```
import java.util.*;

public class quora {

   // 1 很简单，一个数字，求所有位数的乘积减去所有位数的和。
```

product_sum

```
   public static int product_sum (int number) {
      int prod = 1;
      int sum = 0;
      while(number != 0) {
         int a = number%10;
         number /= 10;
         prod *= a;
         sum += a;
      }
      return prod - sum;
   }

   // 输入一组words和一组valid letters，判断有多少个words是valid。
   // 判断条件是words里的所有upper and lower letter必须在valid letters里面。
   // 如果word里面有special character不用管。注意valid letter只有小写，
   // 但是words里面有大写的也算valid。
   // 比如words = [hEllo##, This^^],
   // valid letter = [h, e, l, 0, t, h, s];
   // "hello##" 就是valid，因为h，e，l，o都在valid letter 里面，
   // “This^^” 不valid, 因为i不在valid letter里面
   //
```

wordsIsValid

```
   public static int wordsIsValid(String[] words, char[] letters) {
      Set<Character> set = new HashSet<>();
      for(char c : letters) {
         set.add(Character.toLowerCase(c));
      }
      int res = 0;
      for(String word : words) {
         char[] wordChar = word.toCharArray();
         boolean isValid = true;
         for(int i = 0; i < wordChar.length; i++) {
            char curr = wordChar[i];
            if(Character.isLowerCase(curr) || Character.isUpperCase(curr)) {
               curr = Character.toLowerCase(curr);
```

```
                if(!set.contains(curr)) {
                    isValid = false;
                    break;
                }
            }
        }
        if(isValid) {
            res++;
        }
    }
    return res;
}
```

broken keyboard

输入一组words和一组valid letters，判断有多少个words是valid。判断条件是words里的所有upper and lower letter必须在valid letters里面。如果word里面有special character不用管。注意valid letter只有小写，但是words里面有大写的也算valid。比如words = [hEllo##, This^^], valid letter = [h, e, l, 0, t, h, s]; "hello##" 就是valid，因为h，e，l，o都在valid letter 里面，"This^^" 不valid, 因为i不在valid letter里面

input: a = "Hello, my dear friend!", b = ['h', 'e', 'l', 'o', 'm']

output: 1 题目是键盘坏了，只剩下b中的字母按键和所有的数字和符号案件能用，同时shift键是好的， 所以可以切换大小写。问a中的单词有几个可以用当前坏掉的键盘打出来。

```
public static int brokenKeyboard(String a, char[] b) {
    Set<Character> set = new HashSet<>();
    for(char c : b) {
        set.add(c);
    }
    int res = 0;
    String[] temp = a.split(" ");
    for(String s : temp) {
        char[] sChar = s.toCharArray();
        boolean isIn = true;
        for(char ch : sChar) {
            ch = Character.toLowerCase(ch);
            if(!set.contains(ch)) {
                isIn = false;
                break;
            }
        }
        if(isIn) {
            res++;
        }
    }
    return res;
}
```

Compare the String with Frequency

compare两个string，只有小写字母。 每个stirng内部可以任意换位置，所以位置不重要。每个string内部两个letter出现的频率也可以互换，所以这题只需要两个string每个frequency出现的次数要一样。比如“babzccc” 和 “bbazzcz” 就返回“true”， 因为z和c可以互换频率。 但是 “babzcccm” 和 “bbazzczl” 就不一样，因为m在第一个里出现过，第二个里没有出现过。 **If two strings are close enough.**

Given two rules to define two strings are close enough.

1. you can swap neighbor char any times. Ex. "abb" -> "bba"

2. If two strings have the same character, then you can change the character into another.

Ex. If both strings contain "a" and "b", you can change all "a"s in the first string or change all "b"s in the first string. same as the second string

Ex.

