王争的算法训练营

习题课：排序

配套习题（12）：

面试题 10.01. 合并排序的数组 （简单）

242. 有效的字母异位词（简单）

1502. 判断能否形成等差数列（简单）

252. 会议室 （简单）

56. 合并区间（中等）

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面 （简单）

75. 颜色分类（中等）

147. 对链表进行插入排序（中等）

148. 排序链表（中等） 链表上的归并排序

215. 数组中的第K个最大元素（中等）

面试题 17.14. 最小K个数 （中等）

剑指 Offer 51. 数组中的逆序对（困难）

习题课：排序

面试题 10.01. 合并排序的数组 （简单）

给定两个排序后的数组 A 和 B，其中 A 的末端有足够的缓冲空间容纳 B。 编写一个方法，将 B 合并入 A 并排序。

初始化 A 和 B 的元素数量分别为 *m* 和 *n*。

A= 4、7、8、9、一、一、一
B=1、2、5

示例:

```
输入:
A = [1,2,3,0,0,0], m = 3
B = [2,5,6],       n = 3

输出: [1,2,2,3,5,6]
```

说明:

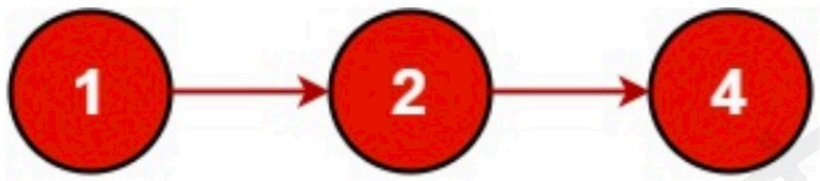
- `A.length == n + m`

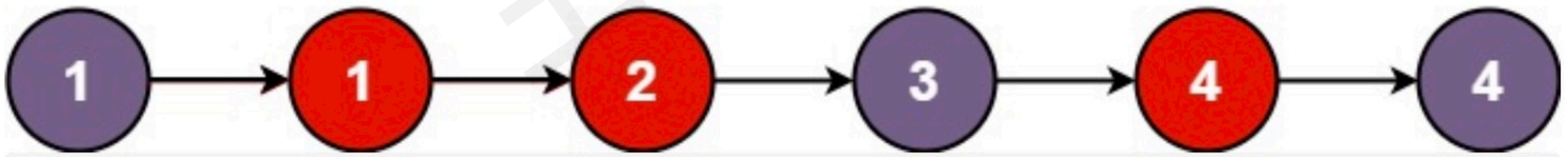
```java
class Solution {
    public void merge(int[] A, int m, int[] B, int n) {
        int k = m+n-1;
        int i = m-1;
        int j = n-1;
        while (i >= 0 && j >= 0) {
            if (A[i] >= B[j]) {
                A[k--] = A[i];
                i--;
            } else {
                A[k--] = B[j];
                j--;
            }
        }
        while (i >= 0) {
            A[k--] = A[i--];
        }
        while (j >= 0) {
            A[k--] = B[j--];
        }
    }
}
```

21. 合并两个有序链表 （简单）

将两个升序链表合并为一个新的 **升序** 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

示例 1：






```
输入：l1 = [1,2,4], l2 = [1,3,4]
输出：[1,1,2,3,4,4]
```

```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
         if (l2 == null) return l1;
         ListNode p1 = l1;
         ListNode p2 = l2;
         ListNode head = new ListNode();//虚拟头节点
         ListNode tail = head;
         while (p1 != null && p2 != null) {
             if (p1.val <= p2.val) {
                 tail.next = p1;
                 tail = p1;
                 p1 = p1.next;
             } else {
                 tail.next = p2;
                 tail = p2;
                 p2 = p2.next;
             }
         }
         // 如果p1还没处理完，就把剩下的直接接到tail后面
         if (p1 != null) tail.next = p1;
         // 同理
         if (p2 != null) tail.next = p2;
         return head.next;
    }
}
```

