Q.1Given 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.

vector**<int>** arraysIntersection(vector**<int>&** A, vector**<int>&** B, vector**<int>&** C) {

**int** i **=** 0, j **=** 0, k **=** 0, R **=** A.size(), S **=** B.size(), T **=** C.size();

vector**<int>** ans;

**while** (i **<** R **&&** j **<** S **&&** k **<** T) {

**int** a **=** A[i], b **=** B[j], c **=** C[k];

**if** (a **==** b **&&** b **==** c) {

ans.push\_back(a);

**++**i, **++**j, **++**k;

} **else** {

**int** mx **=** max({ a, b, c });

**if** (a **<** mx) **++**i;

**if** (b **<** mx) **++**j;

**if** (c **<** mx) **++**k;

}

}

**return** ans;

}

**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].

</aside>

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

        unordered\_set<int>set1(nums1.begin(),nums1.end());

        unordered\_set<int>set2(nums2.begin(),nums2.end());

        vector<int>temp1,temp2;

        for(int i:set1)

        {

            if(set2.count(i)==0)

            {

                temp1.push\_back(i);

            }

        }

        for(int i:set2)

        {

            if(set1.count(i)==0)

            {

                temp2.push\_back(i);

            }

        }

        return {temp1,temp2};

    }

**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]]

</aside>

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

        int m = matrix.size();

        int n=matrix[0].size();

        vector<vector<int>>ans(n,vector<int>(m));

        for(int i=0;i<m;i++)

        {

            for(int j=0;j<n;j++)

            {

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

            }

        }

        return ans;

    }

**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.

int arrayPairSum(vector<int>& nums) {

        int n = nums.size();

        int sum = 0;

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

        for(int i=1;i<n;i+=2)

        {

            sum += min(nums[i],nums[i-1]);

        }

        return sum;

    }

<aside> 💡 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]

</aside>

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

       int n = nums.size();

       int left=0;

       int right = n-1;

       vector<int>ans(n);

        int i=n-1;

       while(left<=right)

       {

           if(abs(nums[left]) < abs(nums[right]))

           {

               ans[i] = nums[right]\*nums[right];

               right--;

               i--;

           }

           else

           {

               ans[i] = nums[left]\*nums[left];

               left++;

               i--;

           }

       }

       return ans;

    }

**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].

</aside>

 vector<int> shuffle(vector<int>& nums, int n) {

        vector<int>ans;

        for(int i=0;i<n;i++)

        {

            ans.push\_back(nums[i]);

            ans.push\_back(nums[n+i]);

        }

        return ans;

    }