Name:- Aryan Juneja roll no 1024030920 batch :- 2C64

Q1

#include <iostream>

#include <algorithm>

using namespace std;

int binarySearch(int arr[], int n, int target) {

int low = 0, high = n - 1;

while (low <= high) {

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

if (arr[mid] == target) {

return mid;

}

else if (arr[mid] < target) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() {

int arr[] = {2, 4, 7, 10, 15, 20, 25};

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

sort (arr,arr+n);

int target;

cout << "Enter element to search: ";

cin >> target;

int result = binarySearch(arr, n, target);

if (result != -1)

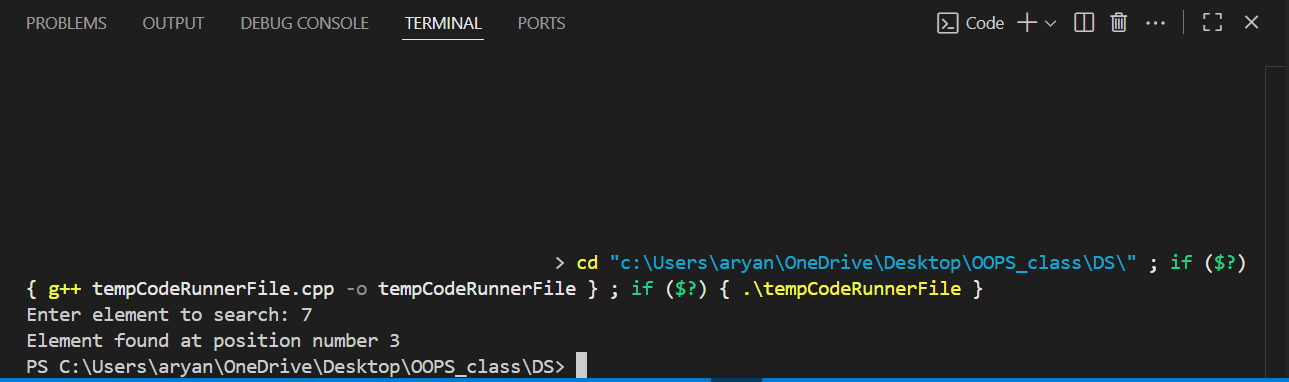
cout << "Element found at position number " << result +1 << endl;

else

cout << "Element not found." << endl;

return 0;

}



Q2

#include<bits/stdc++.h>

using namespace std;

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

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

{

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

{

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

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

}

}

}

return ;

}

int main(){

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

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

bubble\_sort(arr,n);

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

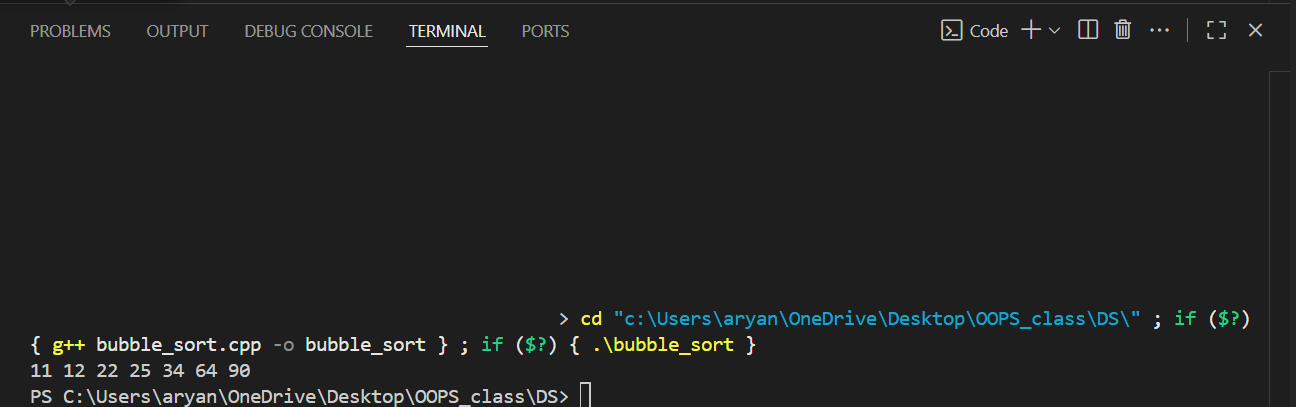
{

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

}

return 0;

}



Q3

#include <iostream>

#include <algorithm>

using namespace std;

int linear\_search(int arr[],int n){

int i=1;

while(i==arr[i-1]){

i++;

}

cout<<i;

return i;