习题课：排序

242. 有效的字母异位词（简单）

1. 排序
2. 哈希

给定两个字符串 *s* 和 *t* ，编写一个函数来判断 *t* 是否是 *s* 的字母异位词。

示例 1:

```
输入: s = "anagram", t = "nagaram"
输出: true
```

示例 2:

```
输入: s = "rat", t = "car"
输出: false
```

说明:
你可以假设字符串只包含小写字母。

```java
class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) {
            return false;
        }
        char[] str1 = s.toCharArray();
        char[] str2 = t.toCharArray();
        Arrays.sort(str1);
        Arrays.sort(str2);
        for (int i = 0; i < str1.length; ++i) {
            if (str1[i] != str2[i]) return false;
        }
        return true;
    }
}
```

习题课：排序

1502. 判断能否形成等差数列（简单）

给你一个数字数组 `arr` 。

如果一个数列中，任意相邻两项的差总等于同一个常数，那么这个数列就称为 **等差数列** 。

如果可以重新排列数组形成等差数列，请返回 `true` ；否则，返回 `false` 。

示例 1：

```
输入：arr = [3,5,1]
输出：true
解释：对数组重新排序得到 [1,3,5] 或者 [5,3,1] ，任意相邻两项的差分别为 2 或 -2 ，可以形成等差数列。
```

示例 2：

```
输入：arr = [1,2,4]
输出：false
解释：无法通过重新排序得到等差数列。
```

提示：

- `2 <= arr.length <= 1000`
- `-10^6 <= arr[i] <= 10^6`

```java
class Solution {
    public boolean canMakeArithmeticProgression(int[] arr) {
        Arrays.sort(arr);
        int diff = arr[1] - arr[0];
        for (int i = 2; i < arr.length; ++i) {
            if (arr[i]-arr[i-1] != diff) return false;
        }
        return true;
    }
}
```

习题课：排序

252. 会议室 （简单）

给定一个会议时间安排的数组 `intervals` ，每个会议时间都会包括开始和结束的时间 `intervals[i] = [start_i, end_i]` ，请你判断一个人是否能够参加这里面的全部会议。

示例 1：

```
输入：intervals = [[0,30],[5,10],[15,20]]
输出：false
```

示例 2：

```
输入：intervals = [[7,10],[2,4]]
输出：true
```

提示：

- $0 <= intervals.length <= 10^4$
- $intervals[i].length == 2$
- $0 <= start_i < end_i <= 10^6$

```java
class Solution {
    public boolean canAttendMeetings(int[][] intervals) {
        Arrays.sort(intervals, new Comparator<int[]>() {
            public int compare(int[] i1, int[] i2) {
                return i1[0] - i2[0];
            }
        });
        for (int i = 1; i < intervals.length; ++i) {
            if (intervals[i][0] < intervals[i-1][1]) return false;
        }
        return true;
    }
}
```

习题课：排序

56. 合并区间（中等）

以数组 `intervals` 表示若干个区间的集合，其中单个区间为 `intervals[i] = [start_i, end_i]` 。请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。

1、暴力解法：时间复杂度O(n^2)
3、先排序：时间复杂度O(nlogn)

示例 1：

```
输入：intervals = [[1,3],[2,6],[8,10],[15,18]]
输出：[[1,6],[8,10],[15,18]]
解释：区间 [1,3] 和 [2,6] 重叠，将它们合并为 [1,6].
```

示例 2：

```
输入：intervals = [[1,4],[4,5]]
输出：[[1,5]]
解释：区间 [1,4] 和 [4,5] 可被视为重叠区间。
```

