- 用求累加数组的最大累加和的方式得到每一步的最大子矩阵的累加和。
- 每一步的累加数组可以利用前一步求出的累加数组很方便地更新得到。

如果矩阵大小为 $N \times N$ 的，以上全部过程的时间复杂度为 $O(N^3)$，具体请参看如下代码中的 maxSum 方法。

```
public int maxSum(int[][] m) {
        if (m == null || m.length == 0 || m[0].length == 0) {
                return 0;
        }
        int max = Integer.MIN_VALUE;
        int cur = 0;
        int[] s = null; // 累加数组
        for (int i = 0; i != m.length; i++) {
                s = new int[m[0].length];
                for (int j = i; j != m.length; j++) {
                        cur = 0;
                        for (int k = 0; k != s.length; k++) {
                                s[k] += m[j][k];
                                cur += s[k];
                                max = Math.max(max, cur);
                                cur = cur < 0 ? 0 : cur;
                        }
                }
        }
        return max;
}
```

在数组中找到一个局部最小的位置

【题目】

定义局部最小的概念。arr 长度为 1 时，arr[0]是局部最小。arr 的长度为 $N(N>1)$时，如果 arr[0]<arr[1]，那么 arr[0]是局部最小；如果 arr[N-1]<arr[N-2]，那么 arr[N-1]是局部最小；如果 $0<i<N-1$，既有 arr[i]<arr[i-1]，又有 arr[i]<arr[i+1]，那么 arr[i]是局部最小。

给定无序数组 arr，已知 arr 中任意两个相邻的数都不相等。写一个函数，只需返回 arr 中任意一个局部最小出现的位置即可。

【难度】

尉 ★★☆☆

【解答】

本题可以利用二分查找做到时间复杂度为 $O(\log N)$、额外空间复杂度为 $O(1)$，步骤如下：

1．如果 arr 为空或者长度为 0，返回-1 表示不存在局部最小。

2．如果 arr 长度为 1 或者 arr[0]<arr[1]，说明 arr[0]是局部最小，返回 0。

3．如果 arr[N-1]<arr[N-2]，说明 arr[N-1]是局部最小，返回 N-1。

4．如果 arr 长度大于 2 且 arr 的左右两头都不是局部最小，则令 left=1，right=N-2，然后进入步骤 5 做二分查找。

5．令 mid=(left+right)/2，然后进行如下判断：

1)如果 arr[mid]>arr[mid-1]，可知在 arr[left..mid-1]上肯定存在局部最小，令 right=mid-1，重复步骤 5。

2）如果不满足 1)，但 arr[mid]>arr[mid+1]，可知在 arr[mid+1..right]上肯定存在局部最小，令 left=mid+1，重复步骤 5。

3）如果既不满足 1)，也不满足 2)，那么 arr[mid]就是局部最小，直接返回 mid。

6．步骤 5 一直进行二分查找，直到 left==right 时停止，返回 left 即可。

如此可见，二分查找并不是数组有序时才能使用，只要你能确定二分两侧的某一侧肯定存在你要找的内容，就可以使用二分查找。具体过程请参看如下的 getLessIndex 方法。

```
public int getLessIndex(int[] arr) {
        if (arr == null || arr.length == 0) {
                return -1; // 不存在
        }
        if (arr.length == 1 || arr[0] < arr[1]) {
                return 0;
        }
        if (arr[arr.length - 1] < arr[arr.length - 2]) {
                return arr.length - 1;
        }
        int left = 1;
        int right = arr.length - 2;
        int mid = 0;
        while (left < right) {
                mid = (left + right) / 2;
                if (arr[mid] > arr[mid - 1]) {
                        right = mid - 1;
                } else if (arr[mid] > arr[mid + 1]) {
                        left = mid + 1;
                } else {
                        return mid;
                }
```

```
            }
            return left;
    }
```

数组中子数组的最大累乘积

【题目】

给定一个 double 类型的数组 arr，其中的元素可正、可负、可 0，返回子数组累乘的最大乘积。例如，arr=[-2.5，4，0，3，0.5，8，-1]，子数组[3，0.5，8]累乘可以获得最大的乘积 12，所以返回 12。

【难度】

尉 ★★☆☆

【解答】

本题可以做到时间复杂度为 $O(N)$、额外空间复杂度为 $O(1)$。所有的子数组都会以某一个位置结束，所以，如果求出以每一个位置结尾的子数组最大的累乘积，在这么多最大累乘积中最大的那个就是最终的结果。也就是说，结果=Max{以 arr[0]结尾的所有子数组的最大累乘积，以 arr[1]结尾的所有子数组的最大累乘积……以 arr[arr.length-1]结尾的所有子数组的最大累乘积}。

如何快速求出所有以 i 位置结尾(arr[i])的子数组的最大乘积呢？假设以 arr[i-1]结尾的最小累乘积为 min，以 arr[i-1]结尾的最大累乘积为 max。那么，以 arr[i]结尾的最大累乘积只有以下三种可能：

- 可能是 max*arr[i]。max 既然表示以 arr[i-1]结尾的最大累乘积，那么当然有可能以 arr[i]结尾的最大累乘积是 max*arr[i]。例如，[3,4,5]在算到 5 的时候。
- 可能是 min*arr[i]。min 既然表示以 arr[i-1]结尾的最小累乘积，当然有可能 min 是负数，而如果 arr[i]也是负数，两个负数相乘的结果也可能很大。例如，[-2,3,-4]在算到-4 的时候。
- 可能仅是 arr[i]的值。以 arr[i]结尾的最大累乘积并不一定非要包含 arr[i]之前的数。例如，[0.1,0.1,100]在算到 100 的时候。

这三种可能的值中最大的那个就作为以 i 位置结尾的最大累乘积，最小的作为最小累