**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**(Deemed to be University)**

**Patiala, Punjab**

****

**LAB ASSIGNMENT-2**

**Submitted By:**

**“Sajid Miya” 102367013**

**Subgroup: 2Q11**

**Submitted To:**

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**Department of Computer Science and Engineering**

**Patiala, Punjab, India**

**“23rd August 2024”**

Contents

[UCS301 Data Structures 1](#_Toc175275287)

[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. 1](#_Toc175275288)

[Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Code the Bubble sort with the following elements: 3](#_Toc175275289)

[Design the Logic to Find a Missing Number in a Sorted Array. 4](#_Toc175275290)

[String Related Programs 9](#_Toc175275291)

[Write a program to concatenate one string to another string. 9](#_Toc175275292)

[Write a program to reverse a string. 10](#_Toc175275293)

[Write a program to delete all the vowels from the string. 10](#_Toc175275294)

[Write a program to sort the strings in alphabetical order. 11](#_Toc175275295)

[Write a program to convert a character from uppercase to lowercase. 12](#_Toc175275296)

[Space required to store any two-dimensional array is 𝑛𝑢𝑚𝑏𝑒𝑟 𝑜ƒ 𝑟𝑜𝑤𝑠 × 𝑛𝑢𝑚𝑏𝑒𝑟 𝑜ƒ 13](#_Toc175275297)

[𝑐𝑜𝑙𝑢𝑚𝑛𝑠. Assuming array is used to store elements of the following matrices, implement an efficient way that reduces the space requirement. 13](#_Toc175275298)

[Diagonal Matrix. 13](#_Toc175275299)

[Tri-diagonal Matrix. 14](#_Toc175275300)

[Lower triangular Matrix. 17](#_Toc175275301)

[Upper triangular Matrix. 18](#_Toc175275302)

[Symmetric Matrix 20](#_Toc175275303)

[Write a program to implement the following operations on a Sparse Matrix, assuming the matrix is represented using a triplet. 21](#_Toc175275304)

[Transpose of a matrix. 21](#_Toc175275305)

[Addition of two matrices. 21](#_Toc175275306)

[Multiplication of two matrices. 21](#_Toc175275307)

[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. 26](#_Toc175275308)

[Write a program to count the total number of distinct elements in an array of length *n*. 27](#_Toc175275309)

[Additional Questions 29](#_Toc175275310)

[Write a program to find a saddle point in a two-dimensional array. A saddle point in a numerical array is a number that is larger than or equal to every number in its column, and smaller than or equal to every number in its row. 29](#_Toc175275311)

# UCS301 Data Structures

**Lab Assignment 2 (Week 2 and Week 3)**

# 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.

**Ans:**

#include <iostream>

#include <vector>

#include <algorithm> // For std::swap

using namespace std;

int search(const vector<int>& arr, int key)

{

int first = 0;

int last = arr.size() - 1;

while (first <= last)

{

int middle = (first + last) / 2;

if (arr[middle] == key)

{

return middle;

}

else if (arr[middle] < key)

{

first = middle + 1;

}

else

{

last = middle - 1;

}

}

return -1;

}

int main()

{

cout << "How many elements do you have?" << endl;

int size;

cin >> size;

if (size <= 0)

{

cout << "Invalid array size." << endl;

return EXIT\_FAILURE;

}

vector<int> arr(size);

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

{

cout << "Enter element " << i + 1 << ":" << endl;

cin >> arr[i];

}

// Bubble Sort

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

{

bool flag = false;

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

{

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

{

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

flag = true;

}

}

if (!flag)

{

break; // No swaps means the array is sorted

}

}

cout << "Enter the element that you want to search for:" << endl;

int key;

cin >> key;

int status = search(arr, key);

if (status != -1)

{

cout << key << " found at index " << status << endl;

}

else

{

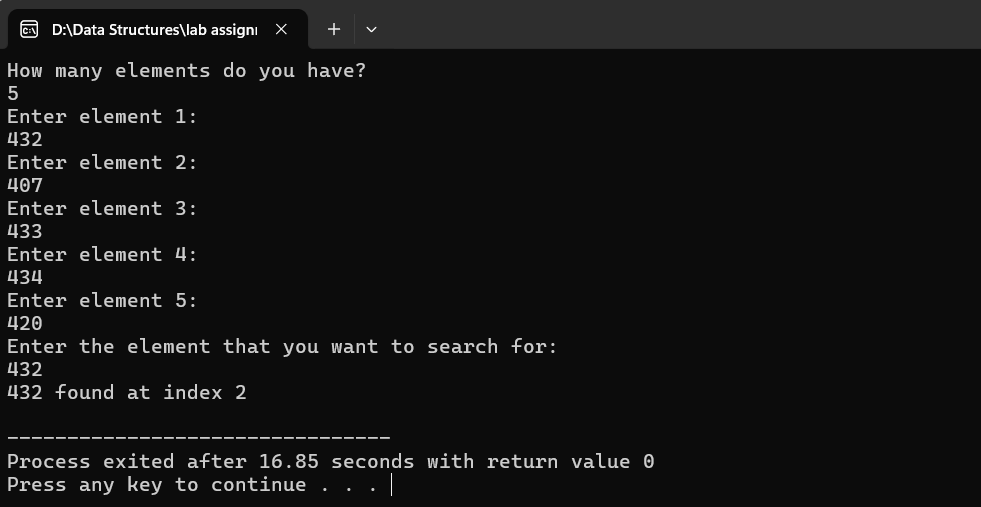
cout << key << " not found" << endl;

