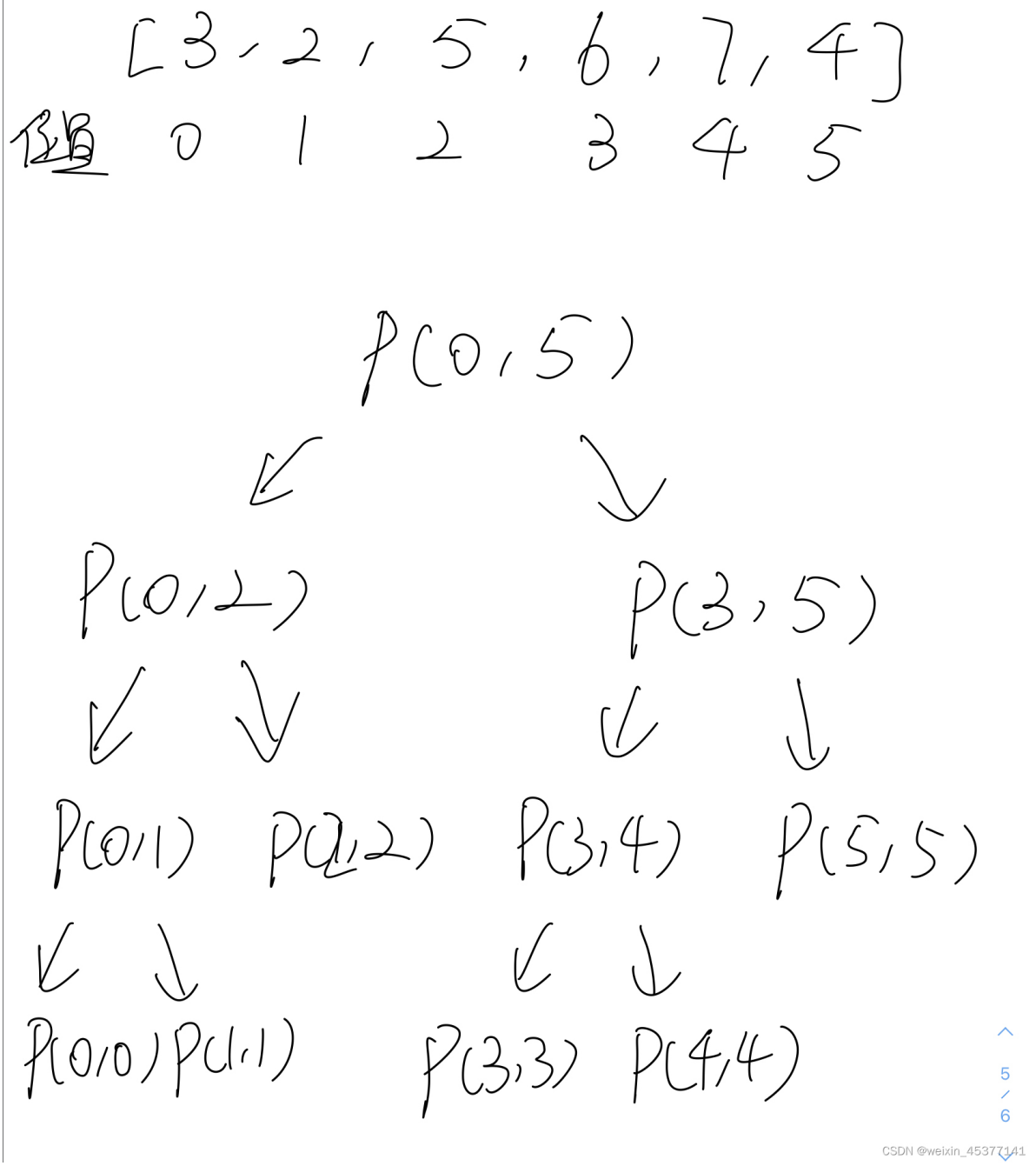
# 递归

问题：求数组中的最大值

```
//arr[L..R]范围上的最大值
public static int process(int[] arr, int L, int R){
    if (L == R){
        return arr[L];
    }//若只有一个数，直接返回
    int mid = L +((R - L) >> 1);//求出中点，防止溢出，如L+R溢出，也可写成mid=L+(R-L)/2
    int leftMax = process(arr, L, mid);
    int rightMax = process(arr, mid + 1, R);
    return Math.max(leftMax, rightMax);
}
```

