<aside> 💡 **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.

Ans:

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

        if(m\*n != original.size())

        {

            return {};

        }

        vector<vector<int>>ans;

        vector<int>row;

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

        {

            row.push\_back(original[i]);

            if(row.size() == n)

            {

                ans.push\_back(row);

                row.clear();

            }

        }

        return ans;

    }

Q.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.

**Input:** n = 5

**Output:** 2

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

 int arrangeCoins(int n) {

        if(n==1)return 1;

        if(n==2)return 1;

        if(n==3)return 2;

        long int coin = 0;

        for(long int i=1;i<=n;i++)

        {

            coin+=i;

            if(coin > n)

                return i-1;

        }

    return 0;

    }

Q.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].

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 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].

</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 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**

</aside>

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

        int cnt=0;

        for(auto it:arr1)

        {

            int flag = 0;

            for(auto it1:arr2)

            {

                if(abs(it-it1) > d)

                {

                    flag = 1;

                }

                else

                {

                    flag = 0;

                    break;

                }

            }

            if(flag == 1)

            {

                cnt++;

            }

        }

        return cnt;

    }

Using binary search:

 bool isValid(vector<int>&arr, int target, int d)

    {

        int left = 0;

        int right = arr.size()-1;

        while(left <= right)

        {

            int mid = (left+right)/2;

            if(abs(arr[mid] - target) <= d)

            {

                return false;

            }

            else if(arr[mid] < target)

            {

                left = mid+1;

            }

            else

            {

                right = mid-1;

            }

        }

        return true;

    }

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

        int cnt=0;

        sort(arr2.begin(),arr2.end());

        for(auto num:arr1)

        {

            if(isValid(arr2,num,d))

                cnt++;

        }

        return cnt;

    }

**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]

</aside>

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

        unordered\_map<int,int>mpp;

        for(auto num:nums)

        {

            mpp[num]++;

        }

        vector<int>temp;

        for(auto i:mpp)

        {

            if(i.second == 2)

            {

                temp.push\_back(i.first);

            }

        }

        return temp;

    }

<aside> 💡 **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.

</aside>

 int findMin(vector<int>& nums) {

        int n = nums.size();

        int left = 0;

        int right = n-1;

        while(left < right)

        {

            int mid = (left+right)/2;

            if(nums[mid] > nums[right])

            {

                left = mid+1;

            }

            else

            {

                right = mid;

            }

        }

        return nums[left];

    }