**Assignment – 2**

**DSA**

**Shefali Sharma – 1024030284 2C35**

**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.**

A1)

#include <iostream> using namespace std;

int binarySearch(int arr[], int n, int target) { int left = 0, right = n - 1; while (left <= right) { int mid = left + (right - left) / 2; if (arr[mid] == target) return mid; // target found at index mid else if (arr[mid] < target) left = mid + 1; else right = mid - 1;

}

return -1; // target not found

}

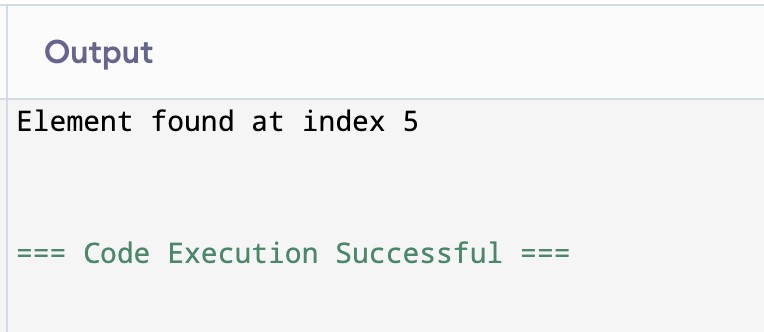
int main() {

int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}; int n = sizeof(arr) / sizeof(arr[0]); int target = 23;

int result = binarySearch(arr, n, target); if (result != -1) cout << "Element found at index " << result << endl; else cout << "Element not found" << endl;

return 0;

}



**2) 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**

A2)

#include <iostream> using namespace std;

void bubbleSort(int arr[], int n) { for (int i = 0; i < n - 1; i++) { bool swapped = false;

for (int j = 0; j < n - i - 1; j++) { if (arr[j] > arr[j + 1]) { swap(arr[j], arr[j + 1]); swapped = true;

}

}

if (!swapped) // No swaps means array is sorted break;

}

}

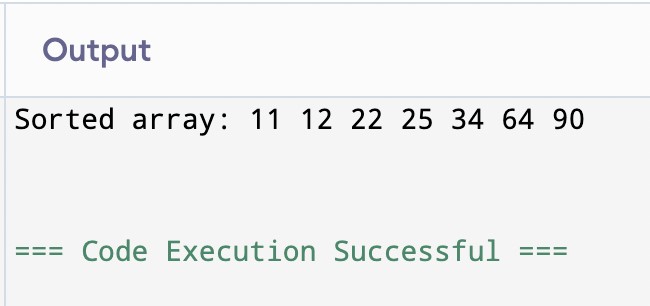
int main() { int arr[] = {64, 34, 25, 12, 22, 11, 90}; int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

cout << "Sorted array: "; for (int i = 0; i < n; i++) cout << arr[i] << " "; cout << endl;

return 0;

}



**3) Design the Logic to Find a Missing Number in a Sorted Array.**

#include <iostream> using namespace std;

// Assumes numbers from start to end with exactly one missing number int findMissingNumber(int arr[], int n) {

int start = arr[0];

for (int i = 0; i < n; i++) { if (arr[i] != start + i) return start + i;

}

return -1; // No missing number

}

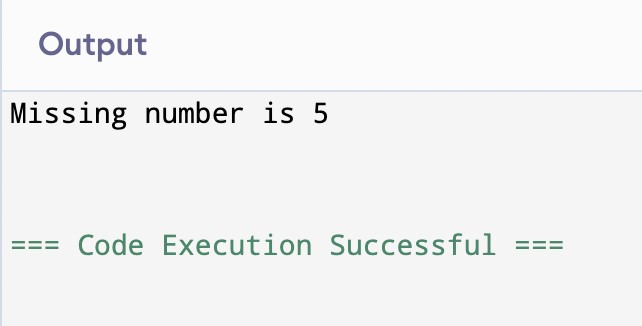
int main() { int arr[] = {1, 2, 3, 4, 6, 7, 8}; // 5 is missing int n = sizeof(arr) / sizeof(arr[0]);

int missing = findMissingNumber(arr, n); if (missing != -1) cout << "Missing number is " << missing << endl; else

cout << "No missing number" << endl;

return 0;