举例：






将递归过程理解为一个多叉树，计算所有数节点的过程可理解为利用栈进行了一次后续遍历，每个节点都通过子节点的汇总的信息进行向上返回。栈空间是整棵树的高度。

**master公式**

$T(N) = a* T(N/b)+o(N^d)$

T(N):母问题数据量为N；

a：调用次数

子问题的规模都是N/b

o(N^d)：除去子问题的调用外剩下的过程的时间复杂度

例如刚才举的例子T(N) =2*T(N/2)+O(1)

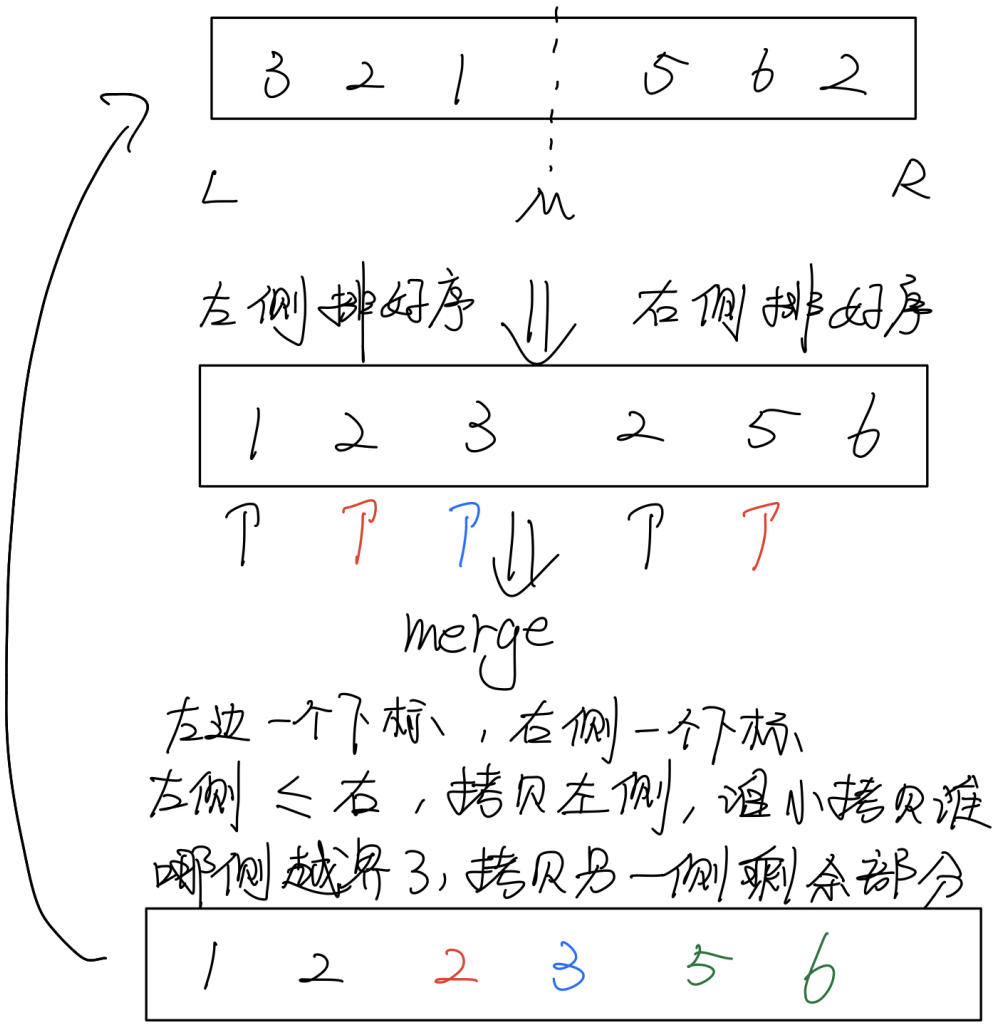
$\log_a b < d$　　$O(N^2)$

$\log_a b > d$ $O(N^{\log_a b})$

$\log_a b = d$ $O(N*\log N)$

## 归并排序

即就是一个简单递归，左边排好序，右边排好序，让整体有序，让整体有序的过程利用了外排序的方法。






```
public static void process(int[] arr, int L, int R){
    if (L == R){
    return;
    }
    int mid = L + ((R - L) >> 1);
    process(arr, L, mid);//左侧排好序
    process(arr, mid + 1, R);//右侧排好序
    merge(arr, L, mid, R);//连在一起
}

