Q-1: You are tasked to write a Program to find the sum of elements above and below the main diagonal of a matrix

Sample test case:

|  |
| --- |
| Input: {{1, 2, 3},  {4, 5, 6},  {7, 8, 9}}  Output:  Sum above diagonal: 11  Sum below diagonal: 19 |

Solution:

#include <iostream>

// Function to find the sum of elements above and below the main diagonal of a matrix using pointer arithmetic

void findDiagonalSums(int\* matrix, int size) {

int sumAboveDiagonal = 0;

int sumBelowDiagonal = 0;

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

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

if (j > i) {

sumAboveDiagonal += \*(matrix + i \* size + j);

} else if (j < i) {

sumBelowDiagonal += \*(matrix + i \* size + j);

}

}

}

std::cout << "Sum above diagonal: " << sumAboveDiagonal << std::endl;

std::cout << "Sum below diagonal: " << sumBelowDiagonal << std::endl;

}

int main() {

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

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

findDiagonalSums(reinterpret\_cast<int\*>(matrix), size);

return 0;

}

Q-2: Given a sorted array nums, remove the duplicates in-place such that each element appears only once and returns the new length.

Sample test case:

|  |
| --- |
| Input: nums = {1, 1, 2, 2, 3, 4, 4, 4, 5, 6}  Output: New length of the array: 6  Updated array: 1 2 3 4 5 6 |

Solution:

#include <iostream>

#include <vector>

using namespace std;

int removeDuplicates(vector<int>& nums) {

int n = nums.size();

if (n == 0)

return 0;

int unique\_count = 1;

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

if (nums[i] != nums[i - 1]) {

nums[unique\_count] = nums[i];

unique\_count++;

}

}

return unique\_count;

}

int main() {

vector<int> nums = {1, 7,8,8,9,10,11,11};

int new\_length = removeDuplicates(nums);

cout << "New length of the array: " << new\_length << endl;

cout << "Updated array: ";

for (int i = 0; i < new\_length; i++)

cout << nums[i] << " ";

cout << endl;

return 0;

}

Q-3: Given a rotated sorted array containing distinct elements, find the smallest element in the array. A rotated sorted array is an array that has been rotated at some pivot, so the elements that were originally in increasing order are now in a different order.

Sample test case:

|  |
| --- |
| Input: arr={4, 5, 6, -6, 0, 1, 2}  Output: Smallest element in the rotated sorted array: -6 |
| Solution: |

#include <iostream>

using namespace std;

// Function to find the smallest element in a rotated sorted array

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

int left = 0, right = n - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (arr[mid] > arr[right]) {

left = mid + 1;

} else if (arr[mid] < arr[right]) {

right = mid;

} else {

// Handle duplicate elements by moving the right pointer

right--;

}

}

return arr[left];

}

int main() {

int arr[] = {4, 5, 6, 7, 0, 1, 2};

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

// Find the smallest element in the rotated sorted array

int smallest = findSmallestElement(arr, n);

cout << "Smallest element in the rotated sorted array: " << smallest;

return 0;

}

Q-4: You are given two sorted arrays arr1 and arr2 of sizes n and m respectively. The task is to merge these two arrays in-place, such that the resulting array is sorted in non-decreasing order. You can assume that arr1 has enough extra space at the end to accommodate all elements from arr2. Implement a function mergeArrays that takes both arrays and their sizes as input and modifies arr1 to contain the merged and sorted elements.

Sample test case:

|  |
| --- |
| Input:  Array 1: 2 5 8  Array 2: 1 3 6 9  Output: Merged and sorted array: 1 2 3 5 6 8 9 |

Solution:

#include <iostream>

using namespace std;

void mergeArrays(int arr1[], int n, int arr2[], int m) {

// Start merging from the last elements of both arrays

int i = n - 1; // Index for arr1

int j = m - 1; // Index for arr2

int k = n + m - 1; // Index for the merged array

// Merge the arrays in descending order, from the largest elements to the smallest

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

if (arr1[i] >= arr2[j]) {

arr1[k] = arr1[i];

i--;

} else {

arr1[k] = arr2[j];

j--;

}

k--;

}

// If there are remaining elements in arr2, copy them to arr1

while (j >= 0) {

arr1[k] = arr2[j];

j--;

k--;

}

}

int main() {

int arr1[10] = {2, 5, 8};

int arr2[] = {1, 3, 6, 9};

int n = 3; // Size of arr1

int m = 4; // Size of arr2

// Print the initial arrays

cout << "Array 1: ";

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

cout << arr1[i] << " ";

cout << endl;

cout << "Array 2: ";

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

cout << arr2[i] << " ";

cout << endl;

// Merge the arrays and modify arr1 in-place

mergeArrays(arr1, n, arr2, m);

// Print the merged and sorted array

cout << "Merged and sorted array: ";

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

cout << arr1[i] << " ";

cout << endl;

return 0;

}