$O(N\log N)$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。

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

-4,5,9

-3,4,9

-3,5,8

0,1,9

0,2,8

1,4,5

【难度】

尉 ★★☆☆

【解答】

利用排序后的数组的特点，打印二元组的过程可以用一个左指针和一个右指针不断向中间压缩的方式实现，具体过程为：

1．设置变量 left=0，right=arr.length-1。

2．比较 arr[left]+arr[right]的值(sum)与 k 的大小：

- 如果 sum 等于 k，打印“arr[left],arr[right]”，则 left++，right--。
- 如果 sum 大于 k，right--。
- 如果 sum 小于 k，left++。

3．如果 left<right，则一直重复步骤 2，否则过程结束。

那么如何保证不重复打印相同的二元组呢？只需在打印时增加一个检查即可，检查 arr[left]是否与它前一个值 arr[left-1]相等，如果相等就不打印。具体解释为：因为整体过程是从两头向中间压缩的过程，如果 arr[left]+arr[right]==k，又有 arr[left]==arr[left-1]，那么之前一定已经打印过这个二元组，此时无须重复打印。比如 arr=[1,1,1,9]，k=10。首先打印 arr[0]和 arr[3]的组合，接下来就不再重复打印 1 和 9 这个二元组。

具体过程请参看如下代码中的 printUniquePair 方法，时间复杂度 $O(N)$。

```
public void printUniquePair(int[] arr, int k) {
        if (arr == null || arr.length < 2) {
                return;
        }
        int left = 0;
        int right = arr.length - 1;
        while (left < right) {
```

```
                if (arr[left] + arr[right] < k) {
                        left++;
                } else if (arr[left] + arr[right] > k) {
                        right--;
                } else {
                        if (left == 0 || arr[left - 1] != arr[left]) {
                            System.out.println(arr[left] + "," + arr[right]);
                        }
                        left++;
                        right--;
                }
        }
}
```

三元组的问题类似于二元组的求解过程。

例如：

arr=[-8,-4,-3,0,1,2,4,5,8,9]，k=10。

- 当三元组的第一个值为-8 时，寻找-8 后面的子数组中所有相加为 18 的不重复二元组。
- 当三元组的第一个值为-4 时，寻找-4 后面的子数组中所有相加为 14 的不重复二元组。
- 当三元组的第一个值为-3 时，寻找-3 后面的子数组中所有相加为 13 的不重复二元组。

依此类推。

如何不重复打印相同的三元组呢？首先要保证每次寻找过程开始前，选定的三元组中第一个值不重复，其次就是和原问题的打印检查一样，要保证不重复打印二元组。

具体请参看如下代码中的 printUniqueTriad 方法，时间复杂度为 $O(N^2)$。

```
public void printUniqueTriad(int[] arr, int k) {
        if (arr == null || arr.length < 3) {
            return;
        }
        for (int i = 0; i < arr.length - 2; i++) {
            if (i == 0 || arr[i] != arr[i - 1]) {
                    printRest(arr, i, i + 1, arr.length - 1, k - arr[i]);
            }
        }
}

public void printRest(int[] arr, int f, int l, int r, int k) {
        while (l < r) {
            if (arr[l] + arr[r] < k) {
                l++;
```

```
            } else if (arr[l] + arr[r] > k) {
                r--;
            } else {
                if (l == f + 1 || arr[l - 1] != arr[l]) {
                    System.out.println(arr[f] + "," + arr[l] + "," + arr[r]);
                }
                l++;
                r--;
            }
        }
    }
```

未排序正数数组中累加和为给定值的最长子数组长度

【题目】

给定一个数组 arr，该数组无序，但每个值均为正数，再给定一个正数 k。求 arr 的所有子数组中所有元素相加和为 k 的最长子数组长度。

例如，arr=[1,2,1,1,1]，k=3。

累加和为 3 的最长子数组为[1,1,1]，所以结果返回 3。

【难度】

尉 ★★☆☆

【解答】

最优解可以做到时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。首先用两个位置来标记子数组的左右两头，记为 left 和 right，开始时都在数组的最左边(left=0,right=0)。整体过程如下：

1. 开始时变量 left=0，right=0，代表子数组 arr[left..right]。
2. 变量 sum 始终表示子数组 arr[left..right]的和。开始时 sum=arr[0]，即 arr[0..0]的和。
3. 变量 len 一直记录累加和为 k 的所有子数组中最大子数组的长度。开始时，len=0。
4. 根据 sum 与 k 的比较结果决定是 left 移动还是 right 移动，具体如下：

- 如果 sum==k，说明 arr[left..right]累加和为 k，如果 arr[left..right]长度大于 len，则更新 len，此时因为数组中所有的值都为正数，那么所有从 left 位置开始，在 right 之后的位置结束的子数组，即 arr[left..i(i>right)]，累加和一定大于 k。所以，令 left 加 1，这表示我们开始考查以 left 之后的位置开始的子数组，同时令 sum-=arr[left]，