关键知识点

归并排序

java代码

```
public static int[] mergeSort(int[] arr) {
    if (arr.length < 2) return arr;
    //计算中间位置
    int mid = arr.length / 2;
    //分解为左右两部分,分别排序
    int[] left = Arrays.copyOfRange(arr, 0, mid);
    left = mergeSort(left);
    int[] right=Arrays.copyOfRange(arr,mid,arr.length);
    right = mergeSort(right);
    //合并两个排序后的数组为一个数组
    return merge(left, right);
}
private static int[] merge(int[] l, int[] r) {
    int[] result = new int[l.length + r.length];
    int lIndex = 0;
    int rIndex = 0;
    for (int i = 0; i < result.length; i++) {
        if (lIndex < l.length && rIndex < r.length) {
            if (l[lIndex] <= r[rIndex]) {
                result[i] = l[lIndex++];
            } else {
                result[i] = r[rIndex++];
            }
        } else if (lIndex >= l.length) {
            result[i] = r[rIndex++];
        } else {
            result[i] = l[lIndex++];
        }
    }
    return result;
}
```

基本解法

Java代码

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    int m = nums1.length;
```

```
    int n = nums2.length;
    //定义指针p1,p2 分别指代nums1和nums2的当前元素
    int p1 = 0, p2 = 0;
    //定义中位数m1和m2，指代当前第i-1大的数和第i大的数
    int m1 = 0, m2 = 0;
    for (int i = 0; i <= (m + n) / 2; i++) {
        m1 = m2; //指针右移
        //若nums1未处理完，并且 nums2已处理完 或 p1元素小于p2元素
        //则第i大的元素为nums1当前元素
        if (p1 < m && (p2 >= n || nums1[p1] < nums2[p2])) {
            m2 = nums1[p1++];
        } else { //否则第i大的元素为nums2当前元素
            m2 = nums2[p2++];
        }
    }
    //若m+n为奇数，返回m2，若m+n为偶数，返回(m1+m2)/2.0
    if (((m + n) % 2) == 0)
        return (m1 + m2) / 2.0;
    else
        return m2;
}
```

更优解知识点

二分查找

Java代码

```
//二分搜索算法，在a[start] ~ a[end]中搜索key
public int binarySearch(int key, int a[], int start, int end) {
    if (start > end) //未找到key，返回-1
        return -1;

    int m = (start + end) / 2;
    if (a[m] == key) //找到key，返回key的id
        return m;

    if (a[m] > key)
        return binarySearch(key, a, start, m - 1);//在m左侧继续搜索

    return binarySearch(key, a, m + 1, end);//在m右侧继续搜索
}
```

优化解法

Java代码

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int m = nums1.length, n = nums2.length;
        //较短的数组在前，确保m <= n
        if (m > n) return findMedianSortedArrays(nums2, nums1);
        //定义p、q为nums1分界线范围，共m+1个可能的划分位置
        int p = 0, q = m;
        //nums1、num2的分界位置
        int i=0,j=0;
        while (p <= q) {
            //二分确定nums1当前分界位置
            i = (p + q) / 2;
            //根据i确定nums2的分界位置，使得左侧元素数-右侧元素数为0或1
            j = (m + n + 1) / 2 - i;
            //nums1右侧最小值小于nums2左侧最大值，nums1划分位置在[i+1, q]之间
            if (j != 0 && i != m && nums1[i] < nums2[j - 1])
                p = i + 1;
            else if (i != 0 && j != n && nums1[i - 1] > nums2[j])
                //nums1左侧最大值大于nums2右侧最小值，nums1划分位置在[p, i-1]之间
                q = i - 1;
            else {//当前划分位置左侧的最大值小于右侧的最小值，满足要求
                break;
            }
        }

        //m+n为奇数，返回左侧的最大值
        //左侧最大值：三种情况 nums1为空、nums2为空、都不为空
        int maxLeft = i==0?nums2[j - 1]:(j==0?nums1[i - 1]:Math.max(nums1[i -
1], nums2[j - 1]));
        if ((m + n) % 2 == 1) return maxLeft;
        //m+n为偶数，返回左侧最大值与右侧最小值的平均
        //右侧最小值：三种情况 nums1为空、nums2为空、都不为空
        int minRight = i==m?nums2[j]:(j==n?nums1[i]:Math.min(nums1[i ],
nums2[j]));
        return  (maxLeft + minRight) / 2.0;
    }
```

C++代码

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int m = nums1.size();
        int n = nums2.size();
    //确保m <= n
    if (m > n)
```

```
            return findMedianSortedArrays(nums2, nums1);
        int left = 0, right = m, ansi = -1;
        // maxLeft：前一部分的最大值
        // minRight：后一部分的最小值
        int maxLeft = 0, minRight = 0;

        while (left <= right) {
            // 前一部分包含 nums1[0 .. i-1] 和 nums2[0 .. j-1]
            // 后一部分包含 nums1[i .. m-1] 和 nums2[j .. n-1]
            int i = (left + right) / 2;
            int j = (m + n + 1) / 2 - i;

            // nums_im1, nums_i, nums_jm1, nums_j 分别表示 nums1[i-1], nums1[i],
nums2[j-1], nums2[j]
            int nums_im1 = (i == 0 ? INT_MIN : nums1[i - 1]);
            int nums_i = (i == m ? INT_MAX : nums1[i]);
            int nums_jm1 = (j == 0 ? INT_MIN : nums2[j - 1]);
            int nums_j = (j == n ? INT_MAX : nums2[j]);

            if (nums_im1 <= nums_j) {
                ansi = i;
                maxLeft = max(nums_im1, nums_jm1);
                minRight = min(nums_i, nums_j);
                left = i + 1;
            }
            else {
                right = i - 1;
            }
        }
        return (m + n) % 2 == 0 ? (maxLeft + minRight) / 2.0 : maxLeft;
    }
};
```

Python代码

```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) ->
float:
        if len(nums1) > len(nums2):
            return self.findMedianSortedArrays(nums2, nums1)

        infinty = 2**40
        m, n = len(nums1), len(nums2)
        left, right, ansi = 0, m, -1
        # maxLeft：前一部分的最大值
        # minRight：后一部分的最小值
        maxLeft, minRight = 0, 0
```

```
        while left <= right:
            # 前一部分包含 nums1[0 .. i-1] 和 nums2[0 .. j-1]
            # // 后一部分包含 nums1[i .. m-1] 和 nums2[j .. n-1]
            i = (left + right) // 2
            j = (m + n + 1) // 2 - i

            # nums_im1, nums_i, nums_jm1, nums_j 分别表示 nums1[i-1], nums1[i],
nums2[j-1], nums2[j]
            nums_im1 = (-infinty if i == 0 else nums1[i - 1])
            nums_i = (infinty if i == m else nums1[i])
            nums_jm1 = (-infinty if j == 0 else nums2[j - 1])
            nums_j = (infinty if j == n else nums2[j])

            if nums_im1 <= nums_j:
                ansi = i
                maxLeft, minRight = max(nums_im1, nums_jm1), min(nums_i,
nums_j)
                left = i + 1
            else:
                right = i - 1

        return (maxLeft + minRight) / 2 if (m + n) % 2 == 0 else maxLeft
```