**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]

vector<int> diStringMatch(string s) {

        int left = 0;

        int right = s.size();

        vector<int>v;

        for(int i=0;i<s.size();i++)

        {

        if(s[i]=='I')

        {

            v.push\_back(left);

            left++;

        }

        else

        {

            v.push\_back(right);

            right--;

        }

        }

        if(s[s.size()-1] == 'I')

        {

            v.push\_back(left);

        }

        else

        {

            v.push\_back(right);

        }

        return v;

    }

<aside> 💡 **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:**

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

**Output:** true

bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int rows = matrix.size(),

            cols = matrix[0].size(),

            row = 0, col = cols - 1;

        while (row < rows && col > -1) {

            int cur = matrix[row][col];

            if (cur == target) return true;

            if (target > cur) row++;

            else col--;

        }

        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] </aside>
* **Example 1:**
* **Input:** arr = [2,1]
* **Output:**
* false

 bool validMountainArray(vector<int>& arr) {

        int n = arr.size();

        int maxi = INT\_MIN;

        if(n< 3)

            return false;

        for(int i=0;i<n;i++)

        {

            maxi = max(maxi,arr[i]);

        }

        int pl = 0;

        for(int i=0;i<n;i++)

        {

            if(arr[i] == maxi)

            {

                pl = i;

                break;

            }

        }

        if(pl==0 || pl ==n-1)

        {

            return false;

        }

        for(int i=0;i<pl;i++)

        {

            if(arr[i] >= arr[i+1])

                return false;

        }

        for(int i=pl;i<n-1;i++)

        {

            if(arr[i] <= arr[i+1])

                return false;

        }

        return true;

    }

<aside> 💡 **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

</aside>

int findMaxLength(vector<int>& nums) {

        int count = 0;

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

            int zeros = 0, ones = 0;

            for (int j = i; j < nums.size(); j++) {

                if (nums[j] == 0) {

                    zeros++;

                } else {

                    ones++;

                }

                if (zeros == ones) {

                    count = max(count, j - i + 1);

                }

            }

        }

        return count;

    }

<aside> 💡 **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

**Explanation:**

We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 3*4 + 5*2 + 4*2 + 2*5 = 40.

</aside>

**int** minProductSum(vector**<int>&** A, vector**<int>&** B) {

sort(begin(A), end(A));

sort(begin(B), end(B), greater**<>**());

**int** ans **=** 0;

**for** (**int** i **=** 0; i **<** A.size(); **++**i) ans **+=** A[i] **\*** B[i];

**return** ans;

}