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>

<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.

</aside>

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

Output: true

```
class Solution {
  public boolean searchMatrix(int[][] matrix, int target) {
    int m= matrix.length;
    int n= matrix[0].length;
    int i=0;
    int j= m*n;
    while(i<m &){
      int mid= i+(j-i)/2;
      if(matrix[i/2][mid]== target){ return true;}
      else if(matrix[i/2][mid]<target){
        i= mid+1;
      }else{
        j= mid-1;
```

```
      }
    }

    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 n = arr.length;
  if (n < 3) {
    return false;
  }

  int i = 0;
  while (i < n - 1 && arr[i] < arr[i + 1]) {
    i++;
  }

  if (i == 0 || i == n - 1) {
    return false;
  }

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

  return i == n - 1;
}

}
```

<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>

```
class Solution {

    public int findMaxLength(int[] nums) {
  HashMap<Integer, Integer> map = new HashMap<>();
  int count = 0;
  int maxLength = 0;

  for (int i = 0; i < nums.length; i++) {
    // Increment count by 1 for '0' and decrement count by 1
for '1'
    count += (nums[i] == 0 ? 1 : -1);


    if (count == 0) {
      maxLength = i + 1;
    }


    if (map.containsKey(count)) {
      maxLength = Math.max(maxLength, i - map.get(count));
    } else {
      map.put(count, i);
    }
  }

  return maxLength;
}

}
```

<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>

```
import java.util.Arrays;

public class Solution {
    public int minProductSum(int[] nums1, int[] nums2) {
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        int minProductSum = 0;
        int left = 0;
        int right = nums2.length - 1;

        while (left <= right) {
            minProductSum += nums1[left] * nums2[right];
            left++;
            right--;
        }

        return minProductSum;
    }
}
```

<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 List findDuplicates(int[] nums) {
        HashSet set= new HashSet<>();
        List list= new ArrayList<>();
        for(int n: nums){
                if(!set.add(n)) list.add(n);
               set.add( n);
        }
        return list;
         }
        }
```

<aside> 💡 **Question 7**

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

</aside>

Input: n = 3

Output: [[1,2,3],[8,9,4],[7,6,5]]

```
class Solution {
  public int[][] generateMatrix(int n) {
    int[][] matrix = new int[n][n];
    int left = 0, right = n - 1, top = 0, bottom = n- 1;
    int i = 1;
    while (left <= right && top <= bottom) {
```

```
        for (int j = left; j <= right && top <= bottom; j++) {
          matrix[top][j] = i++;
        }
        top++;
        for (int j = top; j <= bottom && left <= right; j++) {
          matrix[j][right] = i++;
        }
        right--;
        for (int j = right; j >= left && top <= bottom; j--) {
          matrix[bottom][j] = i++;
        }
        bottom--;
        for (int j = bottom; j >= top && left <= right; j--) {
          matrix[j][left] = i++;
        }
        left++;
      }
      return matrix;

    }
}
```