i=0; i<=c[0][2]; i++)

cout << c[i][0] << " " << c[i][1] << " " << c[i][2] << endl;

return 0;

}

Q-6c : // (c) Multiplication of two matrices.

#include <iostream>

using namespace std;

struct Term {

int row, col, val;

};

void multiply(Term A[], Term B[], Term C[]) {

if (A[0].col != B[0].row) {

cout << "Multiplication not possible\n";

return;

}

int m = A[0].row, n = A[0].col, p = B[0].col;

int aTerms = A[0].val, bTerms = B[0].val;

int k = 1;

for (int i = 1; i <= aTerms; i++) {

for (int j = 1; j <= bTerms; j++) {

if (A[i].col == B[j].row) {

int r = A[i].row;

int c = B[j].col;

int v = A[i].val \* B[j].val;

// check if (r,c) already exists in result

int found = 0;

for (int x = 1; x < k; x++) {

if (C[x].row == r && C[x].col == c) {

C[x].val += v;

found = 1;

break;

}

}

if (!found) {

C[k].row = r;

C[k].col = c;

C[k].val = v;

k++;

}

}

}

}

C[0].row = m;

C[0].col = p;

C[0].val = k - 1;

}

int main() {

Term A[50], B[50], C[100];

int aTerms, bTerms;

cout << "Enter rows cols nonzeros for A: ";

cin >> A[0].row >> A[0].col >> aTerms;

A[0].val = aTerms;

cout << "Enter A (row col val):\n";

for (int i = 1; i <= aTerms; i++) cin >> A[i].row >> A[i].col >> A[i].val;

cout << "Enter rows cols nonzeros for B: ";

cin >> B[0].row >> B[0].col >> bTerms;

B[0].val = bTerms;

cout << "Enter B (row col val):\n";

for (int i = 1; i <= bTerms; i++) cin >> B[i].row >> B[i].col >> B[i].val;

multiply(A, B, C);

cout << "\nResult in Triplet Form:\n";

for (int i = 0; i <= C[0].val; i++)

cout << C[i].row << " " << C[i].col << " " << C[i].val << endl;

return 0;

}

Q-7 : /\*

7) Let A[1 …. n] be an array of n real numbers. A pair (A[i], A[j ]) is said to be an

inversion if these numbers are out of order, i.e., i < j but A[i]>A[j ]. Write a program to

count the number of inversions in an array.

\*/

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter size of array: ";

cin >> n;

int A[100];

cout << "Enter elements: ";

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

cin >> A[i];

int count = 0;

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

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

if (A[i] > A[j])

count++;

}

}

cout << "Number of inversions: " << count << endl;

return 0;

}

Q-8 : /\*

8) Write a program to count the total number of distinct elements in an array of

length n.

\*/

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter size of array: ";

cin >> n;

int A[100];

cout << "Enter elements: ";

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

cin >> A[i];

int count = 0;

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

bool isDistinct = true;

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

if (A[i] == A[j]) {

isDistinct = false;

break;

}

}

if (isDistinct) count++;

}

cout << "Total distinct elements: " << count << endl;

return 0;

}