}

return EXIT\_SUCCESS;

}

**Output:**



# Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Code the Bubble sort with the following elements:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 64 | 34 | 25 | 12 | 22 | 11 | 90 |

**Ans:**

#include <iostream>

using namespace std;

void bubbleSort(int arr[], int size)

{

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

{

bool swapped = false;

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

{

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

{

// Swap the elements

int temp = arr[j];

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

arr[j + 1] = temp;

swapped = true;

}

}

// If no two elements were swapped by inner loop, then break

if (!swapped)

{

break;

}

}

}

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

{

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

{

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

}

cout << endl;

}

int main()

{

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

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

cout << "Original array:" << endl;

printArray(arr, size);

bubbleSort(arr, size);

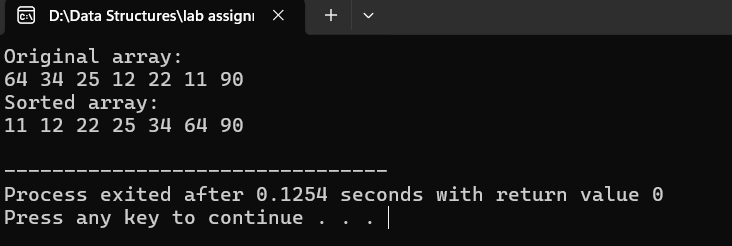
cout << "Sorted array:" << endl;

printArray(arr, size);

return 0;

}

**Output:**



# Design the Logic to Find a Missing Number in a Sorted Array.

**Ans:**

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

bool check(float arr[], float nthterm, int index)

{

if (arr[index] != nthterm)

{

return true;

}

return false;

}

void commonDifference(float arr[], int size)

{

float cd = ((arr[size - 1] - arr[0]) / (float)(size));

float nthterm = 0;

int i = 0;

while (nthterm < arr[size - 1])

{

i++;

nthterm = arr[0] + i \* cd;

if (check(arr, nthterm, i))

{

cout << nthterm << " is missing";

break;

}

}

}

void commonRatio(float arr[], int size)

{

float cr = pow(((float)arr[size - 1] / arr[0]), ((float)1 / (size)));

float nthterm = 0;

int i = 0;

while (nthterm < arr[size - 1])

{

i++;

nthterm = arr[0] \* pow(cr, i - 1);

if (check(arr, nthterm, i - 1))

{

cout << nthterm << " is missing";

break;

}

}

}

void harmonictoarithmetic(float arr[], int size)

{

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

{

arr[i] = (float)1 / arr[i];

}

cout << "1 / ";

commonDifference(arr, size);

}

void checkprogression(float arr[], int size)

{

float cr = pow(((float)arr[size - 1] / arr[0]), ((float)1 / (size)));

float cd = ((arr[size - 1] - arr[0]) / (float)(size));

if ((arr[1] == arr[0] + cd) or (arr[size - 2] == (float) arr[size-1]- cd))

{

cout << "is arithematic progression" << endl;

commonDifference(arr, size);

}

else if ((arr[1] == arr[0] \* cr) or (arr[size - 2] == arr[size-1]/cr))

{

cout << "is geometric progression" << endl;

commonRatio(arr, size);

}

else

{

cout << "is harmonic progression" << endl;

harmonictoarithmetic(arr, size);

}

}

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

int size;

float \*arr = nullptr;

cout << "enter the size of array : ";

cin >> size;

arr = new float[size]();

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

{

cout << "element " << i + 1 << endl;

cin >> arr[i];

}

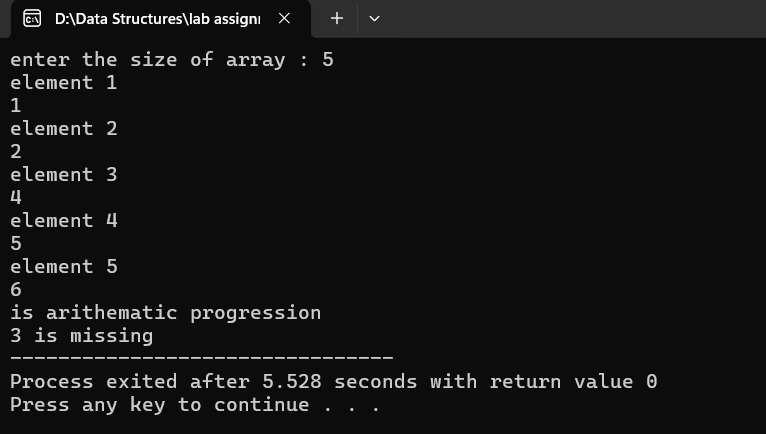
checkprogression(arr, size);