Input: S1 = "babzccc", S2 = "abbzczz" Output: True

Sol: HashMap<Character, Integer> counts

Check if keySet() eqauls()

HashMap<Integer, Integer> counts of counts, check if the same

```
//

public static boolean compareString(String s1, String s2) {
    Map<Character, Integer> map1 = new HashMap<Character, Integer>();
    for(int i=0; i<s1.length(); i++) {
        map1.put(s1.charAt(i), map1.getOrDefault(s1.charAt(i), 0) + 1);
    }
```

```
        Map<Character, Integer> map2 = new HashMap<Character, Integer>();
        for(int i=0; i<s2.length(); i++) {
            map2.put(s2.charAt(i), map2.getOrDefault(s2.charAt(i), 0) + 1);
        }
        for(char ch : map1.keySet()) {
            if(!map2.containsKey(ch)) {
                return false;
            }
        }
        for(char ch : map2.keySet()) {
            if(!map1.containsKey(ch)) {
                return false;
            }
        }
        Map<Integer, Integer> countS1 = new HashMap<Integer, Integer>();
        for(char ch : map1.keySet()) {
            int freq = map1.get(ch);
            countS1.put(freq, countS1.getOrDefault(freq, 0) + 1);
        }

        Map<Integer, Integer> countS2 = new HashMap<Integer, Integer>();
        for(char ch : map2.keySet()) {
            int freq = map2.get(ch);
            countS2.put(freq, countS2.getOrDefault(freq, 0) + 1);
        }

        if(s1.length() != s2.length()) {
            return false;
        }
        for(int freq : countS1.keySet()) {
            if(countS1.get(freq) != countS2.get(freq)) {
                return false;
            }
        }
        return true;
    }
```

coolFeature

输入a，b两个array， 一个query array。query有两种type， 一种是[target]查从a中取一个数，b中取一个数，求加起来等于target的情况有多少种。第二种query是[index, num], 把b中在index位置的数字改成num，这种query不需要输出。最后输出所有第一种query的result。

coolFeature
Give three array a, b and query. This one is hard to explain. Just read the example. Input:
a = [1, 2, 3]
b = [3, 4]
query = [[1, 5], [1, 1 , 1], [1, 5]] Output:
[2, 1]
Explain:
Just ignore every first element in sub array in query.
So we will get a new query like this query = [[5], [1, 1], [5]]
Only record the result when meet the single number in new query array.
And the rule of record is find the sum of the single number.
The example above is 5 = 1 + 4 and 5 = 2 + 3, there are two result.
So currently the output is [2]
When we meet the array length is larger than 1, such as [1, 1]. That means we will replace the b[x] = y, x is the first element, y is second element. So in this example, the b will be modify like this b = [1, 4]
And finally, we meet the [5] again. So we will find sum again. This time the result is 5 = 1 + 4.
So currently the output is [2, 1]
note: Don't have to modify the query array, just ignore the first element.

```
    //
    public static int[] coolFeature(int[] a, int[] b, int[][] querys) {

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

        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < a.length; i++) {
            map.put(a[i], map.getOrDefault(a[i], 0) + 1);
        }

        for(int[] query : querys) {
            if(query.length == 2) {
                int temp = findSum(map, b, query[1]);
                ans.add(temp);
            } else if (query.length == 3) {
                changeArray(a, b, query[1], query[2]);
            }
        }
        int[] ansArray = new int[ans.size()];
        for(int i = 0; i < ans.size(); i++) {
            ansArray[i] = ans.get(i);
        }
        return ansArray;
    }

    public static int findSum(Map<Integer, Integer> map, int[] b, int target) {
        int res = 0;
        for (int i = 0; i < b.length; i++) {
            if(map.containsKey(target - b[i])) {
                res += map.get(target - b[i]);
            }
        }
        return res;
    }

    public static void changeArray(int[] a, int[] b, int loc, int num) {
        b[loc] = num;
    }
```

findEvenDigit

5. Find how many numbers have even digit in a list. Ex.Input: A = [12, 3, 5, 3456]
Output: 2

```
    // test 6
    public static int findEvenDigit(int[] a) {
        int res = 0;
        for(int num : a) {
            String s = Integer.toString(num);
            if(s.length() % 2 == 0) {
                res++;
            }
        }
        return res;
    }
```

findMostCommon

Find the most common elements in a list.

