💡 Question 1

A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

- s[i] == 'I' if perm[i] < perm[i + 1], and
- s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return **any of them**.

Example 1:

Input: s = "IDID"

Output:

[0,4,1,3,2]

```
class Solution {
public:
    vector<int> diStringMatch(string s) {
        int i=0,j=s.length();
        vector <int> v;
        for(int it:s){
            if(it=='I'){
                v.push_back(i);
                i++;
            }
            else{
                 v.push_back(j);
                j--;
            }
        }
        v.push_back(i);
        return v;

    }
};
```

💡 Question 2

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.

You must write a solution in O(log(m * n)) time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

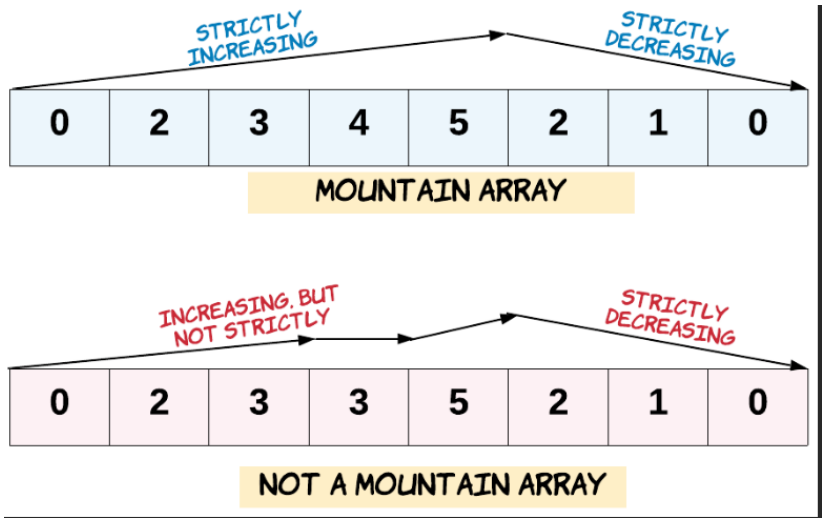
```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size();
        if (m == 0) {
            return false;
        }
        int n = matrix[0].size();

        int low = 0, high = m * n - 1;
        int midIdx, midElement, rowIdx, colIdx;
        while (low <= high) {
            midIdx = low + (high - low) / 2;
            rowIdx = midIdx / n;
            colIdx = midIdx % n;
            midElement = matrix[rowIdx][colIdx];

            if (target == midElement) {
                return true;
            } else {
                if (target < midElement) {
                    high = midIdx - 1;
                } else {
                    low = midIdx + 1;
                }
            }
        }

        return false;
    }
};
```

💡 Question 3

Given an array of integers arr, return *true if and only if it is a valid mountain array*.

Recall that arr is a mountain array if and only if:

- arr.length >= 3
- There exists some i with 0 < i < arr.length - 1 such that:
    - arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
    - arr[i] > arr[i + 1] > ... > arr[arr.length - 1]



Example 1:

Input: arr = [2,1]

Output: false

Example 2:

Input: arr = [3,5,5]

Output: false

```cpp
class Solution {
public:
    bool validMountainArray(vector<int>& arr) {

    int n=arr.size();
    if(n<3)
        return false;

    int count=0;
    for(int i=1;i<n-1;i++)
    {
        if(arr[i]<arr[i-1]&&arr[i]<arr[i+1] || arr[i]==arr[i-1])
            return false;
        if(arr[i]>arr[i-1]&&arr[i]>arr[i+1] )
            count++;
    }
    if(count!=1)
        return false;
    return true;

    }
};
```

💡 Question 4

Given a binary array nums, return *the maximum length of a contiguous subarray with an equal number of* 0 *and* 1.

Example 1:

Input: nums = [0,1]

Output: 2

Explanation:

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

```cpp
class Solution {
public:
    int findMaxLength(vector<int>& nums) {
        unordered_map<int,int> mp;
        int x=0;
        mp[0]=-1;
        int ans=0;
        for(int i=0;i<nums.size();i++)
        {
            if(nums[i]==0)
                x--;
            else
                x++;
            if(mp.find(x)!=mp.end())
                ans=max(ans,i-mp[x]);
            else
                mp[x]=i;
        }
        return ans;
    }
};
```

💡 Question 5

The **product sum** of two equal-length arrays a and b is equal to the sum of a[i] * b[i] for all 0 <= i < a.length (**0-indexed**).

- For example, if a = [1,2,3,4] and b = [5,2,3,1], the **product sum** would be 1*5* + *2*2 + 3*3* + *4*1 = 22.

Given two arrays nums1 and nums2 of length n, return *the **minimum product sum** if you are allowed to **rearrange** the **order** of the elements in* nums1.

Example 1:

Input: nums1 = [5,3,4,2], nums2 = [4,2,2,5]

Output: 40

```
class Solution{
    public:
    long long int minValue(int a[], int b[], int n)
    {
        // Your code goes here
        sort(a,a+n,greater<int>());
        sort(b,b+n);

        long long sum =0;
        for(int i=0;i<n;i++)
        {
          sum= sum+a[i]*b[i];

        }
        return sum;
    }
};
```

💡 Question 6

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.

