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。

|  |  |
| --- | --- |
| 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 | public int getLessIndex(int[] arr) {      if (arr == null || arr.length == 0) {          return -1; // no exist      }      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。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public double maxProduct(double[] arr) {      if (arr == null || arr.length == 0) {          return 0;      }      double max = arr[0];      double min = arr[0];      double res = arr[0];      double maxEnd = 0;      double minEnd = 0;      for (int i = 1; i < arr.length; ++i) {          maxEnd = max \* arr[i];          minEnd = min \* arr[i];          max = Math.max(Math.max(maxEnd, minEnd), arr[i]);          min = Math.min(Math.min(maxEnd, minEnd), arr[i]);          res = Math.max(res, max);      }      return res;  } |

统计完全二叉树的节点数

【题目】

给定一棵完全二叉树的头节点head，返回这棵树的节点个数。

【要求】

如果完全二叉树的节点数为N，请实现时间复杂度低于O(N)的解法。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public int nodeNum(Node head) {      if (head == null) {          return 0;      }      return bs(head, 1, mostLeftLevel(head, 1));  }  public int bs(Node node, int l, int h) {      if (l == h) {          return 1;      }      if (mostLeftLevel(node.right, l + 1) == h) {          return (1 << (h - l)) + bs(node.right, l + 1, h);      } else {          return (1 << (h - l - 1)) + bs(node.left, l + 1, h);      }  }  public int mostLeftLevel(Node node, int level) {      while (node != null) {          level++;          node = node.left;      }      return level - 1;  } |

给定两个有序数组arr1和arr2，两个数组长度都为N，求两个数组中所有数的上中位数。

例如：

arr1 = {1,2,3,4};

arr2 = {3,4,5,6};

一共8个数则上中位数是第4个数，所以返回3。

arr1 = {0,1,2};

arr2 = {3,4,5};

一共6个数则上中位数是第3个数，所以返回2。

要求：时间复杂度O(logN)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public static int getUpMedian(int[] arr1, int[] arr2) {      if (arr1 == null || arr2 == null || arr1.length != arr2.length) {          throw new RuntimeException("Your arr is invalid!");      }      return findProcess(arr1, 0, arr1.length - 1, arr2, 0, arr2.length - 1);  }  public static int findProcess(int[] arr1, int start1, int end1, int[] arr2,          int start2, int end2) {      if (start1 == end1) {          return Math.min(arr1[start1], arr2[start2]);      }      // 元素个数为奇数，则offset为0；元素个数为偶数，则offset为1；      int offset = ((end1 - start1 + 1) & 1) ^ 1;      int mid1 = (start1 + end1) / 2;      int mid2 = (start2 + end2) / 2;      if (arr1[mid1] > arr2[mid2]) {          return findProcess(arr1, start1, mid1, arr2, mid2 + offset, end2);      } else if (arr1[mid1] < arr2[mid2]) {          return findProcess(arr1, mid1 + offset, end1, arr2, start2, mid2);      } else {          return arr1[mid1];      }  } |

给定两个有序数组arr1和arr2，在给定一个整数k，返回两个数组的所有数中第K小的数。

例如：

arr1 = {1,2,3,4,5};

arr2 = {3,4,5};

K = 1;

因为1为所有数中最小的，所以返回1；

arr1 = {1,2,3};

arr2 = {3,4,5,6};

K = 4;

因为3为所有数中第4小的数，所以返回3；

要求：如果arr1的长度为N，arr2的长度为M，时间复杂度请达到O(log(min{M,N}))。

|  |  |
| --- | --- |
| 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  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | public static int findKthNum(int[] arr1, int[] arr2, int kth) {      if (arr1 == null || arr2 == null) {          throw new RuntimeException("Your arr is invalid!");      }      if (kth < 1 || kth > arr1.length + arr2.length) {          throw new RuntimeException("K is invalid!");      }      int[] longArr = arr1.length >= arr2.length ? arr1 : arr2;      int[] shortArr = arr1.length < arr2.length ? arr1 : arr2;      int lenL = longArr.length;      int lenS = shortArr.length;      if (kth <= lenS) {          return getUpMedian(shortArr, 0, kth - 1, longArr, 0, kth - 1);      }      if (kth > lenL) {          if (shortArr[kth - lenL - 1] >= longArr[lenL - 1]) {              return shortArr[kth - lenL - 1];          }          if (longArr[kth - lenS - 1] >= shortArr[lenS - 1]) {              return longArr[kth - lenS - 1];          }          return getUpMedian(shortArr, kth - lenL, lenS - 1, longArr, kth - lenS, lenL - 1);      }      if (longArr[kth - lenS - 1] >= shortArr[lenS - 1]) {          return longArr[kth - lenS - 1];      }      return getUpMedian(shortArr, 0, lenS - 1, longArr, kth - lenS, kth - 1);  }  public static int getUpMedian(int[] arr1, int start1, int end1, int[] arr2,          int start2, int end2) {      if (start1 == end1) {          return Math.min(arr1[start1], arr2[start2]);      }      int offset = ((end1 - start1 + 1) & 1) ^ 1;      int mid1 = (start1 + end1) / 2;      int mid2 = (start2 + end2) / 2;      if (arr1[mid1] > arr2[mid2]) {          return getUpMedian(arr1, start1, mid1, arr2, mid2 + offset, end2);      } else if (arr1[mid1] < arr2[mid2]) {          return getUpMedian(arr1, mid1 + offset, end1, arr2, start2, mid2);      } else {          return arr1[mid1];      }  } |