Ex.
 Input: A = [2, 2, 3, 3, 5]
Output: [2, 3]

```
    // test 7
```

```
public static List<Integer> findMostCommon(int[] A) {
    Map<Integer, Integer> map = new HashMap<Integer, Integer>();
    int maxVal = 0;
    for(int a : A) {
        map.put(a, map.getOrDefault(a, 0) + 1);
        maxVal = Math.max(maxVal, map.get(a));
    }

    List<Integer> res = new ArrayList<>();
    for(int num : map.keySet()) {
        if(map.get(num) == maxVal) {
            res.add(num);
        }
    }

    return res;
}
```

maxRibbon

Given a list representing the length of ribbon, and the target number "k" parts of ribbon. we want to cut ribbon into k parts with the same size, at the same time we want the maximum size.
Ex.
Input: A = [1, 2, 3, 4, 9], k = 5
Output: 3
Explanation:
if size = 1, then we have 19 parts
if size = 2, then we have 8 parts
if size = 3, then we have 5 parts
if size = 4, then we have 3 parts, which is not enough. So return the max size = 3.

```
//test 8
public static int maxRibbon(int[] A, int k) {
    int hi = 0;
    for(int i = 0; i < A.length; i++) {
        hi += A[i];
    }
    int lo = 0;
    int res = 0;
    while(lo <= hi) {
        int mid = (lo + hi) / 2;
        int part = 0;
        for(int i = 0; i < A.length; i++) {
            part += A[i]/mid;
        }
        if(part >= k) {
            res = Math.max(res, mid);
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }
    return res;
}
```

GoodTuples

```
// test 9
//GoodTuples
// Give an array and find the count of a pair number and a
// single number combination in a row of this array. Target array is
// a[i - 1], a, a[i + 1]
// Example:
        Input: a = [1, 1, 2, 1, 5, 3, 2, 3]
        Output: 3
        Explain:
        [1, 1, 2] -> two 1 and one 2(O)
        [1, 2, 1] -> two 1 and one 2(O)
        [2, 1, 5] -> one 2, one 1 and one five(X) [1, 5, 3] -> (X)
        [5, 3, 2] -> (X)
        [3, 2, 3] -> (O)
        different characters
        intput: a = "aabdcreff"
        output: 5
        问a中存在多少a, a[i-1], a[i+1]都不同的情况


public static int goodTuples(int[] a) {
    int res = 0;
    for(int i = 1; i < a.length - 1; i++) {
        res += check(a[i-1], a[i], a[i+1]);
    }
    return res;
}
```

```
public static int check(int a, int b, int c) {
    if(a == b && a != c) {
        return 1;
    } else if (a == c && a != b) {
        return 1;
    } else if (b == c && a != b) {
        return 1;
    } else {
        return 0;
    }
}
```

rotateDiagonal

rotate with k
对角线方向旋转矩阵中的元素k次，其中1<= k <= 4
Example: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] -->
[[1, 4, 3], [8, 5, 2], [7, 6, 9]]
[[1,2,3,4,5],

[6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20], [21,22,23,24,25]] --> [[1,16,11,6,5], [22,7,12,9,2], [23,18,13,8,3], [24,17,14,19,4], [21,10,15,20,25]]
48 rotate image 变形 check是不是diagnol

```
public static void rotateDiagonal(int[][] matrix, int k) {

    int n = matrix.length;

    for(int s = 0; s < k; s++) {
        // rotate
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < i; j++) {
                if(i != j && i + j != n - 1) {
                    int temp = matrix[i][j];
                    matrix[i][j] = matrix[j][i];
                    matrix[j][i] = temp;
                }
            }
        }

        // fanzhuan
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n/2; j++) {
                if(i != j && i + j != n - 1) {
                    int temp = matrix[i][j];
                    matrix[i][j] = matrix[i][n - 1 -j];
                    matrix[i][n - 1 -j] = temp;
                }
            }
        }
    }
}
```

```
// test 11
```