public static void merge(int[] arr, int L, int M, int R){
    int[] help = new int[R - L + 1];
    int i = 0;
    int p1 = L;
    int p2 = M + 1;//左右各一个指针
    while(p1 <= M && p2 <= R){
        help[i++]= arr[p1] <=arr[p2] ? arr[p1++] : arr[p2++];
```

```
        }
        //一侧越界，则将另一侧剩下的数填充
        while(p1 <= M){
            help[i++] = arr[p1++];
        }
        while(p2 <= R){
            help[i++] = arr[p2++];
        }
        for(i = 0; i < help.length; i++){
            arr[L + i] = help[i];
        }
    }
```

利用master公式：`T(N) = 2T(N/2) + O(N)`

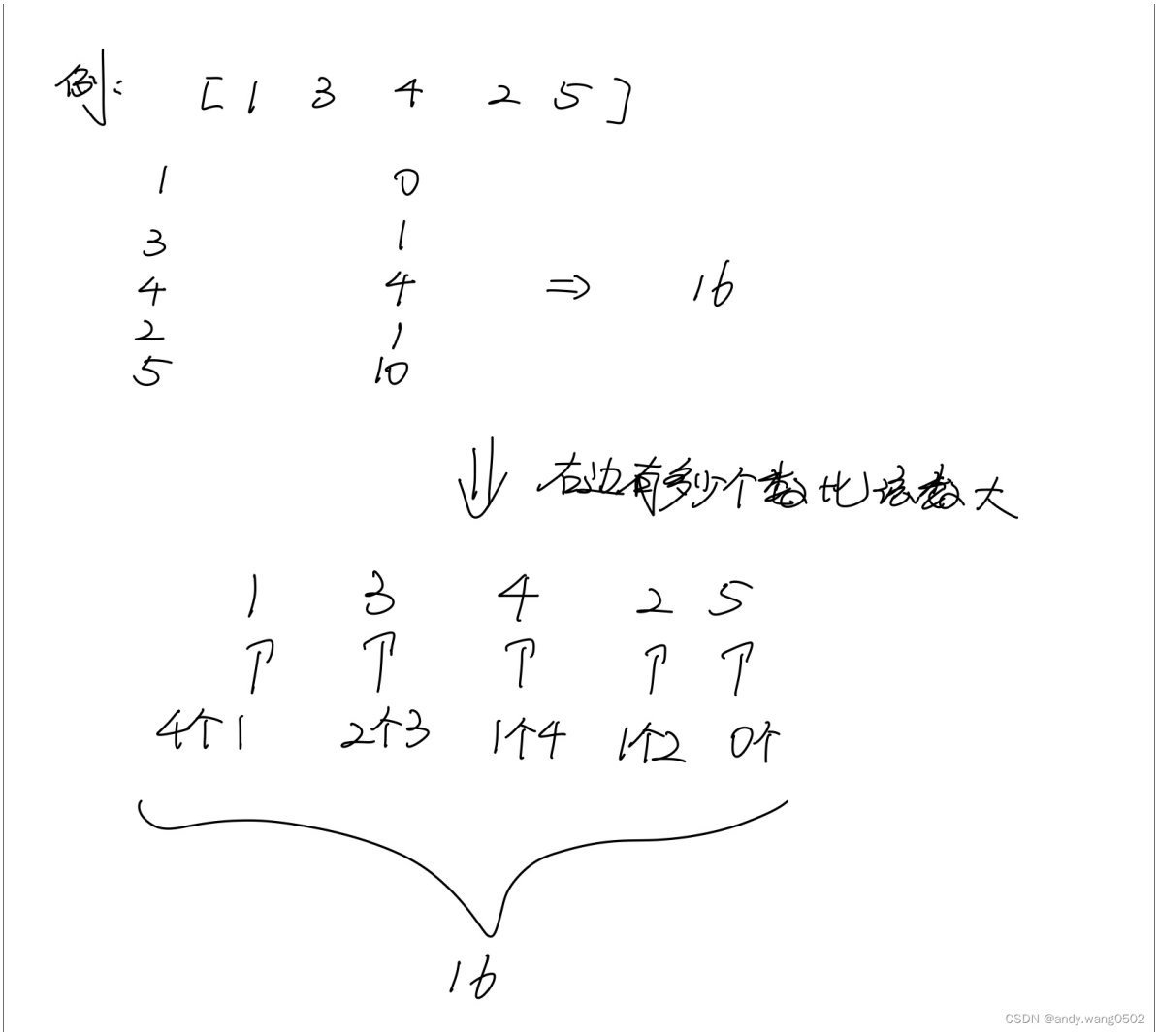
a=2，b = 2，d =1，即 $\log_a b = d$ 时间复杂度`O(N*logN)` 空间复杂度O(N)