}



**4) String Related Programs (a) Write a program to concatenate one string to another string. (b) Write a program to reverse a string. (c) Write a program to delete all the vowels from the string. (d) Write a program to sort the strings in alphabetical order. (e) Write a program to convert a character from uppercase to lowercase.**

4) #include <iostream>

#include <algorithm> // for sort

#include <cctype> // for tolower, toupper using namespace std;

// (a) Concatenate two strings string concatenate(string s1, string s2) { return s1 + s2;

}

// (b) Reverse a string string reverseString(string s) { reverse(s.begin(), s.end()); return s;

}

// (c) Delete all vowels string deleteVowels(string s) { string result; for (char c : s) { if (!(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' || c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')) { result += c;

}

}

return result;

}

// (d) Sort strings alphabetically void sortStrings(string arr[], int n) { sort(arr, arr + n);

}

// (e) Convert uppercase to lowercase char toLowercase(char c) {

if (c >= 'A' && c <= 'Z') return c + 32; return c;

}

int main() { // Test (a) string s1 = "Hello", s2 = "World"; cout << "Concatenated string: " << concatenate(s1, s2) << endl;

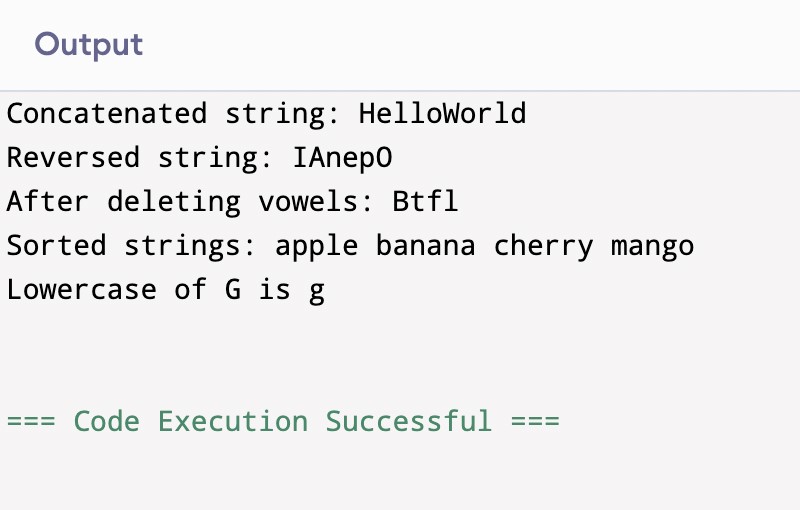
// Test (b) string s3 = "OpenAI"; cout << "Reversed string: " << reverseString(s3) << endl;

// Test (c) string s4 = "Beautiful"; cout << "After deleting vowels: " << deleteVowels(s4) << endl; // Test (d) string arr[] = {"banana", "apple", "mango", "cherry"}; int n = sizeof(arr) / sizeof(arr[0]); sortStrings(arr, n); cout << "Sorted strings: "; for (int i = 0; i < n; i++) cout << arr[i] << " "; cout << endl;

// Test (e) char ch = 'G'; cout << "Lowercase of " << ch << " is " << toLowercase(ch) << endl;

return 0;

}



**5) Space required to store any two-dimensional array is** 𝑛𝑢𝑚𝑏𝑒𝑟𝑜**ƒ** 𝑟𝑜𝑤𝑠 **×** 𝑛𝑢𝑚𝑏𝑒𝑟𝑜**ƒ** 𝑐𝑜𝑙𝑢𝑚𝑛𝑠**. Assuming array is used to store elements of the following matrices, implement an eYicient way that reduces the space requirement.**

