**Question 1**

Convert 1D Array Into 2D Array

You are given a **0-indexed** 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with  m rows and n columns using **all** the elements from original.

The elements from indices 0 to n - 1 (**inclusive**) of original should form the first row of the constructed 2D array, the elements from indices n to 2 \* n - 1 (**inclusive**) should form the second row of the constructed 2D array, and so on.

Return an m x n 2D array constructed according to the above procedure, or an empty 2D array if it is impossible.

**Example 1:**

**Input:** original = [1,2,3,4], m = 2, n = 2

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

**Explanation:** The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

Sol: #include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> construct2DArray(vector<int>& original, int m, int n) {

        vector <int> num;

        vector<vector<int>> dum;

        if(original.size()!=m\*n){

            return dum;

        }

             for(int i=0;i<m;i++){

            num.clear();

            for(int j=0;j<n;j++){

                num.push\_back(original[n\*i+j]);

            }

                dum.push\_back(num);

        }

        return dum;

        }

int main() {

    vector<int> original = {1,2,3,4};

    int m=2;

    int n = 2;

    vector<vector<int>> result = construct2DArray(original,m,n);

    for (const auto& row : result) {

        for (int num : row) {

            cout << num << " ";

        }

        cout << endl;

    }

    return 0;

}

Output:

1 2

3 4

**Question 2**

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 3** 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 4**

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 5**

Given two integer arrays arr1 and arr2, and the integer d, return the distance value between the two arrays.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

**Example 1:**

**Input:** arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2

**Output:** 2

**Explanation:**

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

**|8-10|=2 <= d=2**

**|8-9|=1 <= d=2**

|8-1|=7 > d=2

**|8-8|=0 <= d=2**

Sol: #include <bits/stdc++.h>

using namespace std;

int findTheDistanceValue(vector<int>& arr1, vector<int>& arr2, int d) {

    int ans = 0;

    sort(arr2.begin(), arr2.end());

    for(int i=0;i<arr1.size();i++){

        int it = lower\_bound(arr2.begin(), arr2.end(), arr1[i]) - arr2.begin();

        bool isIt = false;

        if(it<arr2.size() && abs(arr2[it] - arr1[i]) <= d)isIt = true;

        if(it != 0 && abs(arr2[it-1] - arr1[i]) <= d)isIt = true;

        if(!isIt)

            ans++;

    }

    return ans;

}

int main(){

    vector<int> arr1 = {4,5,8};

    vector<int> arr2 = {10,9,1,8};

    int d = 2;

    cout<<findTheDistanceValue(arr1, arr2, d);

}

Output:2

**Question 6**

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return an array of all the integers that appears ***twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

**Example 1:**

**Input:** nums = [4,3,2,7,8,2,3,1]

**Output:**

[2,3]

Sol:

#include <bits/stdc++.h>

using namespace std;

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

        vector<int>v;

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

        for(int i=0;i<nums.size()-1;i++)

        {

           if(nums[i]==nums[i+1])

           {

               v.push\_back(nums[i]);

           }

        }

        return v;

    }

int main()

{

    vector<int> numRay = {4,3,2,7,8,2,3,1};

    vector<int> numRay1 = findDuplicates(numRay);

    for (int num : numRay1) {

        cout << num << " ";

    }

   cout <<endl;

    return 0;

}

output: 2 3

**Question 7**

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

* [4,5,6,7,0,1,2] if it was rotated 4 times.
* [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that **rotating** an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

**Example 1:**

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

**Output:** 1

**Explanation:**

The original array was [1,2,3,4,5] rotated 3 times.

Sol: #include <bits/stdc++.h>

using namespace std;

int findMin(vector<int>& nums) {

         int l=0;

         int h=nums.size()-1;

         if(l+1==h)

         return min(nums[l],nums[h]);

         while(l<h)

         {

             int mid=l+(h-l)/2;

             if(mid>0&&nums[mid]<nums[mid-1]&&nums[mid]<nums[mid+1])

             return nums[mid];

             else if(nums[l]<nums[mid]&&nums[h]<nums[mid])

             l=mid;

             else

             h=mid;

         }

         if(l>0&&nums[l]>nums[l-1]&&nums[l]>nums[l+1])

         return nums[l+1];

         return nums[l];

    }

int main()

{

    vector<int> numRay = {3,4,5,1,2};

    cout<<findMin(numRay)<<endl;

    return 0;

}

Output:1

**Question 8**

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

**Example 1:**

**Input:** changed = [1,3,4,2,6,8]

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

**Explanation:** One possible original array could be [1,3,4]:

* Twice the value of 1 is 1 \* 2 = 2.
* Twice the value of 3 is 3 \* 2 = 6.
* Twice the value of 4 is 4 \* 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

Sol: #include <bits/stdc++.h>

using namespace std;

vector<int> findOriginalArray(vector<int>& changed) {

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

        vector<int> orig;

        queue<int> dbl;

        for(auto n : changed) {

            if(!dbl.empty() && dbl.front() == n)

                dbl.pop();

            else {

                orig.push\_back(n);

                dbl.push(2\*n);

            }

        }

        if(orig.size() \* 2 != changed.size())

            orig.clear();

        return orig;

    }

int main()

{

    vector<int> numRay = {1,3,4,2,6,8};

    vector<int> numRay1 = findOriginalArray(numRay);

    for (int num : numRay1) {

        cout << num << " ";

    }

   cout <<endl;

    return 0;

}

Output: 1 3 4