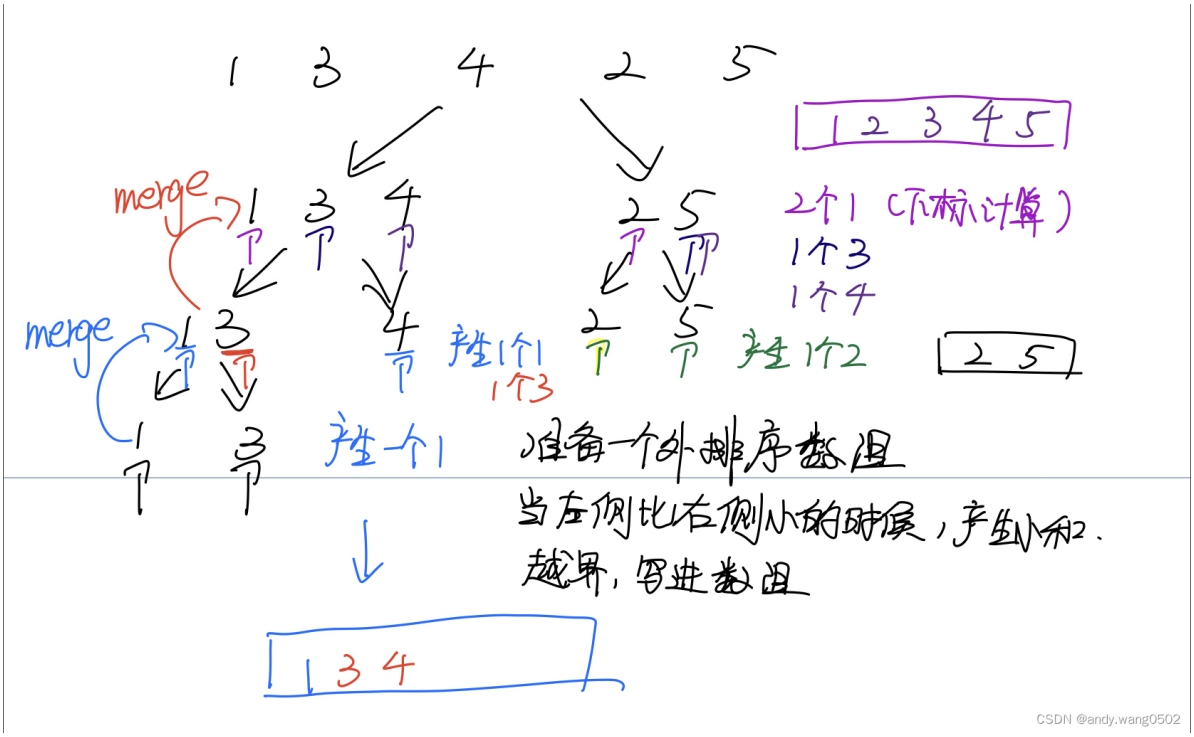
选择排序、插入排序和冒泡排序的时间复杂度都是O(N^2),在每次比较后只搞定了一个数，比较是独立的，浪费了大量的比较行为。归并排序没有浪费比较行为，比较行为成为一个整体有序的部分，下一步生成更长的整体有序的部分。