isPrefix，给两个str array a，b，判断b里面的所有str是不是都是a的str各种组合黏在一起而成的
Sol: 直接建立所有a里面str的permutation然后存入set里

```
public static boolean isPrefix(String[] a, String[] b) {
    // 这题想干啥哦
    return true;
}
```

divideSubString

divisorSubstrings
Give a number n and digit number k find all serial substring is able to divisible n.
Input: n = 120, k = 2 Output: 2
Explain:
120 -> 12 and 20 120 % 12 == 0 (O)

120 % 20 == 0 (O) Input: n = 555, k = 1; Output: 1
Explain:
555 -> 5, 5 and 5 (Duplicate so only count one 5) 555 % 5 == 0 (O)
Input: n = 2345, k = 2
Output: 0
Explain:
2345 -> 23, 34, 45 2345 % 23 != 0 (X) 2345 % 34 != 0 (X) 2345 % 45 != 0 (X)

```
// test 12
// divide sub strings
public static int divideSubString(String s, int k) {
        int res = 0;
        int total = Integer.parseInt(s);
        Set<Integer> set = new HashSet<Integer>();
        for (int i = 0; i < s.length() - k + 1; i++) {
            String temp = s.substring(i, i + k);
            int num = Integer.parseInt(temp);
            System.out.println(num);
            System.out.println(total);
            if(!set.contains(num) && num != 0) {
                if(total % num == 0) {
                    res++;
                }
            }
            set.add(num);
        }
        return res;
    }
```

sumOfString

给两串字符串，每个char就是一个digit，然后从后往前加起来，把结果放到一 个字符串输出，挺简单的。e.g. '99' + '99' = '1818'
如果写Java的话最好用StringBuilder, String 会 TLE

```
    // test 13
    public static String sumOfString(String s1, String s2) {
        if(s1 == null || s1.length() == 0) return s2;
        if(s2 == null || s2.length() == 0) return s1;
        int len1 = s1.length();
        int len2 = s2.length();
        StringBuilder sb = new StringBuilder();
        int idx1 = len1 - 1;
        int idx2 = len2 - 1;
        while(idx1 >= 0 && idx2 >= 0) {
            char c1 = s1.charAt(idx1--);
            char c2 = s2.charAt(idx2--);
            int num1 = c1 - '0';
            int num2 = c2 - '0';
            int sum = num1 + num2;
            sb.insert(0, Integer.toString(sum));
        }

        while(idx1 >= 0) {
            sb.insert(0, s1.charAt(idx1--));
        }

        while(idx2 >= 0) {
            sb.insert(0, s2.charAt(idx2--));
        }

        return sb.toString();
    }

  //test 14
```