}

int binary\_search(int arr[],int n){

int low=0,high= n-1,x;

while(low<=high){

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

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

low=mid+1;

}

else{

x=mid +1;

high=mid-1;

}

}

return x;

}

int main(){

int arr[]={ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 11 , 12 ,13 , 14 , 15 , 16 , 17 , 18 , 19 , 20 , 21 , 22 ,23, 24 , 25 };

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

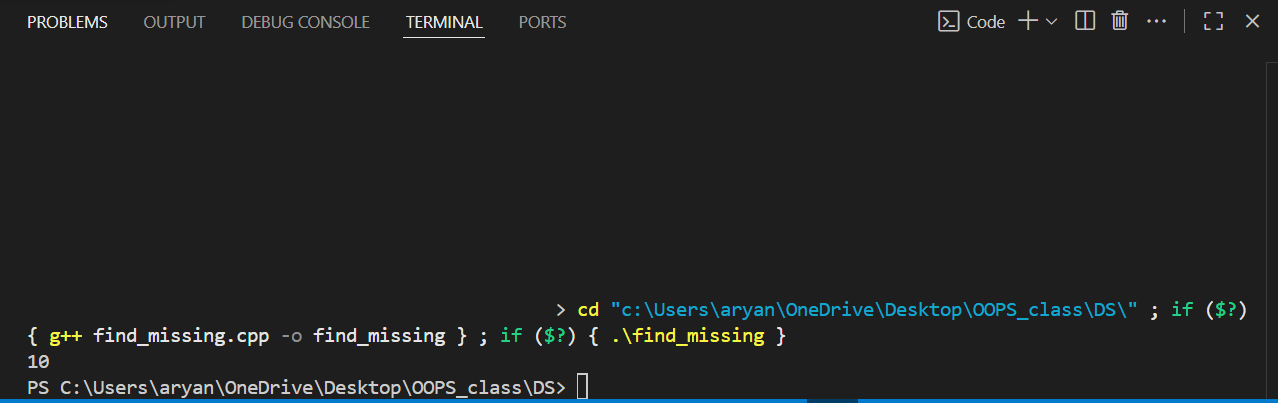
// int a=linear\_search(arr,n);

int a= binary\_search(arr,n);

cout<<a;

return 0;

}



Q4

#include <bits/stdc++.h>

using namespace std;

void concatenate(string s1,string s2){

string s3 = s1+s2;

cout<<s3<<endl;

return;

}

void reversestr(string s1){

reverse (s1.begin(),s1.end());

cout<<s1<<endl;

}

bool is\_vowel(char s1){

s1 = tolower(s1);

if(s1=='a'||s1=='e'||s1=='i'||s1=='o'||s1=='u'){

return true;

}

else{return false;}

}

void delete\_vowels(string s1){

int n=s1.length();

string str;

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

{

if(!is\_vowel){

str+=s1[i];

}

}

cout<<str;

return;

}

void sort(string s1){

int n=s1.length();

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

{

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

{

if(s1[j]>s1[j+1]){

swap(s1[j+1],s1[j]);

}

}

}

cout<<s1<<endl;

}

void to\_lower(string str){

for (int i = 0; i < str.length(); i++)

{

if (str[i]<91) {

str[i]=(char) (str[i]+32);

}

}

cout << str << endl;

return;

}

int main() {

string s1 = "NAME";

string s2 = "name";

concatenate(s1,s2);

reversestr(s1);

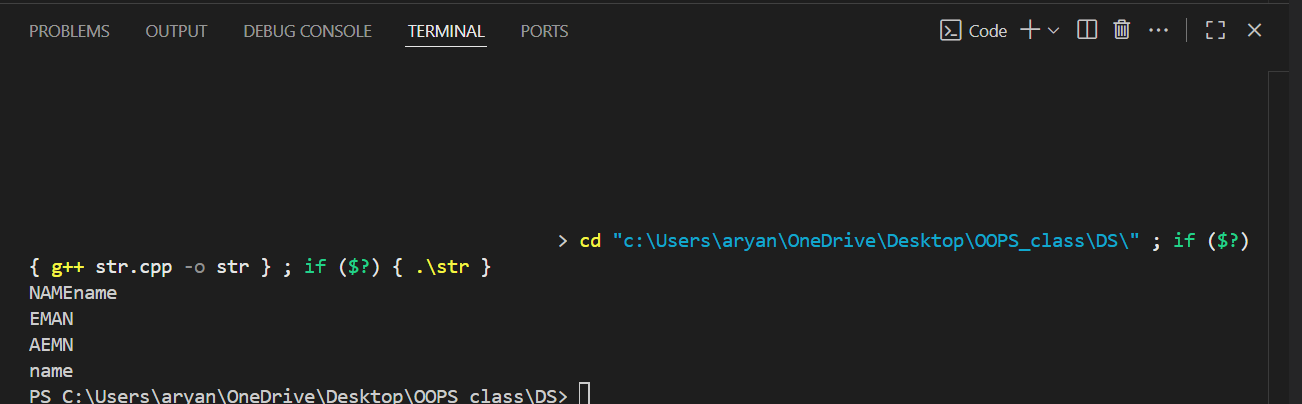
delete\_vowels(s1);

sort(s1);

to\_lower(s1);

return 0;