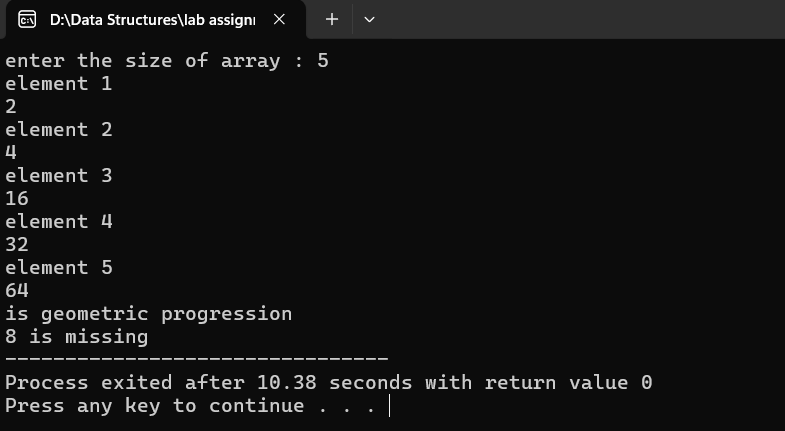
delete[] arr;

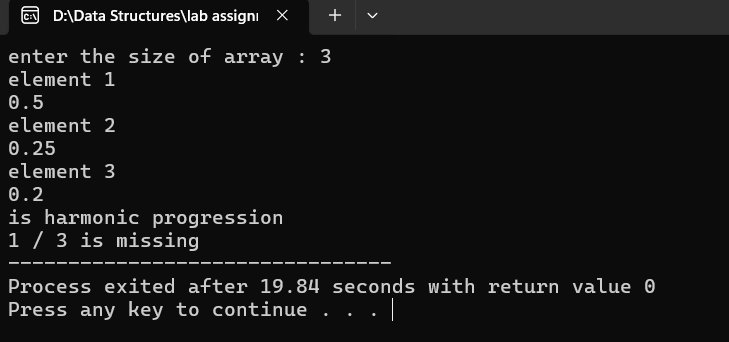
return EXIT\_SUCCESS;

}

**Output:**







# String Related Programs

## Write a program to concatenate one string to another string.

**Ans:**

#include <iostream>

#include <cstdlib>

#include<string>

using namespace std;

string add(string a, string b)

{

return a+b;

}

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

string first,second;

cout<<"enter first string "<<endl;

cin>>first;

cout<<"enter the second string "<<endl;

cin>>second;

string third=add(first,second);

cout<<third<<endl;

//

first.append(second);

cout<<first<<endl;

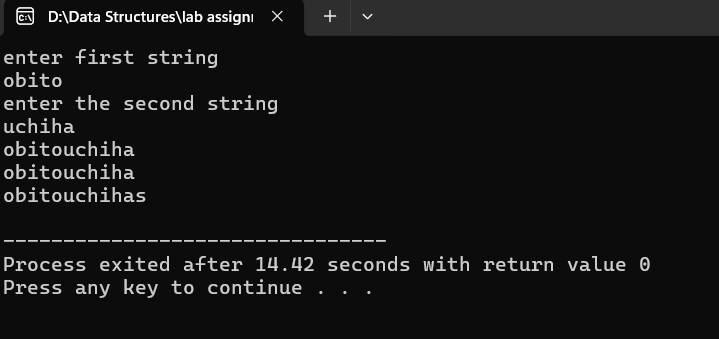
first.push\_back('s');

cout<<first<<endl;

return EXIT\_SUCCESS;

}

**Output:**



## Write a program to reverse a string.

**Ans:**

#include <iostream>

#include <cstdlib>

#include<string>

using namespace std;

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

string str;

cout<<"enter any string"<<endl;

getline(cin,str);

for (int i =0;i<str.length()/2;i++)

{

swap(str[i],str[str.length()-i-1]);

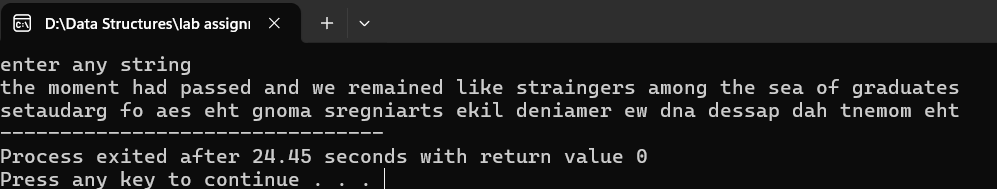
}

cout<<str;

return EXIT\_SUCCESS;

}

**Output:**



## Write a program to delete all the vowels from the string.

**Ans:**

#include <iostream>

#include <cstdlib>

#include<string>

using namespace std;

string removeVowels(string a)

{

string b;

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

{

if (a[i]!='a' and a[i]!='e' and a[i]!='i' and a[i]!='o' and a[i]!='u')

{

b.push\_back(a[i]);

}

}

return b;

}

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

string a;

cout<<"enter string : "<<endl;

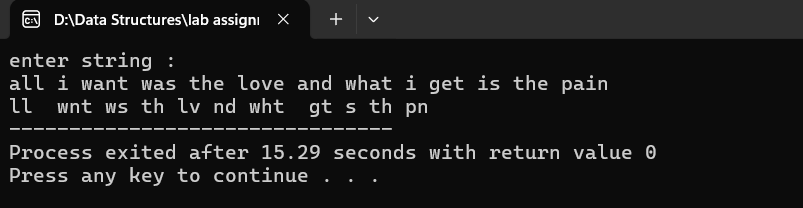
getline(cin,a);

cout<<removeVowels(a);

return EXIT\_SUCCESS;

}

**Output:**



## Write a program to sort the strings in alphabetical order.

**Ans:**

#include <iostream>

#include <cstdlib>

#include <string>

using namespace std;

string stringsort(string a)