1. **Diagonal Matrix.**
2. **Tri-diagonal Matrix.**
3. **Lower triangular Matrix.**
4. **Upper triangular Matrix.**
5. **Symmetric Matrix**

5)

#include <iostream> using namespace std;

// (a) Diagonal Matrix Storage class DiagonalMatrix {

int \*arr; int n; public:

DiagonalMatrix(int size) { n = size; arr = new int[n]();

}

void set(int i, int j, int val) { if (i == j) arr[i - 1] = val;

}

int get(int i, int j) { if (i == j)

return arr[i - 1]; else return 0;

}

~DiagonalMatrix() { delete[] arr; }

};

// (b) Tri-diagonal Matrix Storage class TriDiagonalMatrix {

int \*arr; int n; public:

TriDiagonalMatrix(int size) {

n = size; arr = new int[3 \* n - 2]();

}

void set(int i, int j, int val) { if (i - j == 1) // lower diagonal arr[i - 2] = val; else if (i == j) // main diagonal arr[n - 1 + i - 1] = val; else if (j - i == 1) // upper diagonal arr[2 \* n - 1 + i - 1] = val;

}

int get(int i, int j) { if (i - j == 1) return arr[i - 2]; else if (i == j) return arr[n - 1 + i - 1]; else if (j - i == 1) return arr[2 \* n - 1 + i - 1]; else return 0;

}

~TriDiagonalMatrix() { delete[] arr; }

};

// (c) Lower Triangular Matrix Storage class LowerTriangularMatrix {

int \*arr; int n; public:

LowerTriangularMatrix(int size) {

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

}

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]; else return 0;

}

~LowerTriangularMatrix() { delete[] arr; }

};

// (d) Upper Triangular Matrix Storage class UpperTriangularMatrix {

int \*arr; int n; public:

UpperTriangularMatrix(int size) {

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

}

void set(int i, int j, int val) { if (i <= j) arr[(j \* (j - 1)) / 2 + i - 1] = val;

}

int get(int i, int j) { if (i <= j) return arr[(j \* (j - 1)) / 2 + i - 1]; else return 0;

}

~UpperTriangularMatrix() { delete[] arr; }

};

// (e) Symmetric Matrix Storage (store only lower triangle) class SymmetricMatrix {

int \*arr;

int n; public:

SymmetricMatrix(int size) {

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

}

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;

}

int get(int i, int j) { if (i >= j) return arr[(i \* (i - 1)) / 2 + j - 1]; else

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

}

~SymmetricMatrix() { delete[] arr; }

};

int main() { cout << "--- Diagonal Matrix ---\n"; DiagonalMatrix dm(3); dm.set(1, 1, 5); dm.set(2, 2, 8); dm.set(3, 3, 10);

for (int i = 1; i <= 3; i++) {

for (int j = 1; j <= 3; j++) cout << dm.get(i, j) << " "; cout << "\n";

}

cout << "\n--- Tri-Diagonal Matrix ---\n"; TriDiagonalMatrix tdm(4); tdm.set(1, 1, 1); tdm.set(1, 2, 2); tdm.set(2, 1, 3); tdm.set(2, 2, 4); tdm.set(2, 3, 5); tdm.set(3, 2, 6); tdm.set(3, 3, 7); tdm.set(3, 4, 8); tdm.set(4, 3, 9); tdm.set(4, 4, 10);

for (int i = 1; i <= 4; i++) { for (int j = 1; j <= 4; j++) cout << tdm.get(i, j) << " "; cout << "\n";

}

cout << "\n--- Lower Triangular Matrix ---\n"; LowerTriangularMatrix ltm(3); ltm.set(1, 1, 1);

ltm.set(2, 1, 2); ltm.set(2, 2, 3); ltm.set(3, 1, 4); ltm.set(3, 2, 5); ltm.set(3, 3, 6);

for (int i = 1; i <= 3; i++) { for (int j = 1; j <= 3; j++) cout << ltm.get(i, j) << " "; cout << "\n";

