<aside>

💡 **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]

</aside>

```
public class Solution {
    public int[] diStringMatch(String s) {
        int[] arr = new int[s.length() + 1];
        int max = s.length();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == 'D') {
                arr[i] = max;
                max--;
            }
        }
        for (int i = s.length() - 1; i >= 0 && max > 0; i--) {
            if (s.charAt(i) == 'I' && arr[i + 1] == 0) {
                arr[i + 1] = max;
                max--;
            }
        }
        for (int i = 0; i < arr.length && max > 0; i++) {
            if (arr[i] == 0) {
```

```
            arr[i] = max;
            max--;
          }
        }

        return arr;
    }
}
```

<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:

</aside>

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int i = matrix.length - 1;
        int j = 0;
        while (i >= 0 && j < matrix[0].length) {
            if (matrix[i][j] > target) {
                i --;
            }
            else if (matrix[i][j] < target) {
                j ++;
            }
            else {
                return true;
            }
        }
        return false;
    }
}
```

<aside>

💡 **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>

```
class Solution {
    public boolean validMountainArray(int[] arr) {
        int i = 0;
        int j = arr.length - 1;
        int n = arr.length - 1;
        while (i + 1 < n && arr[i] < arr[i+1]) {
            i++;
        }

        while (j > 0 && arr[j] < arr[j-1]) {
            j--;
        }

        return (i > 0 && i == j && j < n);
    }
}
```

<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>

```
public class Solution {
    public int findMaxLength(int[] nums) {
        int[] arr = new int[2 * nums.length + 1];
        Arrays.fill(arr, -2);
        arr[nums.length] = -1;
        int maxlen = 0, count = 0;
        for (int i = 0; i < nums.length; i++) {
            count = count + (nums[i] == 0 ? -1 : 1);
            if (arr[count + nums.length] >= -1) {
                maxlen = Math.max(maxlen, i - arr[count + nums.length]);
            } else {
                arr[count + nums.length] = i;
            }

        }
        return maxlen;
    }
}
```

<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>

<aside>

💡 **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].

</aside>

```
class Solution {
  public int[] findOriginalArray(int[] changed) {
    List<Integer> ans = new ArrayList<>();
    Queue<Integer> q = new ArrayDeque<>();
    Arrays.sort(changed);
    for (final int num : changed)
      if (!q.isEmpty() && num == q.peek()) {
        q.poll();
      } else {
        q.offer(num * 2);
        ans.add(num);
      }
    return q.isEmpty() ? ans.stream().mapToInt(Integer::intValue).toArray() : new int[] {};
  }
}
```

<aside>

💡 **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:

</aside>

```
public class Solution {
    public int[][] generateMatrix(int n) {
        // Start typing your Java solution below
        // DO NOT write main() function
        if(n<=0) return new int[0][];
        int[][] result=new int[n][n];
        int xBeg=0,xEnd=n-1;
        int yBeg=0,yEnd=n-1;
        int cur=1;
        while(true){
            for(int i=yBeg;i<=yEnd;i++) result[xBeg][i]=cur++;
            if(++xBeg>xEnd) break;
            for(int i=xBeg;i<=xEnd;i++) result[i][yEnd]=cur++;
            if(--yEnd<yBeg) break;
            for(int i=yEnd;i>=yBeg;i--) result[xEnd][i]=cur++;
            if(--xEnd<xBeg) break;
            for(int i=xEnd;i>=xBeg;i--) result[i][yBeg]=cur++;
            if(++yBeg>yEnd) break;
        }
        return result;
    }
}
```

<aside>

💡 **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.

</aside>

```
class Solution {
    public 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;
    }
}
```