# 小和问题

在一个数组中，每一个数左边比当前数小的数累加起来，叫做这个数组的小和。求一个数组的小和

暴力解法：遍历，复杂度O(N^2)





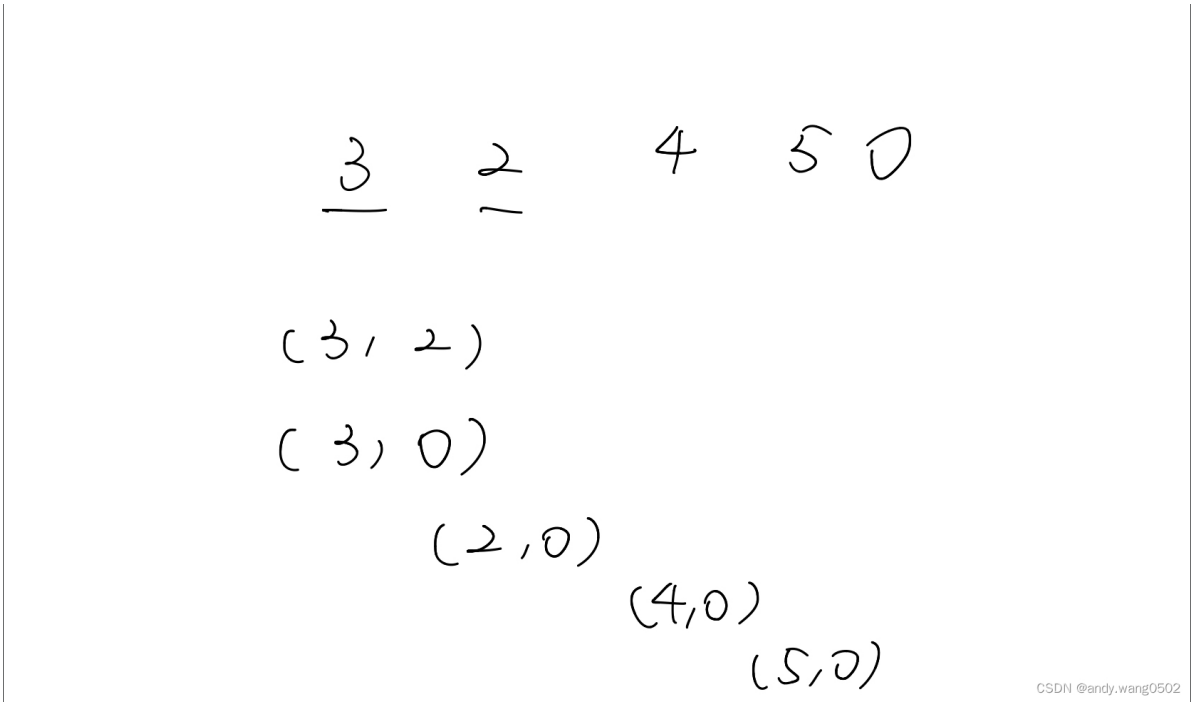

代码：

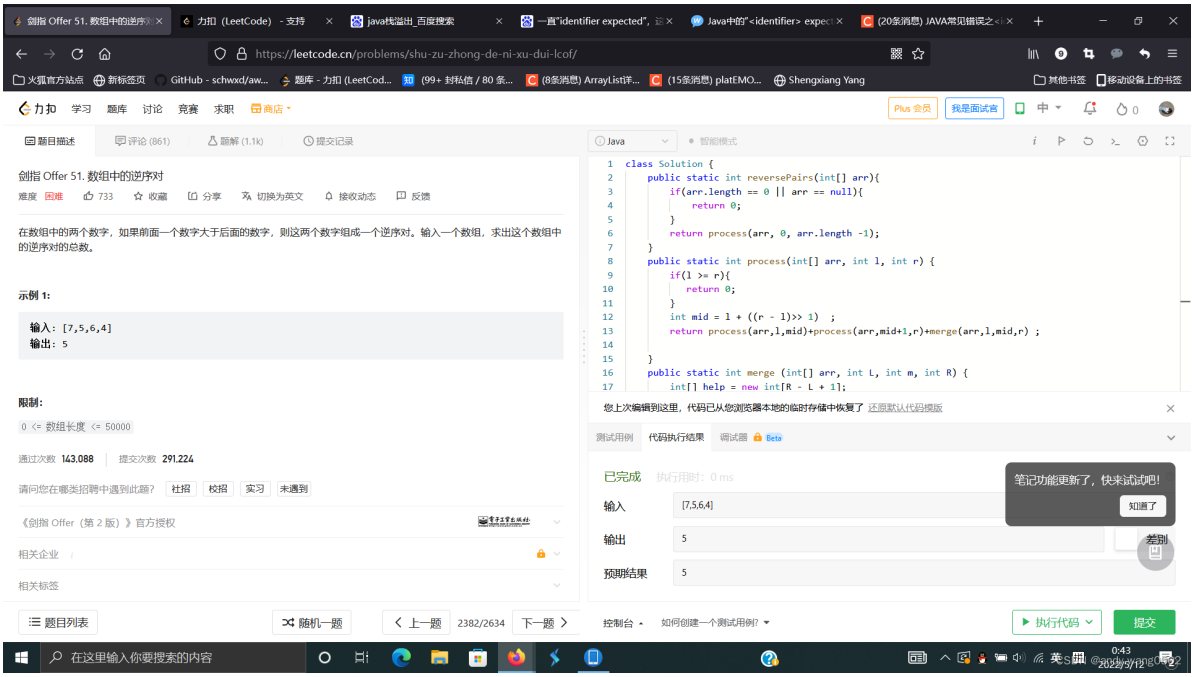
```
public static int smallSum(int[] arr) {
    if(arr == null || arr.length < 2){
        return 0;
    }
    return prcess(arr, 0, arr.length - 1);

}

// arr[L..R]既要做好排序，也要做好小和
public static int process(int[] arr, int l, int r){
    if (l == r){
        return 0;
    }
    int mid = l + (r-l) >> 1;
    return process(arr, l, mid)
            + process(arr, mid + 1, r)
            + merge(arr, l , mid, r);//三块的小和相加才是数组的小和
}

public static int merge(int[] arr, int L, int m, int r){
    int[] help = new help[r - L + 1];
    int i = 0;
    int p1 = L;
    int p2 = m+1;
    int res = 0;
    while (p1 <=m && p2 <=r){
        res += arr[p1] < arr[p2] ? (r - p2 + 1)* arr[p1] : 0;//通过下标一次性计算，节省复杂度
        help[i++] = arr[p1] < arr[p2] ? arr[p1++] : arr[p2++]; //左组数与右组数相等时先拷贝右组数，这样可以知道右边有多少个数比左组数大
    }
    while(p1 <= m){
        help[i++] = arr[p1++];
    }
```

```
    while(p2 <=r){
        help[i++] = arr[p2++];
    }
