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]

Pgrm:

```
class DI_String{
    public static 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;
    }
}
```

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

Prgm:

```
class Matrix{
    public static boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length, n = matrix[0].length;
        int left = 0, right = m * n - 1;
        while (left < right) {
            int mid = (left + right) >> 1;
            int x = mid / n, y = mid % n;
            if (matrix[x][y] >= target) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return matrix[left / n][left % n] == target;
    }
    public static void main(String[] args) {
        int matrix[][] = {{1,3,5,7},{10,11,16,20},{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

Prgm:

```
class mountArray{
    public static boolean validMountArray(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);
    }
    public static void main(String[] args) {
        int arr[] = {2,1};
        System.out.println(validMountArray(arr) ? true : 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

Prgm:

```
class contiguousArray{

    public static int findMaxLength(int[] nums) {

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

        mp.put(0, -1);

        int s = 0, ans = 0;

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

            s += nums[i] == 1 ? 1 : -1;

            if (mp.containsKey(s)) {

                ans = Math.max(ans, i - mp.get(s));

            } else {

                mp.put(s, i);

            }

        }

        return ans;

    }

    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

Prgm:

```
class ProductSum{
```

```
    public 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};
        int[] 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]

Prgm:

```
class originalArray{
   public static List<Integer>findOriginal(int[] arr){
        Map<Integer, Integer> numFreq = new HashMap<>();
         for (int i = 0; i < arr.length; i++) {
           numFreq.put(
               arr[i],
               numFreq.getOrDefault(arr[i], 0)
                   + 1);
       }
        Arrays.sort(arr);

       List<Integer> res = new ArrayList<>();
       for (int i = 0; i < arr.length; i++) {

           int freq = numFreq.get(arr[i]);
           if (freq > 0) {

               res.add(arr[i]);

               numFreq.put(arr[i], freq - 1);

               int twice = 2 * arr[i];

               numFreq.put(
                   twice,
                   numFreq.get(twice) - 1);
           }
       }
       return res;
   }
```

```
    public static void main(String[] args){
        List<Integer> res = findOriginal(
            new int[] {1,3,4,2,6,8 });
        for (int i = 0; i < res.size(); i++) {
            System.out.println(
                res.get(i) + " ");
        }
    }
}
```

Question 8

Given two sparse matrices 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]]

Prgm:

```
class SparseMatrix{
    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;
  }
}
```