```
                stack.push(i);
            }
            while (!stack.isEmpty()) {
                int j = stack.pop();
                int k = stack.isEmpty() ? -1 : stack.peek();
                int curArea = (height.length - k - 1) * height[j];
                maxArea = Math.max(maxArea, curArea);
            }
            return maxArea;
    }
```

最大值减去最小值小于或等于 num 的子数组数量

【题目】

给定数组 arr 和整数 num，共返回有多少个子数组满足如下情况：

max(arr[i..j]) - min(arr[i..j]) <= num

max(arr[i..j])表示子数组 arr[i..j]中的最大值，min(arr[i..j])表示子数组 arr[i..j]中的最小值。

【要求】

如果数组长度为 N，请实现时间复杂度为 $O(N)$的解法。

【难度】

校 ★★★☆

【解答】

首先介绍普通的解法，找到 arr 的所有子数组，一共有 $O(N^2)$个，然后对每一个子数组做遍历找到其中的最小值和最大值，这个过程时间复杂度为 $O(N)$，然后看看这个子数组是否满足条件。统计所有满足的子数组数量即可。普通解法容易实现，但是时间复杂度为 $O(N^3)$，本书不再详述。最优解可以做到时间复杂度 $O(N)$，额外空间复杂度 $O(N)$，在阅读下面的分析过程之前，请读者先阅读本章“生成窗口最大值数组”问题，本题所使用到的双端队列结构与解决“生成窗口最大值数组”问题中的双端队列结构含义基本一致。

生成两个双端队列 qmax 和 qmin。当子数组为 arr[i..j]时，qmax 维护了窗口子数组 arr[i..j]的最大值更新的结构，qmin 维护了窗口子数组 arr[i..j]的最小值更新的结构。当子数组 arr[i..j]向右扩一个位置变成 arr[i..j+1]时，qmax 和 qmin 结构可以在 $O(1)$的时间内更新，并且可以

在 $O(1)$的时间内得到 arr[i..j+1]的最大值和最小值。当子数组 arr[i..j]向左缩一个位置变成 arr[i+1..j]时，qmax 和 qmin 结构依然可以在 $O(1)$的时间内更新，并且在 $O(1)$的时间内得到 arr[i+1..j]的最大值和最小值。

通过分析题目满足的条件，可以得到如下两个结论：

- 如果子数组 arr[i..j]满足条件，即 max(arr[i..j])-min(arr[i..j])<=num，那么 arr[i..j]中的每一个子数组，即 arr[k..l](i<=k<=l<=j)都满足条件。我们以子数组 arr[i..j-1]为例说明，arr[i..j-1]最大值只可能小于或等于 arr[i..j]的最大值，arr[i..j-1]最小值只可能大于或等于 arr[i..j]的最小值，所以 arr[i..j-1]必然满足条件。同理，arr[i..j]中的每一个子数组都满足条件。
- 如果子数组 arr[i..j]不满足条件，那么所有包含 arr[i..j]的子数组，即 arr[k..l](k<=i<=j<=l)都不满足条件。证明过程同第一个结论。

根据双端队列 qmax 和 qmin 的结构性质，以及如上两个结论，设计整个过程如下：

1．生成两个双端队列 qmax 和 qmin，含义如上文所说。生成两个整型变量 i 和 j，表示子数组的范围，即 arr[i..j]。生成整型变量 res，表示所有满足条件的子数组数量。

2．令 j 不断向右移动（j++），表示 arr[i..j]一直向右扩大，并不断更新 qmax 和 qmin 结构，保证 qmax 和 qmin 始终维持动态窗口最大值和最小值的更新结构。一旦出现 arr[i..j]不满足条件的情况，j 向右扩的过程停止，此时 arr[i..j-1]、arr[i..j-2]、arr[i..j-3]、...、arr[i..i]一定都是满足条件的。也就是说，所有必须以 arr[i]作为第一个元素的子数组，满足条件的数量为 *j-i* 个。于是令 res+=j-i。

3．当进行完步骤 2，令 i 向右移动一个位置，并对 qmax 和 qmin 做出相应的更新，qmax 和 qmin 从原来的 arr[i..j]窗口变成 arr[i+1..j]窗口的最大值和最小值的更新结构。然后重复步骤 2，也就是求所有必须以 arr[i+1]作为第一个元素的子数组中，满足条件的数量有多少个。

4．根据步骤 2 和步骤 3，依次求出以 arr[0]、arr[1]、...、arr[N-1]作为第一个元素的子数组中满足条件的数量分别有多少个，累加起来的数量就是最终的结果。

上述过程中，所有的下标值最多进 qmax 和 qmin 一次，出 qmax 和 qmin 一次。i 和 j 的值也不断增加，并且从来不减小。所以整个过程的时间复杂度为 $O(N)$。

最优解全部实现请参看如下代码中的 getNum 方法。

```
public int getNum(int[] arr, int num) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
```

```
        LinkedList<Integer> qmin = new LinkedList<Integer>();
        LinkedList<Integer> qmax = new LinkedList<Integer>();
        int i = 0;
        int j = 0;
        int res = 0;
        while (i < arr.length) {
                while (j < arr.length) {
                        while (!qmin.isEmpty() && arr[qmin.peekLast()] >= arr[j]) {
                                qmin.pollLast();
                        }
                        qmin.addLast(j);
                        while (!qmax.isEmpty() && arr[qmax.peekLast()] <= arr[j]) {
                                qmax.pollLast();
                        }
                        qmax.addLast(j);
                        if (arr[qmax.getFirst()] - arr[qmin.getFirst()] > num) {
                                break;
                        }
                        j++;
                }
                if (qmin.peekFirst() == i) {
                        qmin.pollFirst();
                }
                if (qmax.peekFirst() == i) {
                        qmax.pollFirst();
                }
                res += j - i;
                i++;
        }
        return res;
}
```