**Assignment Questions 6**

💡 **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]

package Assignment\_6;

import java.util.Arrays;

public class DIStringMatch {

    static int[] diStringMatch(String s) {

        int[] result = new int[s.length() + 1];

        int end = result.length - 1, start = 0;

        int index = 0;

        while(index < s.length()){

            if(s.charAt(index) == 'I'){

                result[index] = start;

                start++;

            }else{

                result[index] = end;

                end--;

            }

            index++;

        }

        result[result.length - 1] = start;

        return result;

    }

    public static void main(String[] args) {

        String s = "IDID";

        System.out.println(Arrays.toString(diStringMatch(s)));

    }

}

💡 **Question 2**

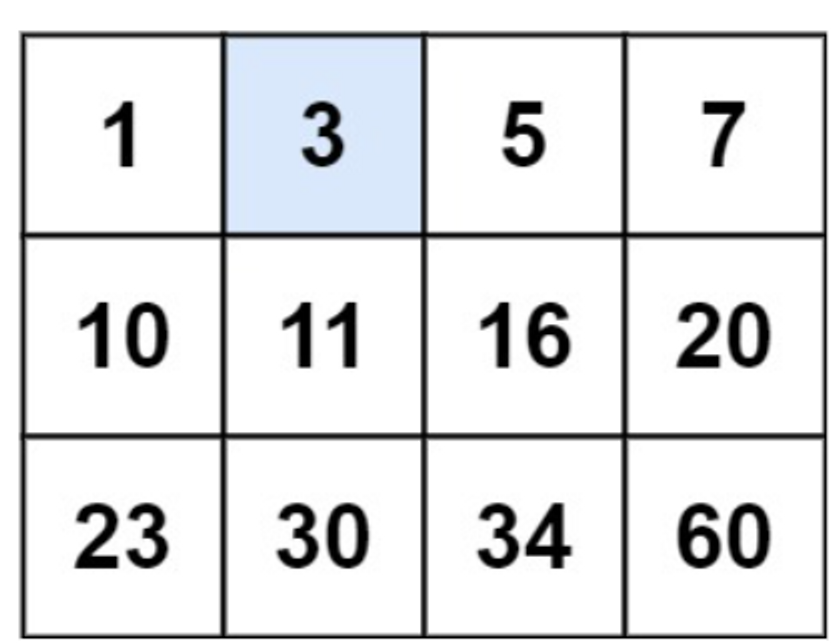
You are given an m x n integer matrix matrix with the following two properties:

* 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

package Assignment\_6;

public class searchA2DMatrix {

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

        int m=matrix.length;

        int n=matrix[0].length;

        int low=0,high=m\*n-1;

        while(low<=high) {

            int midIdx,midEle,rowIdx,colIdx;

            midIdx=low+(high-low)/2;

            rowIdx=midIdx/n;

            colIdx=midIdx%n;

            midEle=matrix[rowIdx][colIdx];

            if(midEle==target)

            return true;

            else if(midEle<target)

            low=midIdx+1;

            else

            high=midIdx-1;

        }

        return false;

    }

    public static void main(String[] args) {

        int [][] matrix = {{1,3,5,},{10,11,16,2},{23,30,34,60}};

        int target = 3;

        System.out.println(searchMatrix(matrix, target));

