**ASSIGNMENT-4**

**Question 1** Given three integer arrays arr1, arr2 and arr3 **sorted** in **strictly increasing** order, return a sorted array of **only** the integers that appeared in **all** three arrays.

**Example 1:**

Input: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

Output: [1,5]

**Explanation:** Only 1 and 5 appeared in the three arrays.

**ANS:**

#include <stdio.h>

#define MAX\_LENGTH 100

int main() {

int arr1[MAX\_LENGTH] = {1, 2, 3, 4, 5};

int arr2[MAX\_LENGTH] = {1, 2, 5, 7, 9};

int arr3[MAX\_LENGTH] = {1, 3, 4, 5, 8};

int n1 = 5, n2 = 5, n3 = 5;

int result[MAX\_LENGTH];

int resultIndex = 0;

int i1 = 0, i2 = 0, i3 = 0;

while (i1 < n1 && i2 < n2 && i3 < n3) {

if (arr1[i1] == arr2[i2] && arr2[i2] == arr3[i3]) {

result[resultIndex++] = arr1[i1];

i1++;

i2++;

i3++;

} else if (arr1[i1] < arr2[i2]) {

i1++;

} else if (arr2[i2] < arr3[i3]) {

i2++;

} else {

i3++;

}

}

printf("Output: ");

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

printf("%d ", result[i]);

}

printf("\n");

return 0;

}

**Question 2**

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

* answer[0] *is a list of all* ***distinct*** *integers in* nums1 *which are* ***not*** *present in* nums2\*.\*
* answer[1] *is a list of all* ***distinct*** *integers in* nums2 *which are* ***not*** *present in* nums1.

**Note** that the integers in the lists may be returned in **any** order.

**Example 1:**

**Input:** nums1 = [1,2,3], nums2 = [2,4,6]

**Output:** [[1,3],[4,6]]

**Explanation:**

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

**ANS:**

#include <stdio.h>

#define MAX\_LENGTH 100

int main() {

int nums1[MAX\_LENGTH] = {1, 2, 3};

int nums2[MAX\_LENGTH] = {2, 4, 6};

int n1 = 3, n2 = 3;

int distinct1[MAX\_LENGTH];

int distinct2[MAX\_LENGTH];

int distinctIndex1 = 0, distinctIndex2 = 0;

for (int i = 0; i < n1; i++) {

int isPresent = 0;

for (int j = 0; j < n2; j++) {

if (nums1[i] == nums2[j]) {

isPresent = 1;

break;

}

}

if (!isPresent) {

distinct1[distinctIndex1] = nums1[i];

distinctIndex1++;

}

}

for (int i = 0; i < n2; i++) {

int isPresent = 0;

for (int j = 0; j < n1; j++) {

if (nums2[i] == nums1[j]) {

isPresent = 1;

break;

}

}

if (!isPresent) {

distinct2[distinctIndex2] = nums2[i];

distinctIndex2++;

}

}

printf("[");

for (int i = 0; i < distinctIndex1; i++) {

printf("%d", distinct1[i]);

if (i != distinctIndex1 - 1) {

printf(",");

}

}

printf("],[");

for (int i = 0; i < distinctIndex2; i++) {

printf("%d", distinct2[i]);

if (i != distinctIndex2 - 1) {

printf(",");

}

}

printf("]");

return 0;

}

**Question 3** Given a 2D integer array matrix, return the ***transpose*** of matrix.

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

**Example 1:**

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[1,4,7],[2,5,8],[3,6,9]]

**ANS:**

#include <stdio.h>

#define ROWS 3

#define COLS 3

int main() {

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

// Print the original matrix

printf("Original Matrix:\n");

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

// Transpose the matrix

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

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

// Swap elements (i, j) and (j, i)

int temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

// Print the transposed matrix

printf("Transposed Matrix:\n");

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

return 0;

}

**Question 4** Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2), ..., (an, bn) such that the sum of min(ai, bi) for all i is **maximized**. Return the maximized sum.

**Example 1:**

Input: nums = [1,4,3,2]

Output: 4

**Explanation:** All possible pairings (ignoring the ordering of elements) are:

1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4

So the maximum possible sum is 4.

**ANS:**

#include <stdio.h>

#define SIZE 4

int main() {

int nums[SIZE] = {1, 4, 3, 2};

// Sort the array in ascending order

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

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

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

int temp = nums[j];

nums[j] = nums[j + 1];

nums[j + 1] = temp;

}

}

}

// Calculate the sum of minimum elements

int sum = 0;

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

sum += nums[i];

}

// Print the maximized sum

printf("Maximized Sum: %d\n", sum);

return 0;

}

**Question 5** You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return the number of ***complete rows*** of the staircase you will build.