提示：

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start_i <= end_i <= 10^4`

```java
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, new Comparator<int[]>() {
            public int compare(int[] interval1, int[] interval2) {
                return interval1[0] - interval2[0];
            }
        });
        List<int[]> result = new ArrayList<int[]>();
        int curLeft = intervals[0][0];
        int curRight = intervals[0][1];
        for (int i = 1; i < intervals.length; ++i) {
            if (intervals[i][0] <= curRight) {
                if (intervals[i][1] > curRight) {
                    curRight = intervals[i][1];
                }
            } else {
                result.add(new int[]{curLeft, curRight});
                curLeft = intervals[i][0];
                curRight = intervals[i][1];
            }
        }
        result.add(new int[]{curLeft, curRight});
        return result.toArray(new int[result.size()][]);
    }
}
```

习题课：排序

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面 （简单）

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

示例：

```
输入：nums = [1,2,3,4]
输出：[1,3,2,4]
注：[3,1,2,4] 也是正确的答案之一。
```

提示：

1. `0 <= nums.length <= 50000`
2. `1 <= nums[i] <= 10000`

```java
class Solution {
    public int[] exchange(int[] nums) {
        int i = 0;
        int j = nums.length-1;
        while (i < j) {
            if (nums[i] % 2 == 1) {
                i++;
                continue;
            }
            if (nums[j] % 2 == 0) {
                j--;
                continue;
            }
            int tmp = nums[i];
            nums[i] = nums[j];
            nums[j] = tmp;
            i++;
            j--;
        }
        return nums;
    }
}
```

时间复杂度?

75. 颜色分类（中等）

给定一个包含红色、白色和蓝色，一共 `n` 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 `0` 、 `1` 和 `2` 分别表示红色、白色和蓝色。

示例 1：

```
输入：nums = [2,0,2,1,1,0]
输出：[0,0,1,1,2,2]
```

```java
class Solution {
    public void sortColors(int[] nums) {
        int p = 0;
        int q = nums.length-1;
        while (p < q) {
            if (nums[p] != 2) {
                p++;
                continue;
            }
            if (nums[q] == 2) {
                q--;
                continue;
            }
            swap(nums, p, q);
            p++;
            q--;
        }

        int i = 0;
        int j = p;
        if (nums[j] == 2) j--;
        while (i < j) {
            if (nums[i] == 0) {
                i++;
                continue;
            }
            if (nums[j] == 1) {
                j--;
                continue;
            }
            swap(nums, i, j);
            i++;
            j--;
        }
    }

    private void swap(int[] nums, int p, int q) {
        int tmp = nums[p];
        nums[p] = nums[q];
        nums[q] = tmp;
    }
}
```

习题课：排序

147. 对链表进行插入排序（中等）

对链表进行插入排序。

插入排序的动画演示如上。从第一个元素开始，该链表可以被认为已经部分排序（用黑色表示）。
每次迭代时，从输入数据中移除一个元素（用红色表示），并原地将其插入到已排好序的链表中。

插入排序算法：

1. 插入排序是迭代的，每次只移动一个元素，直到所有元素可以形成一个有序的输出列表。
2. 每次迭代中，插入排序只从输入数据中移除一个待排序的元素，找到它在序列中适当的位置，并将其插入。
3. 重复直到所有输入数据插入完为止。

示例 1：

```
输入：4->2->1->3
输出：1->2->3->4
```

```java
class Solution {
    public ListNode insertionSortList(ListNode head) {
        if (head == null) return null;
        // 存储已经排好序的节点
        ListNode newHead = new ListNode(Integer.MIN_VALUE, null);

        // 遍历节点
        ListNode p = head;
        while (p != null) {
            ListNode tmp = p.next;

            // 寻找p节点插入的位置,插入到哪个节点后面
            ListNode q = newHead; // 初始化值
            while (q.next != null && q.next.val <= p.val) { // 循环结束条件
                q = q.next;
            }
            // 将p节点插入
            p.next = q.next;
            q.next = p;

            p = tmp;
        }
        return newHead.next;
    }
}
```

时间复杂度?
空间复杂度?

