Q1.import java.util.ArrayList;

import java.util.List;

public class Solution {

public int[] findPermutation(String s) {

int n = s.length();

int current = 0;

List<Integer> perm = new ArrayList<>();

for (char ch : s.toCharArray()) {

if (ch == 'I') {

perm.add(current);

current++;

} else if (ch == 'D') {

perm.add(n);

n--;

}

}

perm.add(current);

int[] result = new int[perm.size()];

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

result[i] = perm.get(i);

}

return result;

}

}

Q2.public class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {

return false;

}

int rows = matrix.length;

int cols = matrix[0].length;

int left = 0;

int right = rows \* cols - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int midValue = matrix[mid / cols][mid % cols];

if (midValue == target) {

return true;

} else if (midValue < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return false;

}

}

Q3.public class Solution {

public boolean validMountainArray(int[] arr) {

int n = arr.length;

int i = 0;

// Check if the array has at least 3 elements

if (n < 3) {

return false;

}

// Traverse up the mountain until we reach its peak

while (i < n - 1 && arr[i] < arr[i + 1]) {

i++;

}

// Check if we reached the peak or the first element

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

return false;

}

// Traverse down the mountain from the peak

while (i < n - 1 && arr[i] > arr[i + 1]) {

i++;

}

// Check if we reached the end of the array

return i == n - 1;

}

}

Q4.import java.util.HashMap;

public class Solution {

public int findMaxLength(int[] nums) {

int maxLength = 0;

int count = 0;

HashMap<Integer, Integer> countMap = new HashMap<>();

countMap.put(0, -1); // Initialize the map with a count of 0 at index -1

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

if (nums[i] == 0) {

count--;

} else {

count++;

}

if (countMap.containsKey(count)) {

int prevIndex = countMap.get(count);

maxLength = Math.max(maxLength, i - prevIndex);

} else {

countMap.put(count, i);

}

}

return maxLength;

}

}

Q5.import java.util.Arrays;

public class Solution {

public int minProductSum(int[] nums1, int[] nums2) {

Arrays.sort(nums1);

Arrays.sort(nums2);

int n = nums1.length;

int minProductSum = 0;

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

minProductSum += nums1[i] \* nums2[n - i - 1];

}

return minProductSum;

}

}

Q6.import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class Solution {

public int[] findOriginalArray(int[] changed) {

if (changed.length % 2 != 0) {

return new int[]{};

}

Map<Integer, Integer> frequencyMap = new HashMap<>();

for (int num : changed) {

frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);

}

List<Integer> original = new ArrayList<>();

for (int num : changed) {

if (frequencyMap.getOrDefault(num, 0) == 0) {

continue;

}

int halfNum = num / 2;

if (frequencyMap.getOrDefault(halfNum, 0) <= 0) {

return new int[]{};

}

original.add(halfNum);

frequencyMap.put(halfNum, frequencyMap.get(halfNum) - 1);

frequencyMap.put(num, frequencyMap.get(num) - 1);

}

int[] result = new int[original.size()];

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

result[i] = original.get(i);

}

return result;

}

}

Q7.public class Solution {

public int[][] generateMatrix(int n) {

int[][] matrix = new int[n][n];

int rowStart = 0, rowEnd = n - 1;

int colStart = 0, colEnd = n - 1;

int num = 1;

while (num <= n \* n) {

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

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

}

rowStart++;

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

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

}

colEnd--;

if (rowStart <= rowEnd) {

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

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

}

rowEnd--;

}

if (colStart <= colEnd) {

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

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

}

colStart++;

}

}

return matrix;

}

}

Q8.public class Solution {

public int[][] multiply(int[][] mat1, int[][] mat2) {

int m = mat1.length;

int k = mat1[0].length;

int n = mat2[0].length;

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

// Transpose mat2 for more efficient column access

int[][] transposedMat2 = transpose(mat2);

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

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

int sum = 0;

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

if (mat1[i][p] != 0 && transposedMat2[j][p] != 0) {

sum += mat1[i][p] \* transposedMat2[j][p];

}

}

result[i][j] = sum;

}

}

return result;

}

// Transpose a matrix

private int[][] transpose(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

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

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

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

result[j][i] = matrix[i][j];

}

}

return result;

}

}