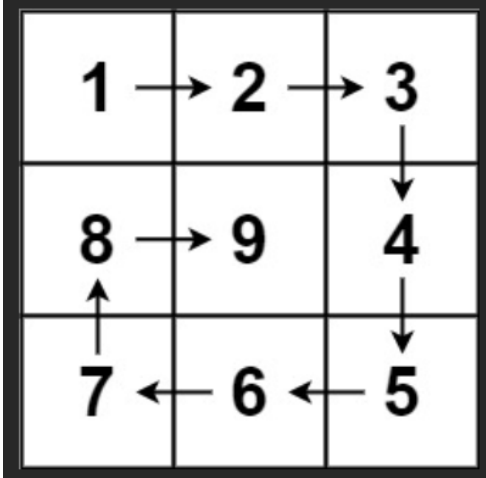
Other original arrays could be [4,3,1] or [3,1,4].

```cpp
class Solution {
public:
    vector<int> findOriginalArray(vector<int>& c) {
        vector<int>ans;
        vector<int>w;
        if(c.size()%2==1)return w;

        sort(c.begin(),c.end());
        unordered_map<int,int>map;
        for(auto i : c) map[i]++;

        for(auto i : c) {
            int cur = i;
            if(map[cur]){
                if(map[cur*2]==0) return w;
                ans.push_back(cur);
                map[cur]--;
                map[cur*2]--;
            }
        }
        return ans;
    }
};
```

💡 Question 7

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

Example 1:



Input: n = 3
Output: [[1,2,3],[8,9,4],[7,6,5]]

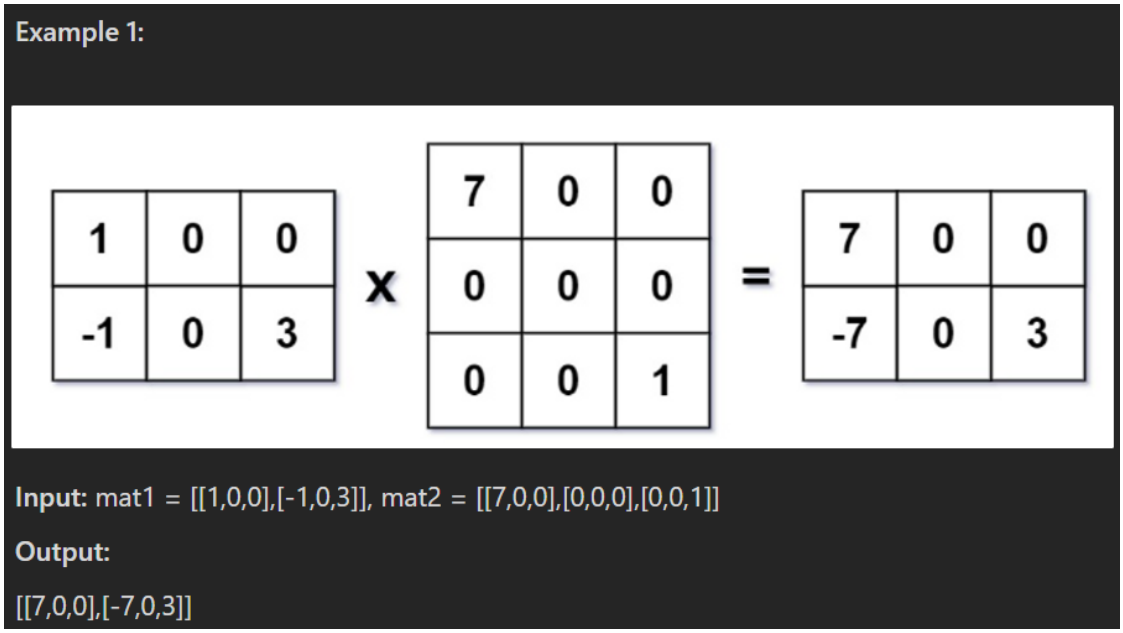
```
class Solution {

public:

    vector<vector<int>> generateMatrix(int n) {

        vector<vector<int>>res(n,vector<int>(n));

        int top=0;int left=0;

        int right=n-1;int bottom=n-1;

        int a=1;

        while(top<=bottom && left<=right){

            for(int i=left;i<=right;i++){//top



                res[top][i]=a;

                a++;

            }

            top++;
```

```
            for(int i=top;i<=bottom;i++){//right
                res[i][right]=a;
                a++;
            }
            right--;
            if(top<=bottom){
                for(int i=right;i>=left;i--){//bottom in reverse
                res[bottom][i]=a;
                a++;
            }
            bottom--;
            }
            if(left<=right){
                for(int i=bottom;i>=top;i--){//left in reverse
                res[i][left]=a;
                a++;
            }
            left++;
            }
        }
        return res;

    }
};
```

💡 Question 8

Given two sparse matrices mat1 of size m x k and mat2 of size k x n, return the result of mat1 x mat2. You may assume that multiplication is always possible.

Example 1:




```
vector<vector<int>> multiplyMatrices(vector<vector<int>>&
mat1, vector<vector<int>>& mat2) {

    int n = mat1.size();

    int m = mat2[0].size();

    int q = mat1[0].size();

    int w = mat2.size();



    vector<vector<int>> a(n, vector<int>(m, 0));
```

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

        for(int j = 0; j < m; j++) {

            for(int k = 0; k < q; k++) {

                a[i][j] += mat1[i][k] * mat2[k][j];

            }

        }

    }

    return a;

}
```