maxArithmeticLength

Suppose we have array a and b (no duplicates & sorted) a = [0,4,8,20]
b = [5,7,12,16,22]
Suppose u can pick any number of element from b (could be 0), and u want to insert them into array a such that all elements in a are increasing by certain number,
so in this example u can pick "12, 16" from b and append into a such that a = [0,4,8,12,16,20], which increase by 4 for each element
write a function to return the maximum number of element in a after appending elements from b (in the exmaple above the result is 6), if there is no such case, return -1

```
 public static void addAfter(int[] b, int idxB, int diff, LinkedList<Integer> temp) {
        while(idxB < b.length) {
            if(b[idxB] == diff + temp.get(temp.size() - 1)) {
                temp.add(b[idxB]);
            }
            idxB++;
        }
    }

    public static void addFront(int[] b, int idxB, int diff, LinkedList<Integer> temp) {
        while(idxB >= 0) {
            if(b[idxB] == temp.get(0) - diff) {
                temp.addFirst(b[idxB]);
            }
            idxB--;
        }
    }

    public static int checkIdxA(int[] a, int idxA, int diff, LinkedList<Integer> temp) {
        while(idxA < a.length) {
            if(a[idxA] == diff + temp.get(temp.size() - 1)) {
                temp.add(a[idxA++]);
            } else {
                break;
            }
        }
        return idxA;
    }
    public static int findLong(int[] b, int val, int pos, int loc) {
        LinkedList<Integer> temp = new LinkedList<Integer>();
        temp.add(val);
        int diff = Math.abs(val - b[loc]);
        int res = 0;
        if(pos == -1) {
            addAfter(b, 0, diff, temp);
        } else if (pos == b.length - 1) {
            addFront(b, b.length - 1, diff, temp);
        } else {
            addAfter(b, pos, diff, temp);
            addFront(b, pos, diff, temp);
        }
        res = Math.max(res, temp.size());
        return res;
    }
    public static int maxArithmeticLength(int[] a, int[] b) {
        int lenA = a.length;
        int lenB = b.length;

        // find the place a[0] in b
        int left = 0, right = lenB - 1;
        int pos = -1;
        while (left <= right) {
            int mid = (right + left) / 2;
            if (b[mid] >= a[0]) {
                right = mid - 1;
            } else {
                pos = mid;
                left = mid + 1;
            }
        }

        // pos is the first b[pos] strictly less than a[0]
        int res = -1;
        if(a.length == 1) {
            // only have a[0] but not sure about the difference
            // the problem is equivalent to find the max Arithmetic length
            // contains A[0]
            for(int i = 0; i < b.length; i++) {
```

```
            res = Math.max(res, findLong(b, a[0], pos, i));
        }
    } else {
        // get the range of the difference
        int diffMax = a[1] - a[0];
        for(int i = 1; i < lenA; i++) {
            diffMax = Math.min(diffMax, a[i] - a[i-1]);
        }
        for(int diff = 0; diff <= diffMax; diff++) {

            LinkedList<Integer> temp = new LinkedList<Integer>();
            temp.add(a[0]);

            if(pos == -1) {
                // all elements in b is greater than A[0]
                int idxA = 1, idxB = 0;
                while(idxA < lenA && idxB < lenB) {
                    if(a[idxA] == diff + temp.get(temp.size() - 1)) {
                        temp.add(a[idxA++]);
                    } else if (b[idxB] == diff + temp.get(temp.size() - 1)) {
                        temp.add(b[idxB++]);
                    } else {
                        idxB++;
                    }
                }
                idxA = checkIdxA(a, idxA, diff, temp);
                if(idxA == lenA) {
                    addAfter(b, idxB, diff, temp);
                }
            } else if (pos == lenB - 1) {
                // all elements in B is smaller than a[0]
                int idxA = 1;
                idxA = checkIdxA(a, idxA, diff, temp);
                if(idxA == lenA) {
                    addFront(b, b.length - 1, diff, temp);
                }
            } else {
                // a[0] split [0, pos] and [pos + 1, lenB - 1]
                int idxA = 1, idxB = pos + 1;
                while (idxA < lenA && idxB < lenB) {
                    if(a[idxA] == diff + temp.get(temp.size() - 1)) {
                        temp.add(a[idxA++]);
                    } else if (b[idxB] == diff + temp.get(temp.size() - 1)) {
                        temp.add(b[idxB++]);
                    } else {
                        idxB++;
                    }
                }
                idxA = checkIdxA(a, idxA, diff, temp);
                // add range [0, pos]
                if(idxA == lenA) {
                    addFront(b, pos, diff, temp);
                    addAfter(b, idxB, diff, temp);
                }
            }
            res = Math.max(res, temp.size());
        }
    }
    return res;
}
```