}

cout << "\n--- Upper Triangular Matrix ---\n"; UpperTriangularMatrix utm(3); utm.set(1, 1, 7); utm.set(1, 2, 8); utm.set(1, 3, 9); utm.set(2, 2, 10); utm.set(2, 3, 11); utm.set(3, 3, 12);

for (int i = 1; i <= 3; i++) { for (int j = 1; j <= 3; j++) cout << utm.get(i, j) << " "; cout << "\n";

}

cout << "\n--- Symmetric Matrix ---\n"; SymmetricMatrix sm(3);

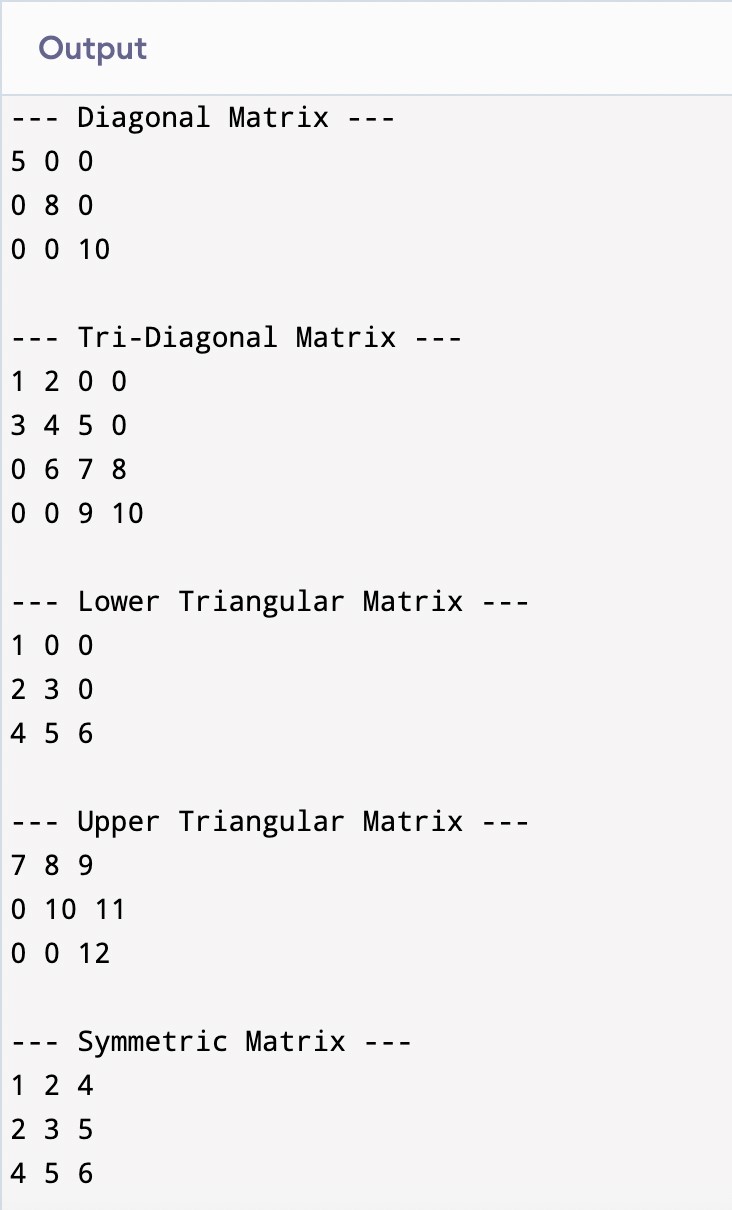
sm.set(1, 1, 1); sm.set(2, 1, 2); sm.set(2, 2, 3); sm.set(3, 1, 4); sm.set(3, 2, 5); sm.set(3, 3, 6);

for (int i = 1; i <= 3; i++) { for (int j = 1; j <= 3; j++) cout << sm.get(i, j) << " "; cout << "\n";

}

return 0;

}



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

1. **Transpose of a matrix.**
2. **Addition of two matrices.**
3. **Multiplication of two matrices.**

