**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]

Sol: #include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<int> reconstructPermutation(const string& s) {

    int n = s.length();

    vector<int> perm;

    int low = 0, high = n;

    for (char c : s) {

        if (c == 'I') {

            perm.push\_back(low);

            low++;

        } else if (c == 'D') {

            perm.push\_back(high);

            high--;

        }

    }

    perm.push\_back(low);

    return perm;

}

int main() {

    string s = "IDID";

    vector<int> result = reconstructPermutation(s);

    cout << "Reconstructed Permutation: ";

    for (int num : result) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

Output: Reconstructed Permutation: 0 4 1 3 2

**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

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

**Output:** true

Sol: #include <iostream>

#include <vector>

using namespace std;

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

    int m = matrix.size();

    int n = matrix[0].size();

    int low = 0;

    int high = m \* n - 1;

    while (low <= high) {

        int mid = (low + high) / 2;

        int row = mid / n;

        int col = mid % n;

        int value = matrix[row][col];

        if (value == target) {

            return true;

        } else if (value < target) {

            low = mid + 1;

        } else {

            high = mid - 1;

        }

    }

    return false;

}

int main() {

    vector<vector<int>> matrix = {{1, 3, 5, 7},

                                            {10, 11, 16, 20},

                                            {23, 30, 34, 60}};

    int target = 3;

    bool result = searchMatrix(matrix, target);

    cout << boolalpha << result << endl;

    return 0;

}

Output: true

**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

Sol:

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

bool validMountainArray(vector<int>& arr) {

    int n = arr.size();

    if (n < 3) {

        return false;

    }

    int peak = INT\_MIN;

    int peakIndex = -1;

    for (int i = 1; i < n - 1; i++) {

        if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) {

            if (arr[i] > peak) {

                peak = arr[i];

                peakIndex = i;

            }

        }

    }

    if (peakIndex == -1 || peakIndex == 0 || peakIndex == n - 1) {

        return false;

    }

    for (int i = 1; i < peakIndex; i++) {

        if (arr[i] <= arr[i - 1]) {

            return false;

        }

    }

    for (int i = peakIndex + 1; i < n; i++) {

        if (arr[i] >= arr[i - 1]) {

            return false;

        }

    }

    return true;

}

int main() {

   vector<int> arr = {2, 1};

    cout << boolalpha << validMountainArray(arr) << endl;

    return 0;

}

Output:False

**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.

Sol: #include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

int findMaxLength(vector<int>& nums) {

    int n = nums.size();

   unordered\_map<int, int> countMap;

    int count = 0;

    int maxLength = 0;

    countMap[0] = -1;

    for (int i = 0; i < n; i++) {

        count += nums[i] == 0 ? -1 : 1;

        if (countMap.find(count) != countMap.end()) {

            int prevIndex = countMap[count];

            maxLength = std::max(maxLength, i - prevIndex);

        } else {

            countMap[count] = i;

        }

    }

    return maxLength;

}

int main() {

   vector<int> nums = {0, 1};

   cout << findMaxLength(nums) <<endl;

    return 0;

}

Output:2

**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.

Sol: #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int minProductSum(vector<int>& nums1, vector<int>& nums2) {

    int n = nums1.size();

    sort(nums1.begin(), nums1.end());

    sort(nums2.rbegin(), nums2.rend());

    int productSum = 0;

    for (int i = 0; i < n; i++) {

        productSum += nums1[i] \* nums2[i];

    }

    return productSum;

}

int main() {

    vector<int> nums1 = {5, 3, 4, 2};

    vector<int> nums2 = {4, 2, 2, 5};

   cout << minProductSum(nums1, nums2) << endl;

    return 0;

}

Output:40

**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].

Sol: #include <iostream>

#include <vector>

#include<map>

#include<algorithm>

#include <unordered\_map>

using namespace std;

 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;

    }

int main() {

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

    vector<int> original = findOriginalArray(changed);

    if (original.empty()) {

        cout << "Empty array" << endl;

    } else {

        cout << "Original array: ";

        for (int num : original) {

            cout << num << " ";

        }

        cout << endl;

    }

    return 0;

}

Output: Original array: 4 3 1

**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]]

Sol: #include <iostream>

#include <vector>

using namespace std;

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

    vector<vector<int>> matrix(n, vector<int>(n, 0));

    int num = 1;

    int rowStart = 0;

    int rowEnd = n - 1;

    int colStart = 0;

    int colEnd = n - 1;

    while (rowStart <= rowEnd && colStart <= colEnd) {

        for (int j = colStart; j <= colEnd; j++) {

            matrix[rowStart][j] = num++;

        }

        rowStart++;

        for (int i = rowStart; i <= rowEnd; i++) {

            matrix[i][colEnd] = num++;

        }

        colEnd--;

        if (rowStart <= rowEnd) {

            for (int j = colEnd; j >= colStart; j--) {

                matrix[rowEnd][j] = num++;

            }

            rowEnd--;

        }

        if (colStart <= colEnd) {

            for (int i = rowEnd; i >= rowStart; i--) {

                matrix[i][colStart] = num++;

            }

            colStart++;

        }

    }

    return matrix;

}

int main() {

    int n = 3;

    vector<vector<int>> matrix = generateMatrix(n);

    for (const auto& row : matrix) {

        for (int num : row) {

           cout << num << " ";

        }

        cout << endl;

    }

    return 0;

}

Sol: 1 2 3

8 9 4

7 6 5

**Question 8**

Given two [sparse matrices](https://en.wikipedia.org/wiki/Sparse_matrix) 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:**

**Input:** mat1 = [[1,0,0],[-1,0,3]], mat2 = [[7,0,0],[0,0,0],[0,0,1]]

**Output:**

[[7,0,0],[-7,0,3]]

Sol: #include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> multiply(vector<vector<int>>& mat1, vector<vector<int>>& mat2) {

    int m = mat1.size();

    int k = mat1[0].size();

    int n = mat2[0].size();

    vector<vector<int>> result(m, vector<int>(n, 0));

    for (int i = 0; i < m; i++) {

        for (int j = 0; j < k; j++) {

            if (mat1[i][j] != 0) {

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

                    if (mat2[j][p] != 0) {

                        result[i][p] += mat1[i][j] \* mat2[j][p];

                    }

                }

            }

        }

    }

    return result;

}

int main() {

    vector<vector<int>> mat1 = {{1, 0, 0}, {-1, 0, 3}};

    vector<vector<int>> mat2 = {{7, 0, 0}, {0, 0, 0}, {0, 0, 1}};

    vector<vector<int>> result = multiply(mat1, mat2);

    for (const auto& row : result) {

        for (int num : row) {

            cout << num << " ";

        }

        cout <<endl;

    }

    return 0;

}

Output: 7 0 0

-7 0 3