【第九课】快速排序

剑指 Offer 51. 数组中的逆序对

思路：这是归并排序的模板题，直接套用模板，只有三步；左边处理一下，得到左边的信息；
右边处理一下，得到右边的信息；最后再处理，横跨左右两边的信息

```
 /**
 * @param {number[]} nums
 * @return {number}
 */

let temp = [];// 因为用到的是归并排序，归并排序得用到一个额外的存储数叫temp
var reversePairs = function(nums) {
    // 把temp的存储大小扩展成和nums的一样大
    while(temp.length < nums.length) temp.push(0);
    // 对应下标
    return countReversePairs(nums, 0 , nums.length - 1)// 首尾下标传进去；
};
var countReversePairs = function(nums, leftRoot ,rightRoot){//待排序数组，从l-r的统计区间
    if(leftRoot >= rightRoot) return 0;//空区间/只包含一个元素的区间，的时候，逆序对的个数为0
    let mid = (leftRoot+rightRoot) >> 1;//右移动1位就是除以2；计算中间数的位置
    let ans = 0;
    ans += countReversePairs(nums,leftRoot,mid);//左边逆序对的个数
    ans += countReversePairs(nums, mid+1,rightRoot);//右边逆序对的个数
    // 加上横跨两边的逆序对的数量
    let k = leftRoot,p1 = leftRoot,p2 = mid+1;//分别指向左右区间的第一位；
    while((p1 <= mid) || (p2 <= rightRoot)){  //第一个区间不为空，第二个区间不为空
        if((p2 > rightRoot) ||(p1<= mid && nums[p1] <= nums[p2])){ //第一个区间的元素放进来
            temp[k++] = nums[p1++];
        }else{
            temp[k++] = nums[p2++];//将第二个区间放进来，这个时候就可以统计逆序对的数量
            ans += (mid - p1 + 1);//左区间剩余的逆序对的数量
        }
    }
    //将temp的元素拷贝到原数组里面，对应下下标，temp是从外面定义的一个全局的元素，nums从l到r，temp也是从l到r,所以这个数组的空间大小是有序的
    for(let i = 0; i <= rightRoot;i++) nums[i] = temp[i];
    return ans;//ans 就是从l -r 的逆序数的数量
}
```

148. 排序链表

思路：将当前的链表分成前后两部分，对两部分分别排序，最后将两个有序的链表进行排序合并最后返回就行了。

```
 /**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
const merge = (head1, head2) => {
    const dummyHead = new ListNode(0);
    let temp = dummyHead, temp1 = head1, temp2 = head2;
    while (temp1 !== null && temp2 !== null) {
        if (temp1.val <= temp2.val) {
            temp.next = temp1;
            temp1 = temp1.next;
        } else {
            temp.next = temp2;
            temp2 = temp2.next;
        }
        temp = temp.next;
    }
    if (temp1 !== null) {
        temp.next = temp1;
    } else if (temp2 !== null) {
        temp.next = temp2;
    }
    return dummyHead.next;
}

const toSortList = (head, tail) => {
    if (head === null) {
        return head;
    }
    if (head.next === tail) {
        head.next = null;
        return head;
    }
    let slow = head, fast = head;
```

```
    while (fast !== tail) {
        slow = slow.next;
        fast = fast.next;
        if (fast !== tail) {
            fast = fast.next;
        }
    }
    const mid = slow;
    return merge(toSortList(head, mid), toSortList(mid, tail));
}

var sortList = function(head) {
    return toSortList(head, null);
};
```

1305. 两棵二叉搜索树中的所有元素

思路：我们知道二叉搜索树就是二叉排序树的特性：中序遍历最后返回的是一个有序序列。这道题就是中序遍历第一棵树，接着遍历中序遍历第二棵树， 将两个有序序列合并成一个有序序列

```
 /**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root1
 * @param {TreeNode} root2
 * @return {number[]}
 */
//二叉搜索树就是二叉排序树的特性：中序遍历最后返回的是一个有序序列，然后就是中序遍历第一棵树，接着遍历中序遍历第二棵树， 将两个有序序列合并成一个有序序列
var getAllElements = function(root1, root2) {
    let temp = [];//结果数组
    let lnums = [],rnums = [];//l从第一棵树中中序遍历出来的有序序列；R第二棵树
    getNum(root1,lnums);
    getNum(root2,rnums);
    let p1= 0, p2 = 0;//两个指针
    // 当前还有元素没有被合并进去的结果数组的时候
    while(p1 < lnums.length || p2 < rnums.length){
        // 这个时候判断，什么时候把左边放进去，什么时候把右边的放进去
        // 当右边为空 或者 左边还有元素并且
```

```
        if((p2 >= rnums.length) || (p1 < lnums.length && lnums[p1] < rnums[p2]))
{
            temp.push(lnums[p1++]);
        }else{
            temp.push(rnums[p2++])//右边元素放到结果数组的末尾
        }
    }
    return temp;
};
// 这里来实现中序遍历的过程
var getNum = function(root,nums){//传两个值，一个是树的根节点地址，一个是结果数组
    if(root == null) return;//当前节点为空
    getNum(root.left,nums);//先中序遍历左子树
    nums.push(root.val);//将当前节点的值放到结果数组的后一位
    getNum(root.right,nums);//中序遍历右子树
    return;
}
```

面试题 04.08. 首个共同祖先

思路：这个题的是用来复习寻找公共祖先的。一共 三种情况，要么p q 有一个是root；要么p, q 分别在 root 左孩子，右孩子上；要么，p，q z都在root的同一侧；

```
 /**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {TreeNode}
 */


// 5和1 的祖先是3，6和4 的祖先就不是3了，就是5了；
// 这个题用一笔递归就可以得到这个过程
// 当前函数代表的返回含义：
// 如果p q 存在于当前节点的左右两侧，左侧可以找到p值，右侧可以找到q值；
// 如果左右子树查找的值不为空，证明当前当前查找的节点就是最近公共祖先
//如果左子树为空，右子树不为空， 证明我在某种一个子树中找到了某一个节点，把这个结果正常返回就行；
```

```
var lowestCommonAncestor = function(root, p, q) {
    if(root == null) return null;
    if(root  == p || root == q)return root;
    let rootLeft = lowestCommonAncestor(root.left,p,q) //在左子树找到结果
    let rootRight = lowestCommonAncestor(root.right,p,q)// 在右子树找到结果
    if(rootLeft != null && rootRight != null) return root;
    // 左子树不为空并且右子树也不为空在左子树中找到了一个p跟q,在右子树中找到了一个p跟q，当前节点就是最近公共祖先
    if(rootLeft != null) return rootLeft;
    return rootRight;
}
```

1302. 层数最深叶子节点的和

思路：非常经典的递归题，拿它可以练习递归技巧。技巧就是，记录两个值，一个是已经查找节点最大深度，第二个是现在记录到的最大深度出节点的和值

```
 /**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
// 技巧：记录两个值，1个是已经找到的当前节点的最大深度，一个是现在记录到的最大深度的和值
var deepestLeavesSum = function(root) {
    let ans = [0],maxDepth = [-1];
    getResult(root,ans,0,maxDepth);
    return ans[0];
};
var getResult = function(root,ans,depth,maxDepth){
    if(root == null) {
        return;
    }
    if(maxDepth[0] < depth){
        maxDepth[0] = depth;
        ans[0] = root.val;
    }else{
        if(maxDepth[0] == depth){
```

```
            ans[0] += root.val;
        }
    }
    getResult(root.left,ans,depth+1,maxDepth);
    getResult(root.right,ans,depth+1,maxDepth);
}
```