Query

给一串数字a，再给一串query，每个是以l, r, x的形式，然后寻找a[l:r+1]之间x出现的次数，query间累和输出。注意TLE
给一个array和一个matrix。
matrix里面每一个vector<int>的形式必定是[l,r,target]，固定只有3个数。 然后要求统计array里index从l 到 r这个区间出现了多少次target这个数。 比如:
array = [1,1,2,3,2]
matrix = [[1,2,1], [2,4,2], [0,3,1]]
output : 5
因为在matrix[0], array的index 1到2区间出现了1 一次， matrix[1], array的index 2到4区间出现 2 两次。 matrx[2], array的index 0到3区间出现1 两次
这个题如果直接暴力解O(n*n)会有两个test case过不了。我是用hashmap<int, vector<pair<int,int>>>。 key是target， value是index区间。 这样走一遍array，每次确定一下当前index在不在区间里就行了。

findMinInArray

给int n, m，想象n*m的矩阵M, M[i,j] = (i+1)*(j+1)，0-based 一系列query，有三种类型，第一种是查询矩阵中最小的元素，第二、三分别是禁用某一行、列。
给一个matrix， 一个整数数组， matrix只有三列，[l,r,target], 要求在整数数组中寻找下标从l 到 r 中包含几个target，找到+1，最后输出个数总和。
栗子:
input: 1,2,1 [1,1,2,3,2]; output: 5 (1 + 2 + 2 = 5)
2,4,2
0,3,1 注意n2解法只能过15/18个test case

15. 给你一个2d array。其中array[j] = (i+1)*(j+1)。这个给定。 然后给一堆query，有三种不同的格式: 第一种是让你返回当前array中的最小值 第二种是让你把某一行disable
第三种是把某一列disable 当然disable了之后最小值就不能用了

```
// test 15
//
public static List<Integer> findMin(int m, int n, int[][] queries) {
    TreeMap<Integer, Integer> map = new TreeMap<Integer, Integer>();
    int[][] matrix = new int[m][n];
    for(int i = 0; i < m; i++) {
        for(int j = 0; j < n; j++) {
            matrix[i][j] = (i + 1) * (j + 1);
            map.put(matrix[i][j], map.getOrDefault(matrix[i][j], 0) + 1);
        }
    }

    List<Integer> res = new ArrayList<>();
    for(int[] query : queries) {
        if(query.length == 1) {
            for(int key : map.keySet()) {
                if(map.get(key) != 0) {
                    res.add(key);
                    break;
                }
            }
        } else if (query.length == 2) {
            // consider the row
            if(query[0] == 1) {
                int row = query[1];
                for(int j = 0; j < n; j++) {
                    if(matrix[row][j] != -1 && map.containsKey(matrix[row][j])) {
                        map.put(matrix[row][j], map.get(matrix[row][j]) - 1);
                        matrix[row][j] = -1;
                    }
                }
            } else {
                int col = query[1];
                for(int i = 0; i < m; i++) {
                    if(matrix[i][col] != -1 && map.containsKey(matrix[i][col])) {
                        map.put(matrix[i][col], map.get(matrix[i][col]) - 1);
                        matrix[i][col] = -1;
                    }
                }
            }
        }
    }
    return res;
}

public static void print2D(int[][] matrix) {
    int m = matrix.length;
    int n = matrix[0].length;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            System.out.print(matrix[i][j]);
        }
        System.out.println();
    }
}
```

diagonalsSort

diagonalsSort
int fun(int[][] a), 把一个 matrix 按斜线顺序重排
[8, 4, 1] [4, 4, 1] [4, 8, 9]
[4, 1, 1] --> [4, 8, 4] [4, 8, 9]

```
    public static void diagonalsSort(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        for(int d = 0; d < n; d++) {
            List<Integer> temp = new ArrayList();
            for(int i = 0; i < n - d; i++) {
                temp.add(matrix[i][i+d]);
            }
            Collections.sort(temp);
            for(int i = 0; i < n - d; i++) {
                matrix[i][i+d] = temp.get(i);
            }
            temp.clear();
            if(d != 0) {
                for(int i = 0; i < n - d; i++) {
                    temp.add(matrix[i+d][i]);
                }
                Collections.sort(temp);
                for(int i = 0; i < n - d; i++) {
                    matrix[i+d][i] = temp.get(i);
                }
            }
        }
    }
```