{

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

{

for (int j = 0; j < a.size() - i - 1; j++)

{

if (a[j] > a[j + 1])

{

// char temp=a[j];

// a[j]=a[j+1];

// a[j+1]=temp;

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

}

}

}

return a;

}

int main()

{

// #ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

// #endif

string a;

cout << "enter string : " << endl;

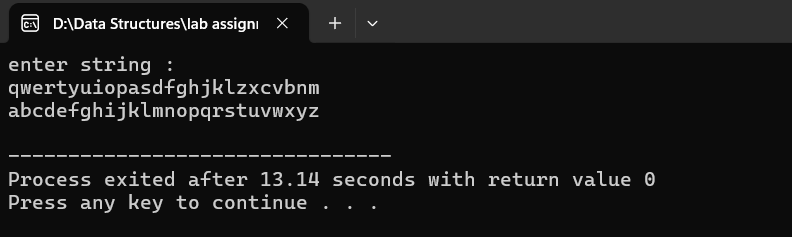
getline(cin, a);

cout<<stringsort(a)<<endl;

return EXIT\_SUCCESS;

}

**Output:**



## Write a program to convert a character from uppercase to lowercase.

**Ans:**

#include <iostream>

#include <cstdlib>

#include <string>

using namespace std;

string toUppercase(string a)

{

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

{

a[i]=a[i]-32;

}

return a;

}

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

string a;

cout << "enter string : " << endl;

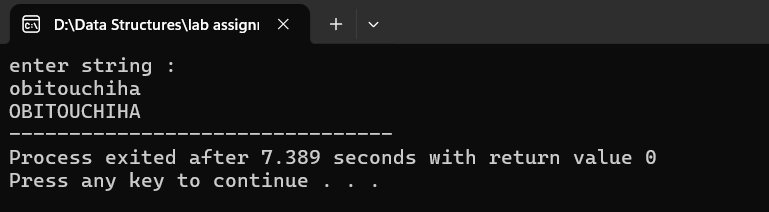
getline(cin, a);

cout<<toUppercase(a);

return EXIT\_SUCCESS;

}

**Output:**



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

# 𝑐𝑜𝑙𝑢𝑚𝑛𝑠. Assuming array is used to store elements of the following matrices, implement an efficient way that reduces the space requirement.

## Diagonal Matrix.

**Ans:**

#include <iostream>

#include<vector>

using namespace std;

class DiagonalMatrix

{

public:

vector <int> arr;

// DiagonalMatrix() = default;

// // Constructor with initialization

// explicit DiagonalMatrix(const vector<int>& values) : arr(values) {}

DiagonalMatrix()

{

}

DiagonalMatrix(int a)

{

arr.push\_back(a);

}

DiagonalMatrix(int a,int b)

{

arr.push\_back(a);

arr.push\_back(b);

}

DiagonalMatrix(int a,int b, int c)

{

arr.push\_back(a);

arr.push\_back(b);

arr.push\_back(c);

}

void display()

{

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

{

for( int j=0;j<arr.size();j++)

{

if (i==j)

{

cout<<arr[j]<<" ";

}

else{

cout<<0<<" ";

}

}

cout<<endl;

}

}

~DiagonalMatrix() = default;

};

main()

{

//#ifndef ONLINE\_JUDGE

//freopen("input.txt", "r", stdin);

//freopen("output.txt", "w", stdout);

//#endif

// two dimensional diagonal array

DiagonalMatrix d1;

d1.arr.push\_back(1);

d1.arr.push\_back(2);

d1.display();

cout<<endl<<endl;

//3 dimensional diagonal array

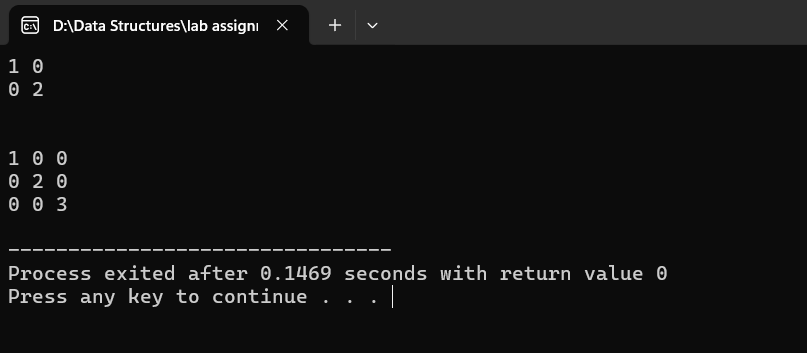
DiagonalMatrix d2(1,2,3);

d2.display();

return 0;

}

**Output :**

****

## Tri-diagonal Matrix.

**Ans:**

#include <iostream>

using namespace std;

class TriDiagonalMatrix

{

private:

int \*upperDiagonal;

int \*mainDiagonal;

int \*lowerDiagonal;

public:

int size;

TriDiagonalMatrix(int size) : size(size)

{

upperDiagonal = new int[size - 1]();

mainDiagonal = new int[size]();

lowerDiagonal = new int[size - 1]();

}

~TriDiagonalMatrix()

{

delete[] mainDiagonal;

delete[] upperDiagonal;

delete[] lowerDiagonal;

}

void set(int i, int j, int value)

{

if (i == j)

{

mainDiagonal[i] = value;

}

else if (i < j)

