**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.

Sol:

#include <bits/stdc++.h>

using namespace std;

void findCommon(int ar1[], int ar2[], int ar3[], int n1,

                int n2, int n3)

{

    int i = 0, j = 0, k = 0;

    while (i < n1 && j < n2 && k < n3) {

        if (ar1[i] == ar2[j] && ar2[j] == ar3[k]) {

            cout << ar1[i] << " ";

            i++;

            j++;

            k++;

        }

        else if (ar1[i] < ar2[j])

            i++;

        else if (ar2[j] < ar3[k])

            j++;

        else

            k++;

    }

}

int main()

{

    int ar1[] = {1,2,3,4,5};

    int ar2[] = {1,2,5,7,9};

    int ar3[] = {1,3,4,5,8};

    int n1 = sizeof(ar1) / sizeof(ar1[0]);

    int n2 = sizeof(ar2) / sizeof(ar2[0]);

    int n3 = sizeof(ar3) / sizeof(ar3[0]);

    cout << "Common Elements are ";

    findCommon(ar1, ar2, ar3, n1, n2, n3);

    return 0;

}

Output: Common Elements are 1 5

**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].

Sol:

#include <bits/stdc++.h>

using namespace std;

 vector<vector<int>> findDifference(vector<int>& nums1, vector<int>& nums2) {

        vector<int> v1, v2;

        unordered\_set<int> s1(nums1.begin(), nums1.end());

        unordered\_set<int> s2(nums2.begin(), nums2.end());

        for(int num: s1) if(s2.find(num)==s2.end()) v1.push\_back(num);

        for(int num: s2) if(s1.find(num)==s1.end()) v2.push\_back(num);

        return {v1, v2};}

int main()

{

    vector<int> arr = {1,2,3};

    vector<int> arr1 = {2,4,6};

    vector<vector<int>> ans = findDifference(arr,arr1);

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

    {

        for (int j = 0; j < ans[i].size(); j++)

        {

            cout << ans[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

output: 3 1

6 4

**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]]

Sol: #include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> transpose(vector<vector<int>>& matrix) {

        int n = matrix.size();

        int m = matrix[0].size();

        vector<vector<int>> res(m,vector<int> (n, 0));

        for (int i = 0; i < n; i++) {

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

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

            }

        }

        return res;

    }

int main() {

    vector<vector<int>> matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    vector<vector<int>> transposedMatrix = transpose(matrix);

    for (const auto& row : transposedMatrix) {

        for (int num : row) {

            cout << num << " ";

        }

        cout << endl;

    }

    return 0;

}

Output: 1 4 7

2 5 8

3 6 9

**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.

Sol: #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int arrayPairSum(const vector<int>& nums) {

   vector<int> sortedNums = nums;

 sort(sortedNums.begin(), sortedNums.end());

    int sum = 0;

    for (int i = 0; i < sortedNums.size(); i += 2) {

        sum += sortedNums[i];

    }

    return sum;

}

int main() {

    vector<int> nums = {1, 4, 3, 2};

    int maxSum = arrayPairSum(nums);

   cout << "Maximized sum: " << maxSum << endl;

    return 0;

}

Sol: Maximized sum: 4

**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.

Sol: #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int arrangeCoins(int n) {

       long long lo = 1 ;

       long long  hi = n ;

       while (lo <= hi ){

           long long  k = lo + (hi - lo)/2 ;

           long long m = k\*(k+1)/2;

           if (m== n ){

               return k ;

           }

           if (m > n )  hi =k -1 ;

           else  lo = k + 1 ;

       }

       return hi ;

    }

int main()

{

    int n = 5;

    cout<<arrangeCoins(n)<<endl;

}

Output:2

**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]

Sol: #include <iostream>

#include <vector>

#include <algorithm>

#include <bits/stdc++.h>

using namespace std;

 vector<int> sortedSquares(vector<int>& nums) {

        for(auto i=0;i<nums.size();i++) {

            nums[i] \*= nums[i];

        }

        sort(nums.begin(),nums.end());

        return nums;

    }

int main() {

    vector<int> nums = {-4, -1, 0, 3, 10};

    vector<int> squaredSorted = sortedSquares(nums);

    for (int num : squaredSorted) {

        cout << num << " ";

    }

   cout <<endl;

    return 0;

}

Output: 0 1 9 16 100

**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.

Sol: #include <iostream>

#include <vector>

using namespace std;

int maxCount(int m, int n, vector<vector<int>>& ops) {

    int minRow = m;

    int minCol = n;

    for (const auto& op : ops) {

        minRow = min(minRow, op[0]);

        minCol = min(minCol, op[1]);

    }

    return minRow \* minCol;

}

int main() {

    int m = 3;

    int n = 3;

    vector<vector<int>> ops = {{2, 2}, {3, 3}};

    int maxIntegers = maxCount(m, n, ops);

    cout << "Number of maximum integers: " << maxIntegers <<endl;

    return 0;

}

Sol: Number of maximum integers: 4

**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].

Sol: #include <iostream>

#include <vector>

using namespace std;

vector<int> shuffleArray(const vector<int>& nums, int n) {

    vector<int> result(2 \* n);

    int i = 0;

    int j = n;

    int k = 0;

    while (i < n) {

        result[k++] = nums[i++];

        result[k++] = nums[j++];

    }

    return result;

}

int main() {

   vector<int> nums = {2, 5, 1, 3, 4, 7};

    int n = 3;

    vector<int> shuffledArray = shuffleArray(nums, n);

    for (int num : shuffledArray) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

Output: 2 3 5 4 1 7