longestEqualSubArray

longestEqualSubarray
int fun(int[] a), a 由 1 和 0 组成. 求 0，1个数相同的subarray 最大长度.
用HashMap就好

```
    public static int longestEqualSubArray(int[] a) {
        for(int i = 0; i < a.length; i++) {
            if(a[i] == 0) a[i] = -1;
        }
        int[] prefixSum = new int[a.length + 1];
        for(int i = 0; i < a.length; i++) {
            prefixSum[i+1] = prefixSum[i] + a[i];
        }

        int res = 0;
        for(int i = 1; i <= a.length; i++) {
            for(int j = 0; j < i; j++) {
                if(prefixSum[i] == prefixSum[j]) {
                    res = Math.max(res, i - j);
                }
            }
        }
        return res;
    }
```

rameWindow

Given an int n, print the *** window frame of the number; Example: input -> n = 6
output -> [
"********", --> 8 *
"* *", -> 2 *加六个' ' (space)
"* *",
"* *",
"* *", "********"
]

Input -> n = 3; Output -> ["***“,
”* *“,
”***]

removeExactOneDigit

remove exact one digit char from string s or t, so that s < t; input: String s1,s2 (lower case letters and digits)
output: number of ways to remove the digit char.

```
public static int removeExactOneDigit(String s1, String s2) {
    if(s1 == null && s2 == null) {
        return 0;
    }
    if(s1 == null) {
        return s2.length();
    }
    if(s2 == null) {
        return 0;
    }
    if(s1.length() == 0 && s2.length() == 0) {
        return 0;
    }
    if(s1.length() == 0) {
        return s2.length();
    } else if(s2.length() == 0) {
        return 0;
    }
    int len1 = s1.length();
    int len2 = s2.length();
    int s1Arrow = 0;
    int s2Arrow = 0;
    int res = 0;
    if(s1.charAt(0) < s2.charAt(0)) {
        res = (s1.length() - 1) + (s2.length() - 1);
        //consider the first two
        if(compareStringVal(s1.substring(1), s2) > 0) {
            res++;
        }
```

```
            if(compareStringVal(s1, s2.substring(1)) > 0){
                res++;
            }

        } else if (s1.charAt(s1Arrow) == s2.charAt(s2Arrow)) {
            res = removeExactOneDigit(s1.substring(1), s2.substring(1));
            if(compareStringVal(s1.substring(1), s2) > 0) {
                res++;
            }
            if(compareStringVal(s1, s2.substring(1)) > 0){
                res++;
            }
        } else {
            if(compareStringVal(s1.substring(1), s2) > 0) {
                res = 1;
            } else {
                res = 0;
            }
        }
        return res;
    }

    public static int compareStringVal(String s1, String s2) {
        int len1 = s1.length();
        int len2 = s2.length();
        int s1Arrow = 0;
        int s2Arrow = 0;
        while (s1Arrow < len1 && s2Arrow < len2) {
            if(s1.charAt(s1Arrow) < s2.charAt(s2Arrow)) {
                return 1;
            } else if (s1.charAt(s1Arrow) == s2.charAt(s2Arrow)) {
                s1Arrow++;
                s2Arrow++;
            } else {
                return -1;
            }
        }
        if(s1Arrow < len1)
            return -1;
        else
            return 1;
    }
```

Other

sort elements in a matrix then re-organize the elements 好像没见过
sort: 1. by frequency of elements 2. by value of the element
re-organize: 矩阵里数字代表Order后element 所处位置，如order后顺序为[-4,-4,2,2,5,5,1,1,1], 对应的index为[1,2,3,4,5,6,7,8,9]
[[16,15,13,10],
[14,12,9,6],
[11,8,5,3],
[7,4,2,1]]
example:
input: matrix[][] (integers)
output: matrix[][](integers)