    for(i = 0; i < help.length; i++){
        arr[L + i] = help[i];
    }
    return res;
}
```

# 逆序对（剑指offer 51）

在一个数组中，左边的数如果比右边大，则这两个数构成一个逆序对，请打印所有的逆序对。




```
var reversePairs = function (nums) {
    if (nums.length < 1 || nums === null) return 0
    return process(nums, 0, nums.length - 1 )
};
```

```
function process(nums, left, right) {
    let mid = left + ((right - left) >> 1)
    if (left == right) {
        return 0
    }
    return process(nums, left, mid) + process(nums, mid + 1, right) +
merge(nums, left, mid, right)
}

function merge(nums, left, middle, right) {
    let help = Array.from({ length: right - left + 1 })
    let l1 = left
    let l2 = middle + 1
    let i = 0
    let res = 0
    while (l1 <= middle && l2 <= right) {
        res += nums[l1] <= nums[l2] ? 0 : middle - l1 + 1
        help[i++] = nums[l1] <= nums[l2] ? nums[l1++] : nums[l2++]
    }
    while (l1 <= middle) {
        help[i++] = nums[l1++]
    }
    while (l2 <= right) {
        help[i++] = nums[l2++]
    }
    for (let i = 0; i <= help.length - 1; i++) {
        nums[left+i] = help[i]
    }
    return res
}
```