{

if (i < size - 1) // Ensure the index is within bounds

{

upperDiagonal[i] = value;

}

}

else

{

if (j < size - 1) // Ensure the index is within bounds

{

lowerDiagonal[j] = value;

}

}

}

void display()

{

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

{

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

{

if (i == j)

{

cout << mainDiagonal[i] << " ";

}

else if (i == j - 1)

{

cout << upperDiagonal[i] << " ";

}

else if (j == i - 1)

{

cout << lowerDiagonal[j] << " ";

}

else

cout << "0 ";

}

cout << endl;

}

}

};

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

TriDiagonalMatrix d1(5);

d1.set(0, 0, 4);

d1.set(0, 1, 1);

d1.set(1, 0, 3);

d1.set(1, 1, 4);

d1.set(1, 2, 1);

d1.set(2, 1, 3);

d1.set(2, 2, 4);

d1.set(2, 3, 1);

d1.set(3, 2, 3);

d1.set(3, 3, 4);

d1.set(3, 4, 1);

d1.set(4, 3, 3);

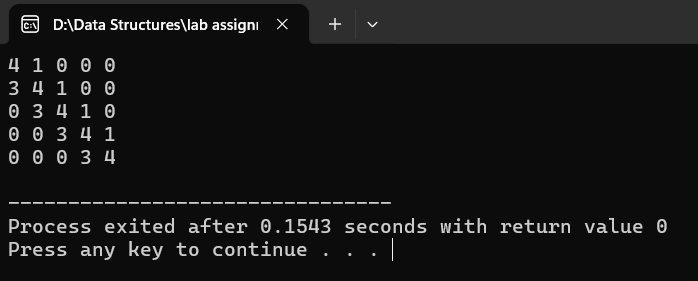
d1.set(4, 4, 4);

d1.display();

return 0;

}

**Output:**



## Lower triangular Matrix.

**Ans:**

#include <iostream>

using namespace std;

class LowerTriangularMatrix

{

protected:

int index = 0;

private:

int size;

int \*arr = nullptr;

public:

LowerTriangularMatrix(int size)

{

this->size = size;

arr = new int[size \* (size + 1) / 2]();

}

void set(int i, int j, int value)

{

if (i >= j)

{

arr[index++] = value;

}

else

{

cout << "not valid";

}

}

void display() {

int k = 0;

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

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

if (i >= j) {

cout << arr[k++] << " ";

} else {

cout << 0 << " ";

}

}

cout << endl;

}

}

};

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

int n = 4; // Example size of the matrix (4x4)

LowerTriangularMatrix matrix(n);

// Set elements in the lower triangular matrix

matrix.set(0, 0, 5);

matrix.set(1, 0, 8);

matrix.set(1, 1, 7);

matrix.set(2, 0, 3);

matrix.set(2, 1, 6);

matrix.set(2, 2, 4);

matrix.set(3, 0, 2);

matrix.set(3, 1, 9);

matrix.set(3, 2, 1);

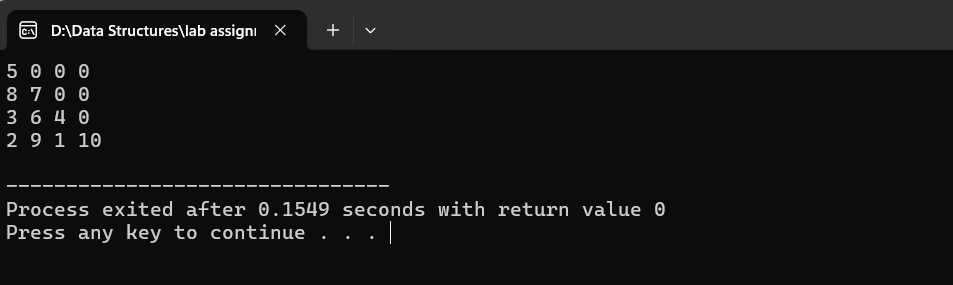
matrix.set(3, 3, 10);

matrix.display();

return 0;

}

**Output:**

****

## Upper triangular Matrix.

**Ans:**

#include <iostream>

using namespace std;

class upperTriangularMatrix

{

protected:

int index = 0;

private:

int size;

int \*arr = nullptr;

public:

upperTriangularMatrix(int size)

{

this->size = size;

arr = new int[size \* (size + 1) / 2]();

}

void set(int i, int j, int value)

{

if (j >= i)

{

arr[index++] = value;

}

else

{

cout << "not valid";

}

}

void display()

{

int k = 0;

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

{

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

{

if (j >= i)

{

cout << arr[k++] << " ";

}

else

{

cout << 0 << " ";

}

}

cout << endl;

}

}

};

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

int n = 4; // Example size of the matrix (4x4)

upperTriangularMatrix matrix(n);

// Set elements in the lower triangular matrix

matrix.set(0, 0, 5);

matrix.set(0, 1, 8);

matrix.set(1, 1, 7);

matrix.set(0, 2, 3);

matrix.set(1, 2, 6);

matrix.set(2, 2, 4);

matrix.set(0, 3, 2);

matrix.set(1, 3, 9);

matrix.set(2, 3, 1);