6) #include <iostream>

#include <vector>

#include <algorithm>

#include <map>

using namespace std;

struct Triplet { int row, col, val;

};

bool compareTriplets(const Triplet &a, const Triplet &b) { if (a.row == b.row) return a.col < b.col; return a.row < b.row;

}

class SparseMatrix { public:

int rows, cols;

vector<Triplet> triplets;

SparseMatrix(int r, int c) : rows(r), cols(c) {}

void addTriplet(int r, int c, int v) { if (v != 0) triplets.push\_back({r, c, v});

}

void display() { cout << "Row\tCol\tValue\n"; for (auto &t : triplets) { cout << t.row << "\t" << t.col << "\t" << t.val << "\n";

}

}

SparseMatrix transpose() { SparseMatrix result(cols, rows); for (auto &t : triplets) { result.addTriplet(t.col, t.row, t.val);

}

sort(result.triplets.begin(), result.triplets.end(), compareTriplets); return result;

}

SparseMatrix add(SparseMatrix &other) { if (rows != other.rows || cols != other.cols) { cout << "Matrix dimensions do not match for addition.\n"; return SparseMatrix(0, 0);

}

SparseMatrix result(rows, cols);

sort(triplets.begin(), triplets.end(), compareTriplets); sort(other.triplets.begin(), other.triplets.end(), compareTriplets);

int i = 0, j = 0;

while (i < (int)triplets.size() && j < (int)other.triplets.size()) {

Triplet &a = triplets[i];

Triplet &b = other.triplets[j];

if (a.row == b.row && a.col == b.col) { int summedVal = a.val + b.val; if (summedVal != 0) result.addTriplet(a.row, a.col, summedVal); i++; j++;

}

else if (a.row < b.row || (a.row == b.row && a.col < b.col)) { result.addTriplet(a.row, a.col, a.val); i++;

} else {

result.addTriplet(b.row, b.col, b.val); j++;

}

}

while (i < (int)triplets.size()) { result.addTriplet(triplets[i].row, triplets[i].col, triplets[i].val); i++;

}

while (j < (int)other.triplets.size()) { result.addTriplet(other.triplets[j].row, other.triplets[j].col, other.triplets[j].val); j++;

}

return result;

}

SparseMatrix multiply(SparseMatrix &other) { if (cols != other.rows) { cout << "Matrix dimensions do not match for multiplication.\n"; return SparseMatrix(0, 0);

}

SparseMatrix result(rows, other.cols);

// Map rows of second matrix for quick access map<int, vector<Triplet>> otherRowMap; for (auto &t : other.triplets) { otherRowMap[t.row].push\_back(t);

}

for (auto &a : triplets) { if (otherRowMap.find(a.col) != otherRowMap.end()) { for (auto &b : otherRowMap[a.col]) {

int r = a.row; int c = b.col; int val = a.val \* b.val;

// Check if already have this position, then add bool found = false; for (auto &resTrip : result.triplets) { if (resTrip.row == r && resTrip.col == c) { resTrip.val += val; found = true; break;

}

} if (!found) { result.addTriplet(r, c, val);

}

}

}

}

// Remove zero entries if any vector<Triplet> filtered; for (auto &t : result.triplets) { if (t.val != 0) filtered.push\_back(t);

}

result.triplets = filtered;

sort(result.triplets.begin(), result.triplets.end(), compareTriplets); return result;

}

};

int main() {

SparseMatrix A(3, 3);

A.addTriplet(0, 0, 1);

A.addTriplet(0, 2, 2);

A.addTriplet(1, 1, 3);

A.addTriplet(2, 0, 4);

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

A.display();

SparseMatrix At = A.transpose(); cout << "\nTranspose of A:\n";

At.display();

SparseMatrix B(3, 3);

B.addTriplet(0, 1, 5);

B.addTriplet(1, 1, 7);

B.addTriplet(2, 2, 6);

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

B.display();