习题课：排序

王争的算法训练营

148. 排序链表（中等） 链表上的归并排序

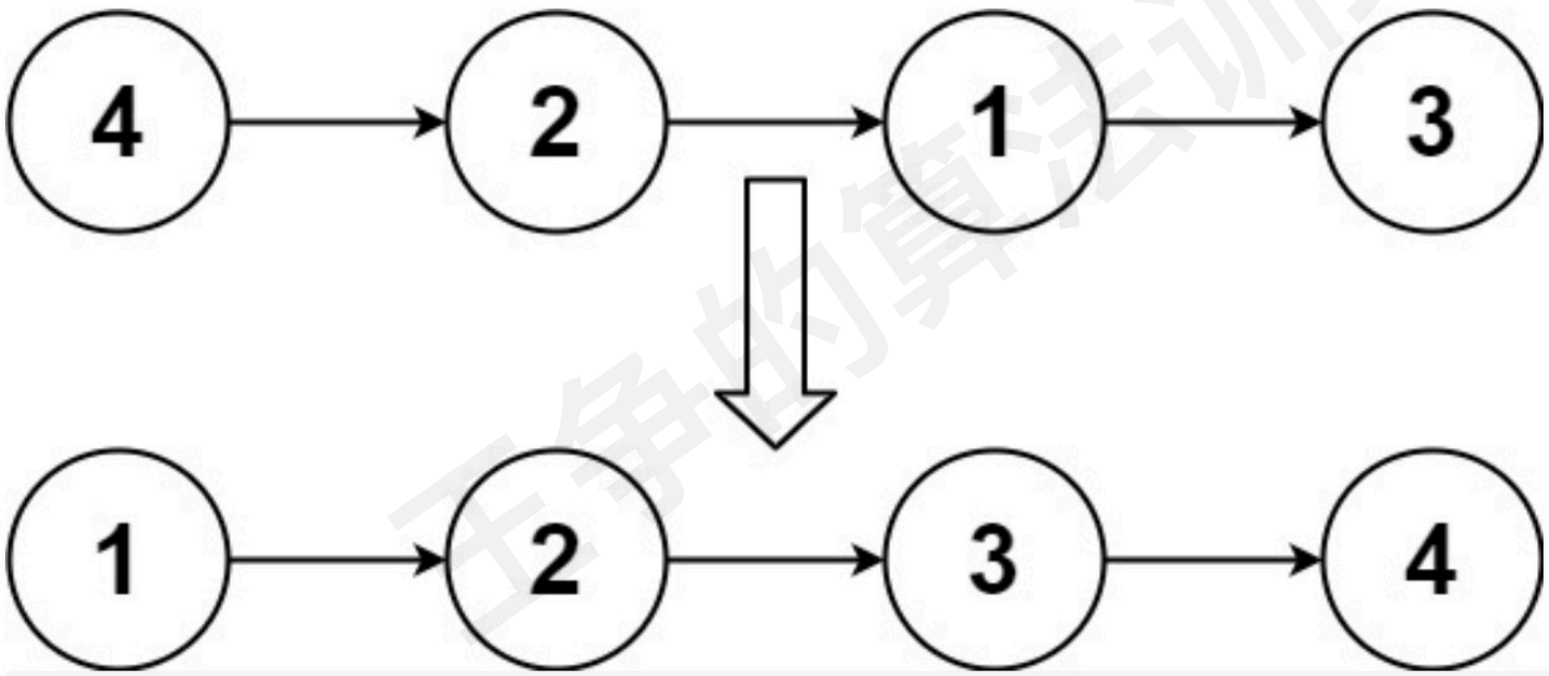
建议把链表上的冒泡、选择、快排都写一下

给你链表的头结点 `head` ，请将其按 **升序** 排列并返回 **排序后的链表** 。

进阶：

- 你可以在 `O(n log n)` 时间复杂度和常数级空间复杂度下，对链表进行排序吗？

示例 1：




```
输入：head = [4,2,1,3]
输出：[1,2,3,4]
```

1. 递归解法：

```
class Solution {
    Public ListNode mergeSortList(ListNode head) {
        if (head == null) return null;
        if (head.next == null) return head;
        ListNode midNode = findMidNode(head);
        ListNode nextNode = midNode.next;
        midNode.next = null;
        ListNode leftHead = mergeSortList(head);
        ListNode rightHead = mergeSortList(nextNode);
        return mergeList(leftHead, rightHead);
    }
}
```

时间复杂度?