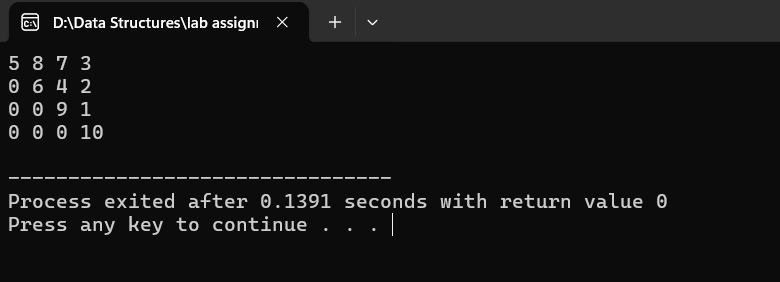
matrix.set(3, 3, 10);

matrix.display();

return 0;

}

**Output:**

****

## Symmetric Matrix

**Ans:**

#include <iostream>

using namespace std;

class symmetricMatrix

{

protected:

int index = 0;

private:

int size;

int \*arr = nullptr;

public:

symmetricMatrix(int size)

{

this->size = size;

arr = new int[size \* (size + 1) / 2]();

}

void set(int i, int j, int value)

{

if (i >= j)

{

arr[index++] = value;

}

}

void display()

{

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

{

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

{

if (i >= j)

{

index = i \* (i + 1) / 2 + j;

cout << arr[index] << " ";

}

else

{

index = j \* (j + 1) / 2 + i;

cout << arr[index] << " ";

}

}

cout << endl;

}

}

};

int main()

{

//#ifndef ONLINE\_JUDGE

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

//#endif

int n = 4;

symmetricMatrix matrix(n);

matrix.set(0, 0, 1);

matrix.set(1, 0, 2);

matrix.set(1, 1, 3);

matrix.set(2, 0, 4);

matrix.set(2, 1, 5);

matrix.set(2, 2, 6);

matrix.set(3, 0, 7);

matrix.set(3, 1, 8);

matrix.set(3, 2, 9);

matrix.set(3, 3, 10);

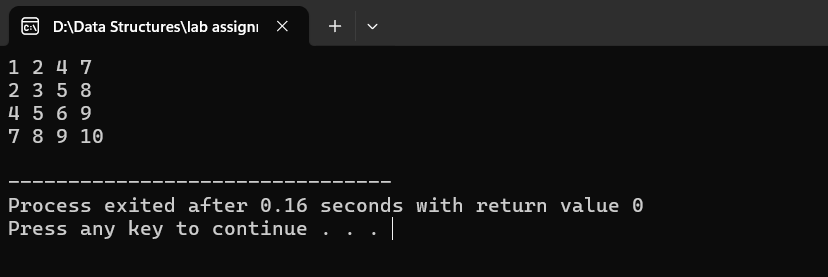
// Display the symmetric matrix

matrix.display();

return 0;

}

**Output:**



## Write a program to implement the following operations on a Sparse Matrix, assuming the matrix is represented using a triplet.

## Transpose of a matrix.

## Addition of two matrices.

## Multiplication of two matrices.

**Ans:**

#include <iostream>

#include <cstdlib>

#include <algorithm> // for std::sort

using namespace std;

class SparseMatrix

{

private:

int \*\*sparse = nullptr;

int nonZeros;

int rows, columns;

int current;

public:

SparseMatrix(int rows, int columns, int nonZeros)

{

this->rows = rows;

this->columns = columns;

this->nonZeros = nonZeros;

this->current = 1; // Start from 1 because index 0 is reserved for matrix dimensions

sparse = new int \*[nonZeros + 1]; // Create an array for storing non-zero values

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

{

sparse[i] = new int[3]; // Each element holds row, column, and value

}

sparse[0][0] = rows;

sparse[0][1] = columns;

sparse[0][2] = nonZeros;

}

~SparseMatrix()

{

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

{

delete[] sparse[i];

}

delete[] sparse;

}

void setMatrix(int row, int column, int value)

{

if (row >= rows || column >= columns || row < 0 || column < 0)

{

cout << "Invalid indexing" << endl;

return;

}

if (current <= nonZeros)

{

sparse[current][0] = row;

sparse[current][1] = column;

sparse[current][2] = value;

current++;

}

else

{

cout << "Exceeded non-zero element limit" << endl;

}

}

void displaySparse() const

{

cout << "Sparse Matrix Representation:" << endl;

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

{

cout << sparse[i][0] << "\t" << sparse[i][1] << "\t" << sparse[i][2] << "\n";

}

}

void displayMatrix() const

{

cout << "Matrix Representation:" << endl;

int count = 1;

for (int i = 0; i < this->sparse[0][0]; i++)

{

for (int j = 0; j < this->sparse[0][1]; j++)

{

if (count <= nonZeros && sparse[count][0] == i && sparse[count][1] == j)

{

cout << this->sparse[count][2] << "\t";

++count;

}

else

{

cout << 0 << "\t";

}

}

cout << endl;

}

}

static bool sortSparse(const int \*a, const int \*b)

{

if (a[0] == b[0])

return a[1] < b[1]; // Sort by column if rows are equal

return a[0] < b[0]; // Sort by row

}

SparseMatrix Transpose() const

{

SparseMatrix transposed(this->columns, this->rows, this->nonZeros); // Create template for transposed matrix

transposed.sparse[0][0] = this->columns;

transposed.sparse[0][1] = this->rows;

transposed.sparse[0][2] = this->nonZeros;