SparseMatrix C = A.add(B);

cout << "\nA + B (triplet form):\n";

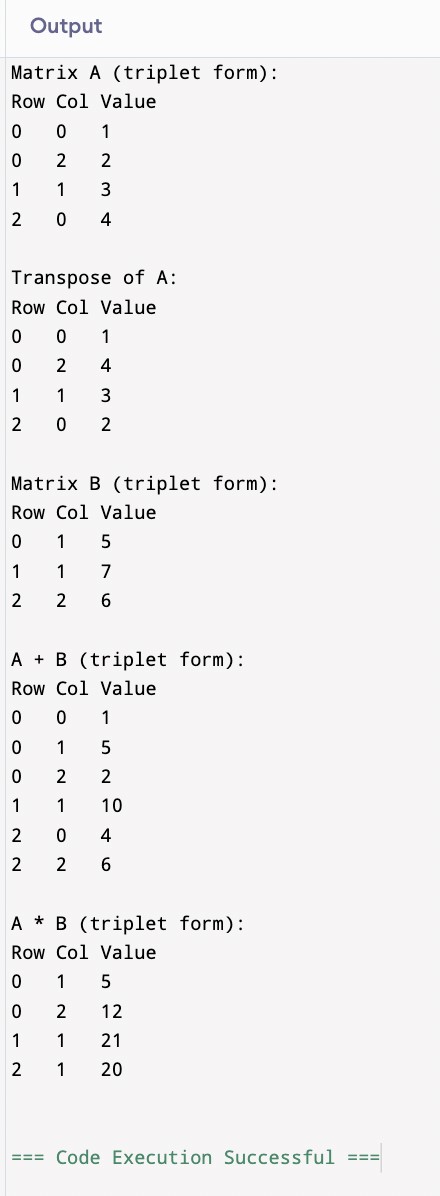
C.display();

SparseMatrix D = A.multiply(B); cout << "\nA \* B (triplet form):\n";

D.display();

return 0;

}



**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.**

7)

#include <iostream> using namespace std;

// Merge function to count inversions int merge(int arr[], int temp[], int left, int mid, int right) { int i = left, j = mid, k = left; int inv\_count = 0;

while (i < mid && j <= right) { if (arr[i] <= arr[j]) temp[k++] = arr[i++]; else { temp[k++] = arr[j++]; inv\_count += (mid - i); // Count inversions

}

}

while (i < mid) temp[k++] = arr[i++]; while (j <= right) temp[k++] = arr[j++]; for (int idx = left; idx <= right; idx++) arr[idx] = temp[idx]; return inv\_count;

}

// Recursive merge sort that counts inversions int mergeSort(int arr[], int temp[], int left, int right) { int inv\_count = 0; if (right > left) { int mid = (left + right) / 2; inv\_count += mergeSort(arr, temp, left, mid); inv\_count += mergeSort(arr, temp, mid + 1, right); inv\_count += merge(arr, temp, left, mid + 1, right);

}

return inv\_count;

}

int countInversions(int arr[], int n) { int \*temp = new int[n]; int result = mergeSort(arr, temp, 0, n - 1); delete[] temp; return result;

}

int main() { int arr[] = {2, 4, 1, 3, 5}; int n = sizeof(arr) / sizeof(arr[0]);

cout << "Number of inversions: " << countInversions(arr, n) << endl;

return 0;

}

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

8) #include <iostream> #include <algorithm> using namespace std;

int countDistinct(int arr[], int n) { if (n == 0) return 0;

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

int count = 1; // First element is always distinct

for (int i = 1; i < n; i++) { if (arr[i] != arr[i - 1]) { count++;

}

}

return count;

}

int main() {

int n;

cout << "Enter number of elements: "; cin >> n;

int arr[n];

cout << "Enter elements:\n";

for (int i = 0; i < n; i++) cin >> arr[i];

int distinctCount = countDistinct(arr, n); cout << "Number of distinct elements: " << distinctCount << endl;

return 0;

}