    }

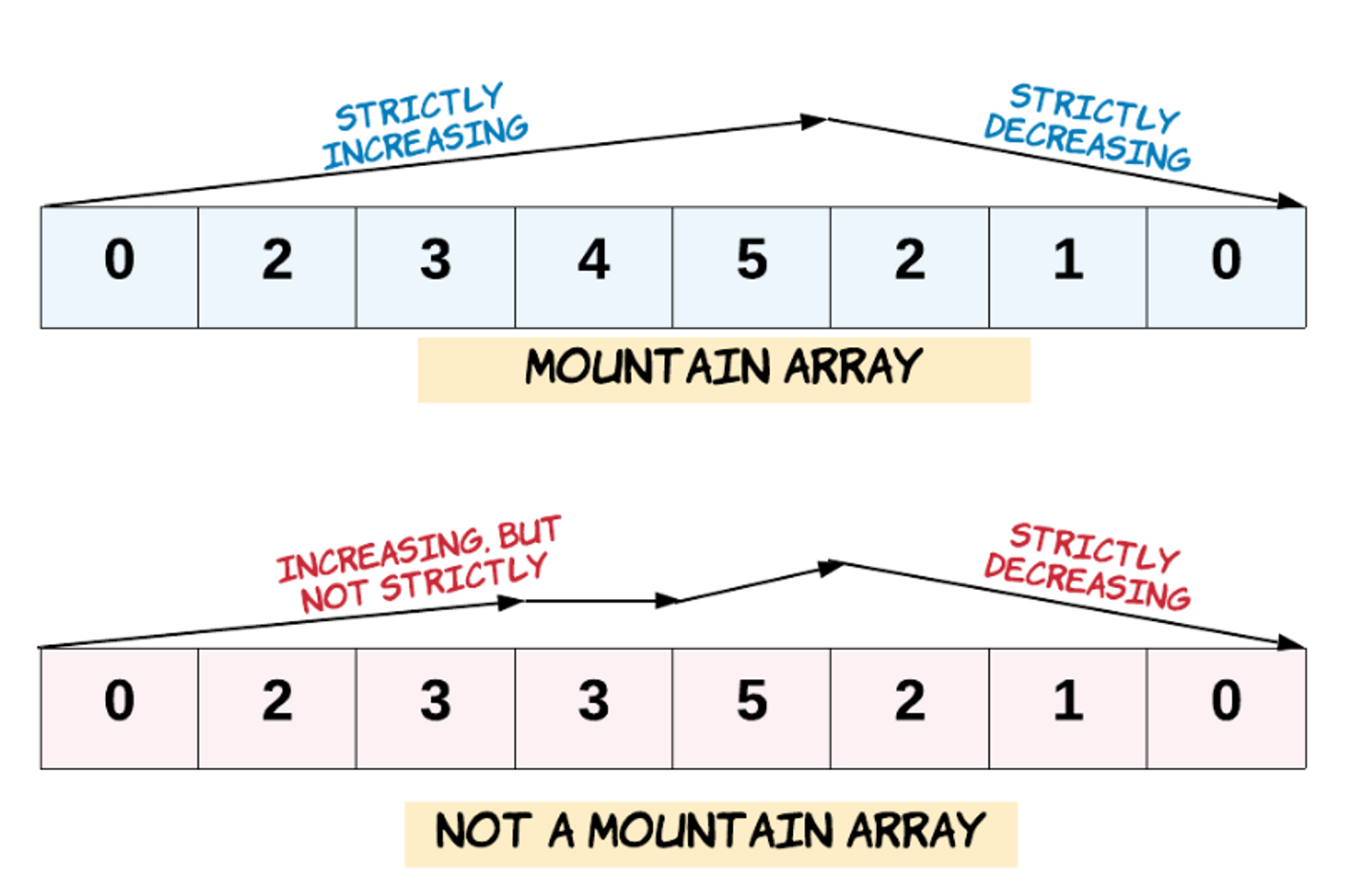
}

💡 **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

package Assignment\_6;

public class validMountainArray {

    static boolean validMountainArray(int[] arr) {

        //if size is < 2 then it not mountain

        if(arr.length<3) return false;

        int topidx=0;

        int top=0;

        //find max value and that index

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

            if(arr[i]>top){

                top = arr[i];

                topidx=i;

            }

        }

        //check that one side mountain or not .

        if(top==arr[arr.length-1] || top==arr[0]) return false;

        //check perfact mountain or not

        int i=0;

        while(i<topidx)

        {

            if(arr[i] >= arr[i+1]) return false;

            i++;

        }

        while(topidx<arr.length-1){

            if(arr[topidx] <= arr[topidx+1]) return false;

            topidx++;

        }

        return true;

    }

    public static void main(String[] args) {

        int []  arr = {0,3,2,1};

        System.out.println(validMountainArray(arr));

    }

}

💡 **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.

package Assignment\_6;

public class contiguousArray {

    static int findMaxLength(int[] nums) {

        int count = 0;

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

            int zeros = 0, ones = 0;

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

                if (nums[j] == 0) {

                    zeros++;

                } else {

                    ones++;

                }

                if (zeros == ones) {

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

                }

            }

        }

        return count;

    }

    public static void main(String[] args) {

        int [] nums = {0,1};

        System.out.println(findMaxLength(nums));

    }

}

💡 **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.

package Assignment\_6;

import java.util.\*;

public class minimizeProductSumTwoArrays {

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

        int ans = 0;

        Arrays.sort(nums2);

        Arrays.sort(nums1);

        int i = 0;

        int j = nums2.length-1;

        while(i < nums1.length && j >= 0)

        {

            ans += nums1[i] \* nums2[j];

            i++;

            j--;

        }

        return ans;

    }

    public static void main(String[] args) {

        int [] nums1 = {5,3,4,2}, nums2 = {4,2,2,5};

        System.out.println(minProductSum(nums1, nums2));

    }

}

💡 **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].

package Assignment\_6;

import java.util.\*;

public class findOriginalArrayFromDoubledArray {

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

        int[] temp = new int[changed.length/2];

        Queue<Integer> x = new LinkedList<>();

        Arrays.sort(changed);

        int i = 0;

        for(int num : changed){

            if(!x.isEmpty() && x.peek()==num)

                temp[i++] = x.poll()/2;

            else x.add(num\*2);

        }

        return x.size()>0 ? new int[]{}:temp;

    }

    public static void main(String[] args) {

        int [] changed = {1,3,4,2,6,8};

        System.out.println(Arrays.toString(findOriginalArray(changed)));