// Transpose the elements by swapping rows and columns

for (int i = 1; i <= this->nonZeros; i++)

{

transposed.sparse[i][0] = this->sparse[i][1]; // Swap row and column

transposed.sparse[i][1] = this->sparse[i][0];

transposed.sparse[i][2] = this->sparse[i][2];

}

// Sort the transposed matrix to maintain row-major order

sort(transposed.sparse + 1, transposed.sparse + nonZeros + 1, sortSparse);

return transposed;

}

SparseMatrix operator\*(const SparseMatrix &b) const

{

if (this->columns != b.rows)

{

cout << "Matrix dimensions do not match for multiplication." << endl;

exit(EXIT\_FAILURE);

}

SparseMatrix bTransposed = b.Transpose();

SparseMatrix result(this->rows, b.columns, this->rows \* b.columns);

int apos = 1;

int bpos;

while (apos <= this->nonZeros)

{

int r = this->sparse[apos][0];

bpos = 1;

while (bpos <= bTransposed.nonZeros)

{

int c = bTransposed.sparse[bpos][0];

int tempa = apos;

int tempb = bpos;

int sum = 0;

while (tempa <= this->nonZeros && this->sparse[tempa][0] == r &&

tempb <= bTransposed.nonZeros && bTransposed.sparse[tempb][0] == c)

{

if (this->sparse[tempa][1] < bTransposed.sparse[tempb][1])

++tempa;

else if (this->sparse[tempa][1] > bTransposed.sparse[tempb][1])

++tempb;

else

sum += this->sparse[tempa++][2] \* bTransposed.sparse[tempb++][2];

}

if (sum != 0)

result.setMatrix(r, c, sum);

while (bpos <= bTransposed.nonZeros && bTransposed.sparse[bpos][0] == c)

++bpos;

}

while (apos <= this->nonZeros && this->sparse[apos][0] == r)

++apos;

}

result.nonZeros = result.current - 1;

return result;

}

};

int main()

{

// Initialize matrices

SparseMatrix mat1(4, 4, 4);

mat1.setMatrix(0, 0, 1);

mat1.setMatrix(1, 1, 3);

mat1.setMatrix(2, 2, 2);

mat1.setMatrix(3, 3, 1);

SparseMatrix mat2(4, 4, 4);

mat2.setMatrix(0, 0, 2);

mat2.setMatrix(1, 1, 3);

mat2.setMatrix(2, 2, 4);

mat2.setMatrix(3, 3, 5);

cout << "Matrix A (4x4):" << endl;

mat1.displayMatrix();

cout << endl;

cout << "Matrix B (4x4):" << endl;

mat2.displayMatrix();

cout << endl;

SparseMatrix product = mat1 \* mat2;

cout << "Product Matrix A \* B (4x4):" << endl;

product.displayMatrix();

cout << endl;

// Test with another example

SparseMatrix A(2, 2, 2);

A.setMatrix(0, 0, 1);

A.setMatrix(0, 1, 2);

SparseMatrix B(2, 2, 2);

B.setMatrix(1, 0, 1);

B.setMatrix(1, 1, 2);

cout << "Matrix A (2x2):" << endl;

A.displayMatrix();

cout << endl;

cout << "Matrix B (2x2):" << endl;

B.displayMatrix();

cout << endl;

SparseMatrix C = A \* B;

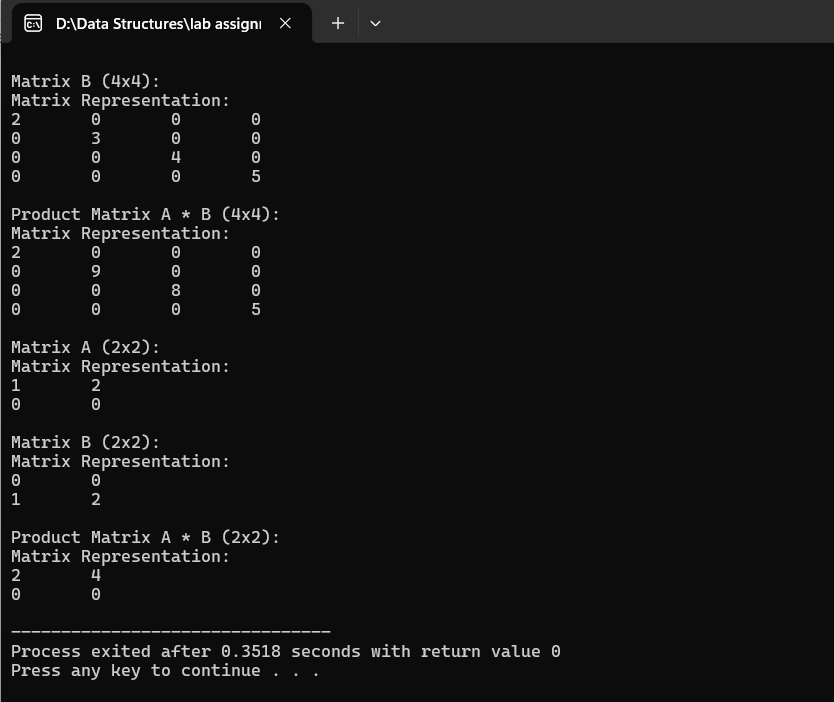
cout << "Product Matrix A \* B (2x2):" << endl;

C.displayMatrix();

return EXIT\_SUCCESS;

