Assignment-4

31 May 2023 00:03

💡 **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.

```
#include <bits/stdc++.h>
vector<int> findCommonElements(vector<int> &a, vector<int> &b, vector<int> &c)
{
    // Write your code here
int i=0;
int j=0;
int k=0;
vector<int> ans;
while(i<a.size() && j<b.size() && k<c.size()){
if(a[i]==b[j] and b[j]==c[k]){
ans.push_back(a[i]);
i++;
j++;
k++;
}
else if(a[i]<b[j]){
i++;
}
else if(b[j]<c[k]){
j++;
            } else {
              k++;
            }
            int prev1 = a[i - 1];
            while (a[i] == prev1)
              i++;

            int prev2 = b[j - 1];
            while (b[j] == prev2)
              j++;

            int prev3 = c[k - 1];
            while (c[k] == prev3)
              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].

```
class Solution {
public:
    vector<vector<int>> findDifference(vector<int>& nums1, vector<int>& nums2) {
        set<int> s1,s2;

        for(auto it:nums1){
            s1.insert(it);
        }
        for(auto it:nums2){
            s2.insert(it);
        }

        vector<vector<int>> ans(2);

        for(auto it:s1){
            if(s2.count(it)==0)
                ans[0].push_back(it);

        }
        for(auto it:s2){
            if(s1.count(it)==0)
                ans[1].push_back(it);
        }
        return ans;
    }
};
```

💡 **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]]

```
class Solution {
public:
    vector<vector<int>> transpose(vector<vector<int>>& matrix) {
        int m=matrix.size();
        for(int i=0;i<m;i++){
            for(int j=0;j<=i;j++){
                swap(matrix[i][j],matrix[j][i]);
            }
        }
        return matrix;
    }
};
```

💡 **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.

```
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        int sum=0;
        sort(nums.begin(),nums.end());
        for(int i=0;i<nums.size()-1;i+=2){
            sum+=nums[i];
        }
        return sum;
    }
};
```

💡 **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]

```
class Solution {
   public int[] sortedSquares(int[] a) {
      int i=0;
      int j=a.length;
     for(i=0;i<j;i++)
     {
        a[i]=a[i]*a[i];
     }
      Arrays.sort(a);
      return a;

      }
}
```

💡 **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].

```
class Solution {
public:
   vector<int> shuffle(vector<int>& nums, int n) {
      vector<int>ans(2*n);
      int j=0;
      int k=nums.size()/2;
      int i=0;
      while(i<nums.size()){
         ans[i++]=nums[j++];
         ans[i++]=nums[k++];
      }
       return ans;
   }
  };
```

💡 **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.

```
class Solution {
public:
   int arrangeCoins(int n) {


      int i=1;
      int stairs=0;
      while(n>=i){
         n-=i++;
         stairs++;
      }
      return stairs;
   }
};
```

💡 **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.

```
class Solution {
public:
    int maxCount(int m, int n, vector<vector<int>>& ops) {
        int minRow=m;
        int minCol=n;

        for(int i=0;i<ops.size();i++){
            minRow=min(minRow,ops[i][0]);
            minCol=min(minCol,ops[i][1]);
        }
        return minRow * minCol;
    }
};
```