【第十课】归并排序__第二节

彩蛋题目：OJ 上面题号232题 【https://oj.kaikeba.com/playground/232】

23. 合并K个升序链表

思路：也是找出中间数、左区间、右区间，然后定义三个指针，对数据进行比较，比较规则就是i.val < j.val。

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode[]} lists
 * @return {ListNode}
 */
var mergeKLists = function (lists) {
    return merge(lists, 0, lists.length - 1);
};
var merge = function (lists, l, r) {
    if (l == r) return lists[l];
    if (l > r) return null;
    let mid = (l + r) >> 1;
    return mergeTwoLists(merge(lists, l, mid), merge(lists, mid + 1, r));
}
var mergeTwoLists = function (a, b) {
    if (a == null || b == null) return a == null ? b : a;
    let head = new ListNode(0);
    let tail = head, i = a, j = b;
    while (i != null && j != null) {
        if (i.val < j.val) {
            tail.next = i;
            i = i.next;
        } else {
            tail.next = j;
            j = j.next;
        }
        tail = tail.next;
    }
    tail.next = i == null ? j : i;
    return head.next;
}
```

1508. 子数组和排序后的区间和

思路：我们先遍历整个数组，把数组里面的所有区间和都计算出来，然后进行排序，最后对10^9 + 7 取模后返回。

```
/**
 * @param {number[]} nums
 * @param {number} n
 * @param {number} left
 * @param {number} right
 * @return {number}
 */
var rangeSum = function(nums, n, left, right) {
    let arr=[]
    // 找子数组和
    for(let i=0;i<nums.length;i++){
        arr.push(nums[i])
        let tmp = nums[i]
        for(let j=i+1;j<nums.length;j++){
            tmp = tmp + nums[j]
            arr.push(tmp)
        }
    }
    // 排序
    let sArr = arr.sort((a,b)=> a-b)
    let sum = 0
    for(let i = left ; i<=right ;i++){
        sum = (sum+sArr[i-1])%1000000007
    }
    return sum
};
```

327. 区间和的个数

思路：这道题就是给你一个数组，让你计算出来有多少个区间和值是在给出的值域里面。接下来把所有的区间和值给它列出来采用归并排序的方式，能够得到左右两个数组排序后的形式，以及对应的下标对数量。

```
const countRangeSumRecursive = (sum, lower, upper, left, right) => { //递归
    if (left === right) {
        return 0;
    } else {
        const mid = Math.floor((left + right) / 2);
        const n1 = countRangeSumRecursive(sum, lower, upper, left, mid);
        const n2 = countRangeSumRecursive(sum, lower, upper, mid + 1, right);
        let ret = n1 + n2;

        // 首先统计下标对的数量
        let i = left;
        let l = mid + 1;
        let r = mid + 1;
        while (i <= mid) {
            while (l <= right && sum[l] - sum[i] < lower) l++;
            while (r <= right && sum[r] - sum[i] <= upper) r++;
            ret += (r - l);
            i++;
        }

        // 随后合并两个排序数组
        const sorted = new Array(right - left + 1);
        let p1 = left, p2 = mid + 1;
```

```
        let p = 0;
        while (p1 <= mid || p2 <= right) {
            if (p1 > mid) {
                sorted[p++] = sum[p2++];
            } else if (p2 > right) {
                sorted[p++] = sum[p1++];
            } else {
                if (sum[p1] < sum[p2]) {
                    sorted[p++] = sum[p1++];
                } else {
                    sorted[p++] = sum[p2++];
                }
            }
        }
        for (let i = 0; i < sorted.length; i++) {
            sum[left + i] = sorted[i];
        }
        return ret;
    }
}
var countRangeSum = function(nums, lower, upper) {
    let s = 0;
    const sum = [0];
    for(const v of nums) {
        s += v;
        sum.push(s);
    }
    return countRangeSumRecursive(sum, lower, upper, 0, sum.length - 1);
};
```

315. 计算右侧小于当前元素的个数

思路： 给出了一个数组，求出每个数组前面有多少小于他的数；这个用归并排序求，就是按着三步走，第一步，求出左边有多少个元素小于他，第二步右边有多少个元素小于他，现在求横跨左右区间的。之前的归并排序是从小到大排，现在的是从大到小排序；因为是从大到小排序，假设我现在已经得到了左边的信息，得到了右边的信息，如何求得横跨左右两边的信息。当把第一个区间的元素，放到结果数组里面。那么右区间的剩余的元素数量就是右边区间小于当前元素的数量。

```
var countSmaller = function (nums) {
 if (!nums.length) return []
 let objArr = [] //对象和index
 let resArr = [] //存放次数
 for (let i = 0; i < nums.length; i++) {
   const ele = nums[i]
   let obj = {
     num: ele,
     index: i
   }
   resArr.push(0)
   objArr.push(obj)
 }
 sliceNums(objArr)
 return resArr
 function sliceNums(nums) {
   let mid = nums.length >> 1
   let left = nums.slice(0, mid)
```

```
    let right = nums.slice(mid, nums.length)
    if (nums.length === 1) return nums
    return merge(sliceNums(left), sliceNums(right))
  }
  function merge(left, right) {
    let res = []
    // 定义“后有序数组”中一个指针
    let j = 0
    while (left.length && right.length) {
      if (left[0].num > right[0].num) {
        res.push(right[0])
        right.shift()
        j++
      } else {
        // “前有序数组” 的元素出列的时候，数一数 “后有序数组” 已经出列了多少元素
        resArr[left[0].index] += j
        res.push(left[0])
        left.shift()
      }
    }
    while (left.length) {
      // 同理
      resArr[left[0].index] += j
      res.push(left[0])
      left.shift()
    }
    while (right.length) {
      res.push(right[0])
      right.shift()
      j++
    }
    return res
  }
}
```

53. 最大子序和

思路：在序列中找到一个子序列，让这个子序列的区间和是最大值。只要是区间和，就用到前缀和，区间和=前缀和两项相减。其实维护snum[j]-snum[i]的最小值，先算的是前缀和，再算的是以每个位置结尾的最大和值。

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var maxSubArray = function(nums) {
  const len = nums.length;
  let max = nums[0];
  let min = 0;
  let sum = 0;
  for (let i = 0; i < len; i++) {
    sum += nums[i];
    if (sum - min > max) max = sum - min;
    if (sum < min) {
      min = sum;
    }
```

```
  }

  return max;
}
```