# 荷兰国旗问题

问题1：给定一个数组arr，和一个数`num`，请把小于等于`num`的数放在数组的左边，大于`num`的数放在数组的右边，要求空间复杂度O(1)，时间复杂度O(N)

分析：把一个数组分成俩块，左边小于等于`num`，右边大于`num`，不一定有序。

1. `arr[i]<= num`,把 arr[i]和小于等于区的下一个数做交换，然后小于等于区往右扩一个位置，i++
2. `arr[i]> num，i++`

i越界，停止

```
function NetherlandsFlag(arr, pivot) {
    let i = 0;
    let left = -1;
    while (i < arr.length) {
        if (arr[i] <= pivot) {
            swap(arr, ++left, i++);
        } else {
            i++;
        }
    }
    return arr;
}
```

```
function swap(arr, i, j) {
    let temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

问题2：给定一个数组arr，和一个数`num`，请把小于`num`的数放在数组的左边，等于`num`的数放在数组的中间，大于`num`的数放在数组的右边，要求空间复杂度O(1)，时间复杂度O(N)

分析：采用双指针，分成三块，大于区，等于区，小于区

1. `[i]<num`, [i]和<区下一个做交换，<区右扩，i++
2. `[i]==num, i++`
3. `[i]>num`,[i]和>区前一个做交换，>区左扩大，i原地不变。

```
function NetherlandsFlag2(arr, pivot) {
    let i = 0;
    let left = -1;
    let right = arr.length;

    while (i < right) { // 小于right
        if (arr[i] < pivot) {
            swap(arr, ++left, i++);
        } else if (arr[i] > pivot) {
            swap(arr, --right, i);
        } else {
            i++;
        }
    }
    return arr;
}

function swap(arr, i, j) {
    let temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

# 快排

快排1.0 O(N^2)

拿最后一个数做`num`，将前面的一段（不包括`num`）分为小于等于区在左边，大于区在右边，然后将`num`与大于区的第一个数做交换。接着让左侧和右侧重塑这个行为，递归

```
```

快排2.0 O(N^2)

利用荷兰国旗问题，拿最后一个数做`num`，前面的区域分为小于`num`的在左边，之间是等于`num`的一段，右边是大于`num`的，然后将`num`与大于`num`的第一个区域做交换，接着小于`num`的和大于`num`的递归。

相比1.0，搞定了一批等于`num`，稍快一些

快排3.0 `O(N*logN)`

在数组的排序时候，选择`num`做划分时随机选择某位置上的数，不局限于最后一个位置

随机选择划分值，好情况和坏情况是概率事件，有1/N的权重，所以时间复杂度为`O(N*logN)`

```
public static void quickSort(int[] arr, int L, int R) {
    if (L < R){
       swap(arr, L + (int)(Math.random() * (R - L + 1)), R); //将最后位置的数与随机位置交换
        int[] p = partition(arr, L, R); // 最后位置为荷兰期问题中要比较的数
        quickSort(arr, L, p[0] - 1); //< 区
        quickSort(arr, p[1] + 1, R);//>区
    }
}

public static int[] partition(int[] arr, int l, int r) {
    int less = l - 1;
    int more = r;
    while (l < more) {
        if (arr[l] < arr[r]) {
            swap(arr, ++less, l++);
        } else if (arr[l] > arr[r]) {
            swap(arr, --more, l);
        } else {
            l++;
        }
    }
    swap(arr, more, r);
    return new int[] { less + 1, more };
}
```

```
function qucikSort(arr, L, R) {
    if (L < R) {
        swap(arr, Math.floor((R - L) * Math.random() + L), R);
        let [left, right] = partition(arr, L, R);
        qucikSort(arr, L, left - 1);
        qucikSort(arr, right + 1, R);
    }
    return arr;
}

function partition(arr, L, R) {
    let left = L - 1;
    let right = R + 1;
    let i = L;
```

```
    let pivot = arr[R]

    while (i < right) {
        if (arr[i] < pivot) {
            swap(arr, ++left, i++);
        } else if (arr[i] > pivot) {
            swap(arr, --right, i);
        } else {
            i++;
        }
    }
    return [left + 1, right - 1];
}
```