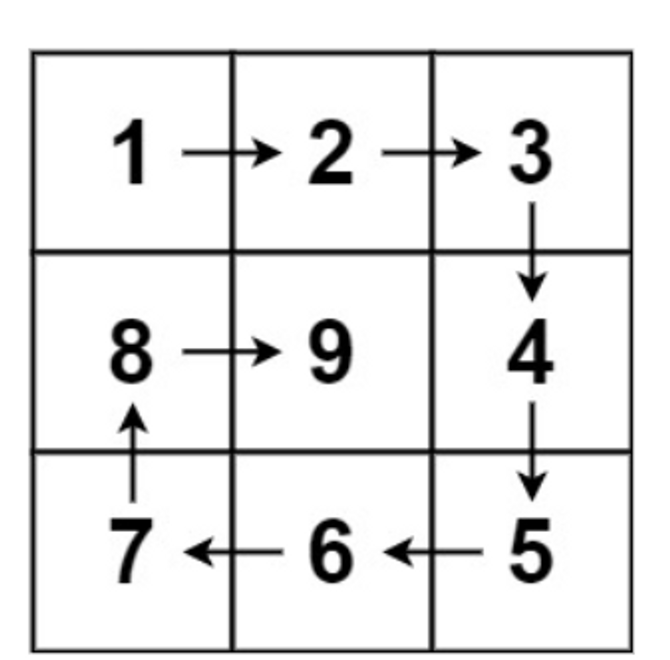
    }

}

💡 **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]]

package Assignment\_6;

public class spiralMatrixII {

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

        int num[][] = new int[n][n];

        int count = 1;

        int top = 0,left = 0,bottom = n -1,right = n - 1;

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

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

                num[top][i] = count++;

            }

            top++;

            for(int i = top ; i <= bottom; i++){

                num[i][right] = count++;

            }

            right--;

            if(top <= bottom){

                for(int i = right; i >= left; i --){

                    num[bottom][i] = count++;

                }

                bottom--;

            }

            if(left <= right){

                for(int i = bottom; i >= top; i --){

                    num[i][left] = count++;

                }

                left++;

            }

        }

        return num;

    }

    public static void main(String[] args) {

        int n = 3;

        int [] [] arr = generateMatrix(n);

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

            for (int j = 0; j < arr[i].length; j++)

                System.out.print(arr[i][j] + " ");

            System.out.print("\n");

        }

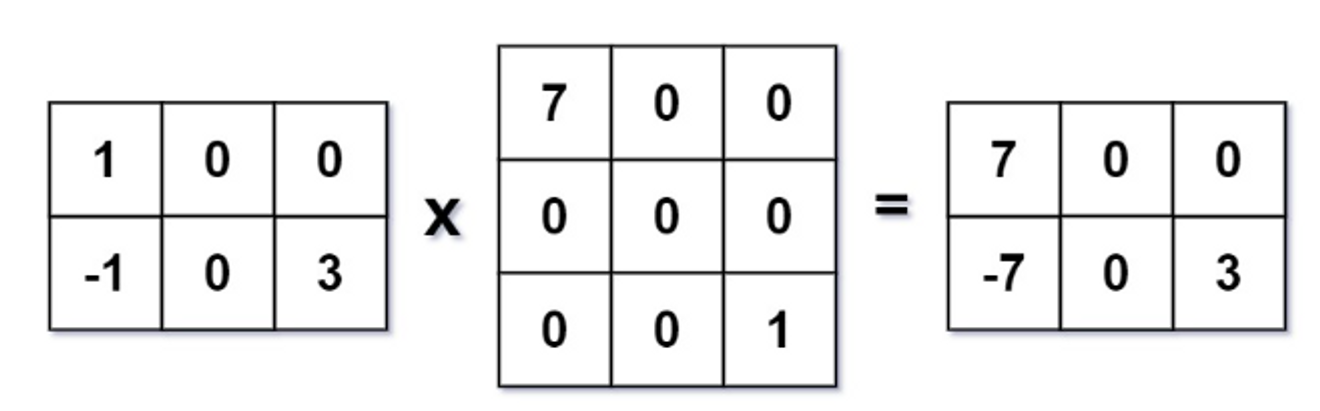
    }

}

💡 **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]]

package Assignment\_6;

import java.util.\*;

public class sparseMatrixMultiplication {

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

        int r1 = mat1.length, c1 = mat1[0].length, c2 = mat2[0].length;

        int[][] res = new int[r1][c2];

        Map<Integer, List<Integer>> mp = new HashMap<>();

        for (int i = 0; i < r1; ++i) {

            for (int j = 0; j < c1; ++j) {

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

                    mp.computeIfAbsent(i, k -> new ArrayList<>()).add(j);

                }

            }

        }

        for (int i = 0; i < r1; ++i) {

            for (int j = 0; j < c2; ++j) {

                if (mp.containsKey(i)) {

                    for (int k : mp.get(i)) {

                        res[i][j] += mat1[i][k] \* mat2[k][j];

                    }

                }

            }

        }

        return res;

    }

    public static void main(String[] args) {

        int [][]  mat1 = {{1,0,0},{-1,0,3}}, mat2 = {{7,0,0},{0,0,0},{0,0,1}};

        int [][] arr = multiply(mat1, mat2);

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

            System.out.println(Arrays.toString(arr[i]));

        }

    }

}