数上方的数都会比 K 小，则没有必要继续在第 col 列上寻找，令 col=col+1，重复步骤 2。

3．如果找到越界都没有发现与 K 相等的数，则返回 false。

具体请参看如下代码中的 isContains 方法：

```
public boolean isContains(int[][] matrix, int K) {
        int row = 0;
        int col = matrix[0].length - 1;
        while (row < matrix.length && col > -1) {
                if (matrix[row][col] == K) {
                        return true;
                } else if (matrix[row][col] > K) {
                        col--;
                } else {
                        row++;
                }
        }
        return false;
}
```

最长的可整合子数组的长度

【题目】

先给出可整合数组的定义。如果一个数组在排序之后，每相邻两个数差的绝对值都为 1，则该数组为可整合数组。例如，[5,3,4,6,2]排序之后为[2,3,4,5,6]，符合每相邻两个数差的绝对值都为 1，所以这个数组为可整合数组。

给定一个整型数组 arr，请返回其中最大可整合子数组的长度。例如，[5,5,3,2,6,4,3]的最大可整合子数组为[5,3,2,6,4]，所以返回 5。

【难度】

尉 ★★☆☆

【解答】

时间复杂度高但容易理解的做法。对 arr 中的每一个子数组 arr[i..j](0<=i<=j<=N-1)，都验证一下是否符合可整合数组的定义，也就是把 arr[i..j]排序一下，看是否依次递增且每次递增 1。然后在所有符合可整合数组定义的子数组中，记录最大的那个长度，返回即可。

需要注意的是，在考查每一个 arr[i..j]是否符合可整合数组定义的时候，都得把 arr[i..j]单独复制成一个新的数组，然后把这个新的数组排序、验证，而不能直接改变 arr 中元素的顺序。所以大体过程如下：

1．依次考查每一个子数组 arr[i..j](0<=i<=j<=N-1)，一共有 $O(N^2)$个。

2．对每一个子数组 arr[i..j]，复制成一个新的数组，记为 newArr，把 newArr 排序，然后验证是否符合可整合数组的定义，这一步代价为 $O(N\log N)$。

3．步骤 2 中符合条件的、最大的那个子数组的长度就是结果。

具体请参看如下代码中的 getLIL1 方法，时间复杂度为 $O(N^2)\times O(N\log N)->O(N^3 \log N)$。

```
public int getLIL1(int[] arr) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
        int len = 0;
        for (int i = 0; i < arr.length; i++) {
                for (int j = i; j < arr.length; j++) {
                        if (isIntegrated(arr, i, j)) {
                                len = Math.max(len, j - i + 1);
                        }
                }
        }
        return len;
}

public boolean isIntegrated(int[] arr, int left, int right) {
        int[] newArr = Arrays.copyOfRange(arr, left, right + 1); // O(N)
        Arrays.sort(newArr); // O(N*logN)
        for (int i = 1; i < newArr.length; i++) {
                if (newArr[i - 1] != newArr[i] - 1) {
                        return false;
                }
        }
        return true;
}
```

第一种方法严格按照题目的意思来验证每一个子数组是否是可整合数组，但是验证可整合数组真的需要如此麻烦吗？有没有更好的方法来加速验证过程？这也是本书提供方法的核心。判断一个数组是否是可整合数组还可以用以下方法来判断，一个数组中如果没有重复元素，并且如果最大值减去最小值，再加 1 的结果等于元素个数（max-min+1==元素个数），那么这个数组就是可整合数组。比如[3,2,5,6,4]，max-min+1=6-2+1=5==元素个数，所以这个数组是可整合数组。

这样，验证每一个子数组是否是可整合数组的时间复杂度可以从第一种方法的

$O(N\log N)$加速至 $O(1)$，整个过程的时间复杂度就可加速到 $O(N^2)$。具体请参看如下代码中的 getLIL2 方法。

```
public int getLIL2(int[] arr) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
        int len = 0;
        int max = 0;
        int min = 0;
        HashSet<Integer> set = new HashSet<Integer>(); // 判断重复
        for (int i = 0; i < arr.length; i++) {
                max = Integer.MIN_VALUE;
                min = Integer.MAX_VALUE;
                for (int j = i; j < arr.length; j++) {
                        if (set.contains(arr[j])) {
                                break;
                        }
                        set.add(arr[j]);
                        max = Math.max(max, arr[j]);
                        min = Math.min(min, arr[j]);
                        if (max - min == j - i) { // 新的检查方式
                                len = Math.max(len, j - i + 1);
                        }
                }
                set.clear();
        }
        return len;
}
```

不重复打印排序数组中相加和为给定值的所有二元组和三元组

【题目】

给定排序数组 arr 和整数 k，不重复打印 arr 中所有相加和为 k 的不降序二元组。

例如，arr=[-8,-4,-3,0,1,2,4,5,8,9]，k=10，打印结果为：

1,9

2,8

【补充题目】

给定排序数组 arr 和整数 k，不重复打印 arr 中所有相加和为 k 的不降序三元组。