}



Q5

A

#include <iostream>

using namespace std;

int main()

{

int matrix[3][3]= {{20,0,0},

{0,30,0},

{0,0,40}

};

int size=sizeof(matrix[0])/sizeof(matrix[0][0]);

int diag[size];

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

diag[i]=matrix[i][i];

}

return 0;

}

B

#include <iostream>

using namespace std;

void Display(int\* arr, int n) {

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

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

}

cout<<endl;

}

int main()

{

int matrix[4][4]={{10,2,0,0},{100,20,4,0},{0,200,30,6},{0,0,300,40}};

int size=sizeof(matrix[0])/sizeof(matrix[0][0]);

int maindiag[size];

int subdiag[size-1];

int supdiag[size-1];

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

maindiag[i]=matrix[i][i];

if(i<size-1){

supdiag[i]=matrix[i][i+1];

}

if(i>0){

subdiag[i-1]=matrix[i][i-1];

}

}

Display(maindiag,size);

Display(supdiag,size-1);

Display(subdiag,size-1);

return 0;

}

C

#include <iostream>

using namespace std;

void Display(int\* arr, int n) {

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

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

}

}

int main()

{

int mat[4][4]={{1,0,0,0},{2,3,0,0},{4,5,6,0},{7,8,9,10}};

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

int totele= (n\*(n+1))/2;

int ltm[totele];

int k=0;

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

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

ltm[k]=mat[i][j];

k++;

}

}

Display(ltm,totele);

return 0;

}

D

#include <iostream>

using namespace std;

void Display(int\* arr, int n) {

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

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

}

}

int main()

{

int mat[4][4]={{1,2,3,4},{0,5,6,7},{0,0,8,9},{0,0,0,10}};

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

int totele= (n\*(n+1))/2;

int utm[totele];

int k=0;

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

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

utm[k]=mat[i][j];

k++;

}

}

Display(utm,totele);

return 0;

}

E

#include <iostream>

using namespace std;

void Display(int\* arr, int n) {

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

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

}

}

int main()

{

int mat[4][4]={{1,2,3,4},{2,5,6,7},{3,6,8,9},{4,7,9,10}};

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

int totele= (n\*(n+1))/2;

int sm[totele];

int k=0;

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

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

sm[k]=mat[i][j];

k++;

}

}

Display(sm,totele);

return 0;

}

6

#include <bits/stdc++.h>

using namespace std;

struct Element {

int row, col, val;

};

struct SparseMatrix {

int rows, cols;

vector<Element> data;

// Function to add a non-zero element

void addElement(int r, int c, int v) {

if (v != 0) data.push\_back({r, c, v});

}

// Display the triplet form

void display() {

cout << "Row Col Val\n";

for (auto &e : data) {

cout << e.row << " " << e.col << " " << e.val << "\n";

}

}

};

// (a) Transpose

SparseMatrix transpose(SparseMatrix &A) {

SparseMatrix res;

res.rows = A.cols;

res.cols = A.rows;

for (auto &e : A.data) {

res.addElement(e.col, e.row, e.val);

}

return res;

}

// (b) Addition of two sparse matrices

SparseMatrix add(SparseMatrix &A, SparseMatrix &B) {

if (A.rows != B.rows || A.cols != B.cols) {

throw invalid\_argument("Matrix dimensions do not match for addition");

}

SparseMatrix res;

res.rows = A.rows;

res.cols = A.cols;

int i = 0, j = 0;

while (i < A.data.size() && j < B.data.size()) {

if (A.data[i].row < B.data[j].row ||

(A.data[i].row == B.data[j].row && A.data[i].col < B.data[j].col)) {

res.addElement(A.data[i].row, A.data[i].col, A.data[i].val);

i++;

} else if (B.data[j].row < A.data[i].row ||

(B.data[j].row == A.data[i].row && B.data[j].col < A.data[i].col)) {

res.addElement(B.data[j].row, B.data[j].col, B.data[j].val);

j++;