Test Code

```
public static void main(String[] args) {
    // // test 1
    // int number = 102;
    // System.out.println(number + " " + product_sum(number));
    // number = 55;
    // System.out.println(number + " " + product_sum(number));

    // // test 2
    // String[] words = {"hEllo##", "This^^"};
    // char[] valid = {'h', 'e', 'l', 'o', 'i', 't', 'h', 's'};
    // System.out.println(wordsIsValid(words, valid));

    // // // test 3
    // String a = "Hello my dear friend";
    // char[] b = {'h', 'e', 'l', 'o', 'i', 'y'};
    // System.out.println(brokenKeyboard(a, b));

    // // test 4
    // String s1 = "babzccc";
    // String s2 = "abczzzz";
    // System.out.println(compareString(s1, s2));

    // test 5
// int[] a = {1, 2, 3};
// int[] b = {3, 4};
// int[][] query = new int[][] {{1, 5}, {1, 1, 1}, {1, 5}};
// int[] result = coolFeature(a, b, query);
// for(int res : result)
//     System.out.println(res);

// test 6
// int[] A = {12, 3, 5, 34567};
// System.out.println(findEvenDigit(A));
//
//

// // test 7
// int[] A = {2, 2, 3, 3, 5};
// for(int a : findMostCommon(A))
//     System.out.println(a);
//

// // test 8
// int[] A = {1, 2, 3, 4, 9};
// int k = 5;
// System.out.println(maxRibbon(A, k));
//
//

// test 9
// int[] A = {1, 1, 2, 1, 5, 3, 2, 3};
// System.out.println(goodTuples(A));
//

// test 10
// int[][] A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
// rotateDiagonal(A, 3);
// for(int i = 0; i < A.length; i++) {
//     for(int j = 0; j < A[i].length; j++) {
//         System.out.print(A[i][j]);
//     }
//     System.out.println();
// }

// test 12
// String s = "120";
// System.out.println(divideSubString(s, 2) == 2);
// System.out.println(divideSubString(s, 1) == 2);
// s = "555";
// System.out.println(divideSubString(s, 1) == 1);


// test 13
// String s1 = "99";
// String s2 = "99";
// System.out.println(sumOfString(s1, s2));
// s1 = "199";
// s2 = "2";
// System.out.println(sumOfString(s1, s2));
// s1 = "2";
// s2 = "99";
// System.out.println(sumOfString(s1, s2));
//
//
// test 14
// int[] array = {1, 1, 2, 3, 2};
// int[][] matrix = {{1,2, 1}, {2, 4, 2}, {0, 3, 1}};
// System.out.println(matrixQuery(array, matrix));

// test 15
// int m = 3, n = 3;
// int[][] query = {{1}, {1, 2}, {2, 0}, {1, 0}, {1}};
// List<Integer> res = findMin(m, n, query);
// for(int ans : res) {
//     System.out.print(ans);
// }

// test 15
// diagonal sort
// int[][] matrix = {{8, 4, 1}, {4, 4, 1}, {4, 8, 9}};
// diagonalsSort(matrix);
// print2D(matrix);

//test 16
// int[] array = {0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1};
// System.out.println(longestEqualSubArray(array));
//

// test 17
String s1 = "heflo";
String s2 = "hhllo";
System.out.println(removeExactOneDigit(s1, s2));
s1 = "h";
s2 = "hhllo";
System.out.println(removeExactOneDigit(s1, s2));
```

```
        s1 = "hf";
        s2 = "hhllo";
        System.out.println(removeExactOneDigit(s1, s2));
        s1 = "hi";
        s2 = "hhllo";
        System.out.println(removeExactOneDigit(s1, s2));
    }
}
```