Assignment 5

**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*.

**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 =

class Solution {

public int[][] construct2DArray(int[] original, int m, int n) {

if(original.length!= m\*n) return new int [][]{};

int[][]ans =new int[m][n];

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

{

ans[i/n][i%n]=original[i];

}

return ans;

}

}

**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*.

**Input:** n = 5

**Output:** 2

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

ANS=

class Solution {

public int arrangeCoins(int n) {

long i=0;

for(i=1;i\*(i+1)/2<=n;i++);

return (int)i-1;

}

}

**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].

ANS =

int distinctSubseqII(string S) {

long endsWith[26] = {}, mod = 1e9 + 7;

for (char c : S)

endsWith[c - 'a'] = accumulate(begin(endsWith), end(endsWith), 1L) % mod;

return accumulate(begin(endsWith), end(endsWith), 0L) % mod;

}

**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.

ANS =

class Solution {

public:

int findMin(vector<int>& nums) {

return \*min\_element(nums.begin(),nums.end());

}

};

**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]

ANS=

class Solution {

public:

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

int n= changed.size();

if (n&1)return {};

vector<int> ans;

map<int,int> mp;

sort(changed.begin(),changed.end(),greater<int>());

for (int i=0; i<n; i++){

if (mp.find(changed[i]\*2)!=mp.end()){

if (mp[changed[i]\*2]>0){mp[changed[i]\*2]--;}

ans.push\_back(changed[i]);

if (mp[changed[i]\*2]==0) {mp.erase(changed[i]\*2);}

}

else {mp[changed[i]]++;}

}

for (auto val: mp){

if (val.second!=0)return {};

}

return ans;

}

};