**Example 1:**

**Input:** n = 5

**Output:** 2

**Explanation:** Because the 3rd row is incomplete, we return 2.

**ANS:**

#include <stdio.h>

int countCompleteRows(int n) {

int completeRows = 0;

int i = 1;

while (n >= i) {

n -= i;

completeRows++;

i++;

}

return completeRows;

}

int main() {

int n = 5;

int completeRows = countCompleteRows(n);

printf("Number of complete rows: %d\n", completeRows);

return 0;

}

**Question 6** Given an integer array nums sorted in **non-decreasing** order, return an array of ***the squares of each number*** sorted in non-decreasing order.

**Example 1:**

Input: nums = [-4,-1,0,3,10]

Output: [0,1,9,16,100]

**Explanation:** After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100]

**ANS:**

#include <stdio.h>

void sortSquares(int\* nums, int numsSize) {

int i, j;

// Find the first positive number or zero in the array

for (i = 0; i < numsSize; i++) {

if (nums[i] >= 0) {

break;

}

}

// Squaring the negative numbers in reverse order

int left = i - 1;

int right = i;

int index = 0;

int result[numsSize];

while (left >= 0 && right < numsSize) {

if (nums[left] \* nums[left] < nums[right] \* nums[right]) {

result[index++] = nums[left] \* nums[left];

left--;

} else {

result[index++] = nums[right] \* nums[right];

right++;

}

}

// Copy the remaining elements

while (left >= 0) {

result[index++] = nums[left] \* nums[left];

left--;

}

while (right < numsSize) {

result[index++] = nums[right] \* nums[right];

right++;

}

// Copy the sorted squares back to the original array

for (i = 0; i < numsSize; i++) {

nums[i] = result[i];

}

}

int main() {

int nums[] = {-4, -1, 0, 3, 10};

int numsSize = sizeof(nums) / sizeof(nums[0]);

int i;

printf("Input array: ");

for (i = 0; i < numsSize; i++) {

printf("%d ", nums[i]);

}

printf("\n");

sortSquares(nums, numsSize);

printf("Sorted squares: ");

for (i = 0; i < numsSize; i++) {

printf("%d ", nums[i]);

}

printf("\n");

return 0;

}

**Question 7** You are given an m x n matrix M initialized with all 0's and an array of operations ops, where ops[i] = [ai, bi] means M[x][y] should be incremented by one for all 0 <= x < ai and 0 <= y < bi.

Count and return the number of maximum integers in the matrix after performing all the operations

**Example 1:**

**Input:** m = 3, n = 3, ops = [[2,2],[3,3]]

**Output:** 4

**Explanation:** The maximum integer in M is 2, and there are four of it in M. So return 4.

**ANS:**

#include <stdio.h>

int maxCount(int m, int n, int\*\* ops, int opsSize, int\* opsColSize) {

int i;

int minRow = m;

int minCol = n;

// Find the minimum row and column value from the operations

for (i = 0; i < opsSize; i++) {

if (ops[i][0] < minRow) {

minRow = ops[i][0];

}

if (ops[i][1] < minCol) {

minCol = ops[i][1];

}

}

return minRow \* minCol;

}

int main() {

int m = 3;

int n = 3;

int opsSize = 2;

int opsColSize = 2;

int ops[2][2] = {{2, 2}, {3, 3}};

int result = maxCount(m, n, ops, opsSize, &opsColSize);

printf("Number of maximum integers: %d\n", result);

return 0;

}

**Question 8**

Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].

Return the array in the form [x1,y1,x2,y2,...,xn,yn].

**Example 1:**

**Input:** nums = [2,5,1,3,4,7], n = 3

**Output:** [2,3,5,4,1,7]

**Explanation:** Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].

**ANS:**

#include <stdio.h>

void shuffleArray(int\* nums, int numsSize, int n) {

int result[numsSize]; // Create a new array to store the shuffled elements

int i, j, k;

// Iterate through the array and shuffle the elements

for (i = 0, j = 0, k = n; i < numsSize; i += 2, j++, k++) {

result[i] = nums[j]; // Store xi at even indices

result[i + 1] = nums[k]; // Store yi at odd indices

}

// Copy the shuffled array back to the original array

for (i = 0; i < numsSize; i++) {

nums[i] = result[i];

}

}

int main() {

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

int numsSize = sizeof(nums) / sizeof(nums[0]);

int n = numsSize / 2;

shuffleArray(nums, numsSize, n);

printf("Shuffled array: ");

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

printf("%d ", nums[i]);

}

printf("\n");

return 0;

}