} else {

int sum = A.data[i].val + B.data[j].val;

if (sum != 0) res.addElement(A.data[i].row, A.data[i].col, sum);

i++, j++;

}

}

while (i < A.data.size()) res.addElement(A.data[i].row, A.data[i].col, A.data[i++].val);

while (j < B.data.size()) res.addElement(B.data[j].row, B.data[j].col, B.data[j++].val);

return res;

}

// (c) Multiplication

SparseMatrix multiply(SparseMatrix &A, SparseMatrix &B) {

if (A.cols != B.rows) {

throw invalid\_argument("Matrix dimensions do not match for multiplication");

}

SparseMatrix res;

res.rows = A.rows;

res.cols = B.cols;

// Convert B to map for fast access

map<pair<int,int>, int> Bmap;

for (auto &e : B.data) {

Bmap[{e.row, e.col}] = e.val;

}

// Multiply

for (auto &a : A.data) {

for (int j = 0; j < B.cols; j++) {

auto it = Bmap.find({a.col, j});

if (it != Bmap.end()) {

res.addElement(a.row, j, a.val \* it->second);

}

}

}

// Combine duplicate positions

map<pair<int,int>, int> combine;

for (auto &e : res.data) {

combine[{e.row, e.col}] += e.val;

}

res.data.clear();

for (auto &p : combine) {

if (p.second != 0) res.addElement(p.first.first, p.first.second, p.second);

}

return res;

}

// ================== MAIN ==================

int main() {

SparseMatrix A, B;

A.rows = 3; A.cols = 3;

B.rows = 3; B.cols = 3;

// Matrix A

A.addElement(0, 0, 1);

A.addElement(0, 2, 2);

A.addElement(1, 1, 3);

// Matrix B

B.addElement(0, 1, 4);

B.addElement(1, 1, 5);

B.addElement(2, 2, 6);

cout << "Matrix A in triplet form:\n"; A.display();

cout << "\nMatrix B in triplet form:\n"; B.display();

// Transpose

SparseMatrix T = transpose(A);

cout << "\nTranspose of A:\n"; T.display();

// Addition

SparseMatrix Sum = add(A, B);

cout << "\nA + B:\n"; Sum.display();

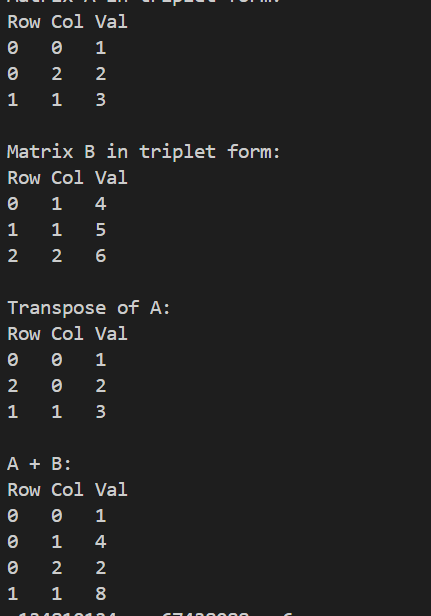
// Multiplication

SparseMatrix Prod = multiply(A, B);

cout <<"\nA \* B:\n"; Prod.display();

return 0;

}



7

#include <iostream>

#include <vector>

using namespace std;

int NoOfInversions(int \*arr, int n){

int cnt=0;

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

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

if(arr[j]<arr[i]){

cnt++;

}

}

}

return cnt;

}

int Merge(int \*arr, int low, int mid, int high){

vector<int>temp;

int left=low;

int right=mid+1;

int cnt=0;

while(left<=mid && right<=high){

if(arr[left]<=arr[right]){

temp.push\_back(arr[left]);

left++;

}

else{

temp.push\_back(arr[right]);

cnt += (mid-left+1);

right++;

}

}

while(left<=mid){

temp.push\_back(arr[left]);

left++;

}

while(right<=high){

temp.push\_back(arr[right]);

right++;

}

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

arr[i]=temp[i-low];

}

return cnt;

}

int NoOfInversionsUsingMergeSort(int \*arr,int low, int high){

int cnt=0;

if(low>=high){

return cnt;

}

int mid= (low+high)/2;

cnt += NoOfInversionsUsingMergeSort(arr,low,mid);

cnt += NoOfInversionsUsingMergeSort(arr,mid+1,high);

cnt += Merge(arr,low,mid,high);

return cnt;

}

int main()

{

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

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

int num=NoOfInversionsUsingMergeSort(A,0,n-1);

cout<<num;

return 0;

}