}

**Output:**



# 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.

**Ans:**

#include <iostream>

#include <cstdlib>

using namespace std;

int countInversion(int \*arr, int size)

{

int count = 0;

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

{

for (int j = i + 1; j < size; j++)

{

if (i < j and arr[i] > arr[j])

{

++count;

}

}

}

return count;

}

int main()

{

int \*arr = nullptr;

int size;

cout << "enter the size of array :";

cin >> size;

arr = new int[size];

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

{

cout << "Element " << i + 1<<"\t";

cin >> arr[i];

}

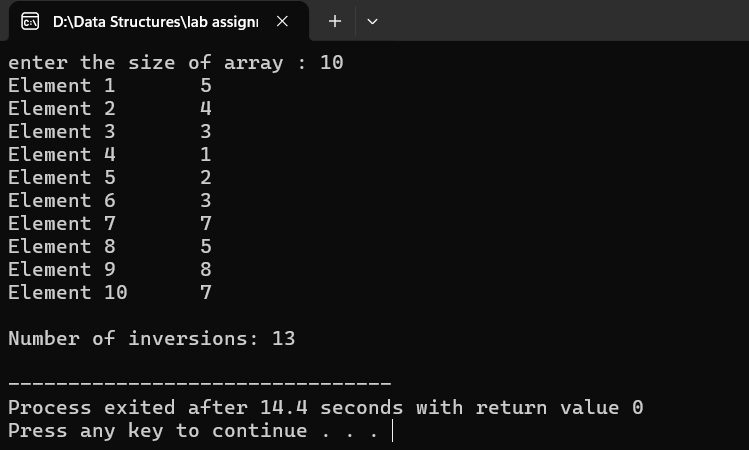
cout << endl

<< "Number of inversions: " << countInversion(arr, size) << endl;

return EXIT\_SUCCESS;

}

**Output:**



# Write a program to count the total number of distinct elements in an array of length *n*.

**Ans:**

#include <iostream>

#include <cstdlib>

using namespace std;

int countUnique(int \*&arr, int size) {

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

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

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

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

}

}

}

int count = 0;

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

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

++count;

}

}

return ++count;

}

int main() {

int size;

int \*arr = nullptr;

cout << "How many data do you have?" << endl;

cin >> size;

arr = new int[size];

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

cout << "Element " << i + 1 << ": ";

cin >> arr[i];

}

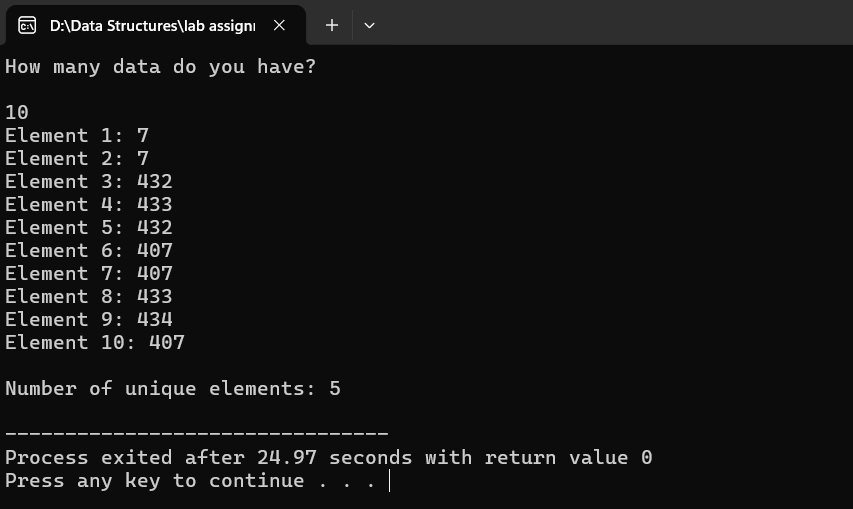
cout << endl << "Number of unique elements: " << countUnique(arr, size) << endl;

delete[] arr;

return EXIT\_SUCCESS;

}

**Output:**



# Additional Questions

# Write a program to find a saddle point in a two-dimensional array. A saddle point in a numerical array is a number that is larger than or equal to every number in its column, and smaller than or equal to every number in its row.

**Ans:**

#include <iostream>

#include <cstdlib>

using namespace std;

int saddlePoint(int \*\*arr, int row, int column)

{

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

{

int minRow = arr[i][0];

int colIndex = 0;

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

{

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

{

minRow = arr[i][j];

colIndex = j;

}

}

bool isSaddle = true;

for (int k = 0; k < row; k++)

{

if (arr[k][colIndex] > minRow)

{

isSaddle = false;

break;

}

}

if (isSaddle)

{

return minRow;

}

}

return -1;

}

int main()

{

int \*\*arr;

int row, column;

cout << "Enter the dimensions of the 2-D array (rows and columns): ";

cin >> row >> column;

arr = new int \*[row];

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

{

arr[i] = new int[column];

}

cout << "Enter the elements of the array: \n";

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

{

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

{

cin >> arr[i][j];

}

}

int saddle = saddlePoint(arr, row, column);

if (saddle == -1)

{

cout << "No saddle point found.\n";

}

else

{

cout << "Saddle point: " << saddle << endl;

}

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

{

delete[] arr[i];

}

delete[] arr;

return EXIT\_SUCCESS;

}

**Output:**