空间复杂度?

```
private ListNode findMidNode(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while (fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}

private ListNode mergeList(ListNode headA, ListNode headB) {
    ListNode newHead = new ListNode();
    ListNode tail = newHead;
    ListNode pa = headA;
    ListNode pb = headB;
    while (pa != null && pb != null) {
        if (pa.val <= pb.val) {
            tail.next = pa;
            tail = tail.next;
            pa = pa.next;
        } else {
            tail.next = pb;
            tail = tail.next;
            pb = pb.next;
        }
    }
    if (pa != null) tail.next = pa;
    if (pb != null) tail.next = pb;
    return newHead.next;
}
```

1. 非递归解法：

```
class Solution {
    public ListNode sortList(ListNode head) {
        int n = len(head);
        int step = 1;
        while (step < n) {
            ListNode newHead = new ListNode(); // 结果链表
            ListNode tail = newHead;
            ListNode p = head;
            while (p != null) {
                // [p, q]
                ListNode q = p;
                int count = 1;
                while (q != null && count < step) {
                    q = q.next;
                    count++;
                }
                if (q == null || q.next == null) {//这一轮合并结束了
                    tail.next = p;
                    break;
                }
                //[q+1, r]
                ListNode r = q.next;
                count = 1;
                while (r != null && count < step) {
                    r = r.next;
                    count++;
                }
                // 保存下一个step的起点
                ListNode tmp = null;
                if (r != null) {
                    tmp = r.next;
                }
                // merge[p, q][q+1, r]
                ListNode[] headAndTail = merge(p, q, r);
                tail.next = headAndTail[0];
                tail = headAndTail[1];
                p = tmp;
            }
            head = newHead.next;
            step *= 2;
        }
        return head;
    }
}
```

```java
private int len(ListNode head) {
    if (head == null) return 0;
    int n = 1;
    ListNode p = head;
    while (p != null) {
        n++;
        p = p.next;
    }
    return n;
}

private ListNode[] merge(ListNode p, ListNode q, ListNode r) {
    ListNode newHead = new ListNode();
    ListNode tail = newHead;
    ListNode pa = p;
    ListNode pb = q.next;
    q.next = null;
    if (r != null) {
        r.next = null;
    }
    while (pa != null && pb != null) {
        if (pa.val <= pb.val) {
            tail.next = pa;
            tail = tail.next;
            pa = pa.next;
        } else {
            tail.next = pb;
            tail = tail.next;
            pb = pb.next;
        }
    }
    if (pa != null) {
        tail.next = pa;
        tail = q;
    }
    if (pb != null) {
        tail.next = pb;
        tail = r;
    }
    return new ListNode[]{newHead.next, tail};
}
```

215. 数组中的第K个最大元素（中等）已讲

在未排序的数组中找到第 **k** 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

示例 1:

```
输入: [3,2,1,5,6,4] 和 k = 2
输出: 5
```

示例 2:

```
输入: [3,2,3,1,2,4,5,5,6] 和 k = 4
输出: 4
```

说明:

你可以假设 k 总是有效的，且 1 ≤ k ≤ 数组的长度。

```java
class Solution {
    public int findKthLargest(int[] nums, int k) {
        if (nums.length < k) return -1;
        return quickSort(nums, 0, nums.length-1, k);
    }

    private int quickSort(int[] nums, int p, int r, int k) {
        if (p > r) return -1;
        int q = partition(nums, p, r);
        if (q-p+1 == k) {
            return nums[q];
        } else if (q-p+1 < k) {
            return quickSort(nums, q+1, r, k-(q-p+1));
        } else {
            return quickSort(nums, p, q-1, k);
        }
    }

    private int partition(int[] nums, int p, int r) {
        int i = p-1;
        int j = p;
        while (j < r) {
            if (nums[j] > nums[r]) {
                swap(nums, j, i+1);
                i++;
            }
            j++;
        }
        swap(nums, i+1, r);
        return i+1;
    }

    private void swap(int[] nums, int i, int j) {
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
}
```

习题课：排序

面试题 17.14. 最小K个数 （中等）

1/排序
2/大顶堆
3/利用快排

设计一个算法，找出数组中最小的k个数。以任意顺序返回这k个数均可。

示例：

```
输入：  arr = [1,3,5,7,2,4,6,8], k = 4
输出：  [1,2,3,4]
```

提示：

- `0 <= len(arr) <= 100000`
- `0 <= k <= min(100000, len(arr))`

```java
class Solution {
    private int[] result;
    private int count = 0;
    public int[] smallestK(int[] arr, int k) {
        if (k == 0 || arr.length < k) return new int[0];
        result = new int[k];
        quickSort(arr, 0, arr.length-1, k);
        return result;
    }

    private void quickSort(int[] nums, int p, int r, int k) {
        if (p > r) {
            return;
        }
        int q = partition(nums, p, r);
        if (q-p+1 == k) {
            for (int i = p; i <= q; ++i) {
                result[count++] = nums[i];
            }
        } else if (q-p+1 < k) {
            for (int i = p; i <= q; ++i) {
                result[count++] = nums[i];
            }
            quickSort(nums, q+1, r, k-(q-p+1));
        } else {
            quickSort(nums, p, q-1, k);
        }
    }

}
```

时间复杂度

空间复杂度

```java
    private int partition(int[] nums, int p, int r) {
        int i = p-1;
        int j = p;
        while (j < r) {
            if (nums[j] < nums[r]) {
                swap(nums, j, i+1);
                i++;
            }
            j++;
        }
        swap(nums, i+1, r);
        return i+1;
    }

    private void swap(int[] nums, int i, int j) {
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
```

习题课：排序

剑指 Offer 51. 数组中的逆序对（困难）

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

示例 1:

```
输入: [7,5,6,4]
输出: 5
```

限制:

```
0 <= 数组长度 <= 50000
```

暴力解法：时间复杂度是O(n^2)

逆序对个数=逆序度，排序的过程就是不断的减小逆序度的过程，我们只要在排序的过程中，记录每步操作逆序度降低的个数，累加起来就能得到原始数据的逆序度。

```java
class Solution {
    int reverseCount = 0;
    public int reversePairs(int[] nums) {
        mergeSort(nums, 0, nums.length-1);
        return reverseCount;
    }

    private void mergeSort(int[] nums, int p, int r) {
        if (p >= r) return;
        int q = (p+r)/2;
        mergeSort(nums, p, q);
        mergeSort(nums, q+1, r);
        merge(nums, p, q, r);
    }

    private int merge(int[] nums, int p, int q, int r) {
        int[] tmp = new int[r-p+1];
        int i = p;
        int j = q+1;
        int k = 0;
        while (i <= q && j <= r) {
            if (nums[j] < nums[i]) {
                reverseCount += (q-i+1);
                tmp[k++] = nums[j];
                j++;
            } else {
                tmp[k++] = nums[i];
                i++;
            }
        }
        while (j <= r) {
            tmp[k++] = nums[j];
            j++;
        }
        while (i <= q) {
            tmp[k++] = nums[i];
            i++;
        }
        for (i = 0; i < r-p+1; ++i) {
            nums[i+p] = tmp[i];
        }
        return reverseCount;
    }
}
```

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料