1排序

概念：将一组杂乱无章的数，按照一定的顺序排列起来。

分类：内部排序和外部排序

2十大排序

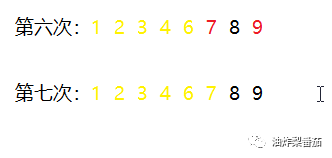
01

选择排序

基本思想：假定把一组需要排序的数放在数组中。第一趟从arry【0】开始，经过比较，到arry【n-1】结束，选出最小的数，并与arry【0】交换位置；第二趟从arry【1】开始，经过比较，到arry【n-1】结束，选出最小的数，并与arry【1】交换位置；以此类推，经过n-1趟的比较

如下图所示：





上图中黄色为已经排好序，红色为交换位置的两个数。

代码实现：

*/\*\* \* 选择排序 \*/* public static void selectSort(){ int[] arr={4,3,6,9,8,1,2,7}; for (int i = 0; i < arr.length-1; i++) {​ int indexNum=i; for (int j = i+1; j < arr.length; j++) { if (arr[indexNum] > arr[j]) { indexNum=j; } } int temp=arr[indexNum]; arr[indexNum]=arr[i]; arr[i]=temp;​ System.out.print("第"+""+(i+1)+"次："); for (int m = 0; m < arr.length; m++) { System.out.print(arr[m]+" "); } System.out.println(); } }

运行结果：

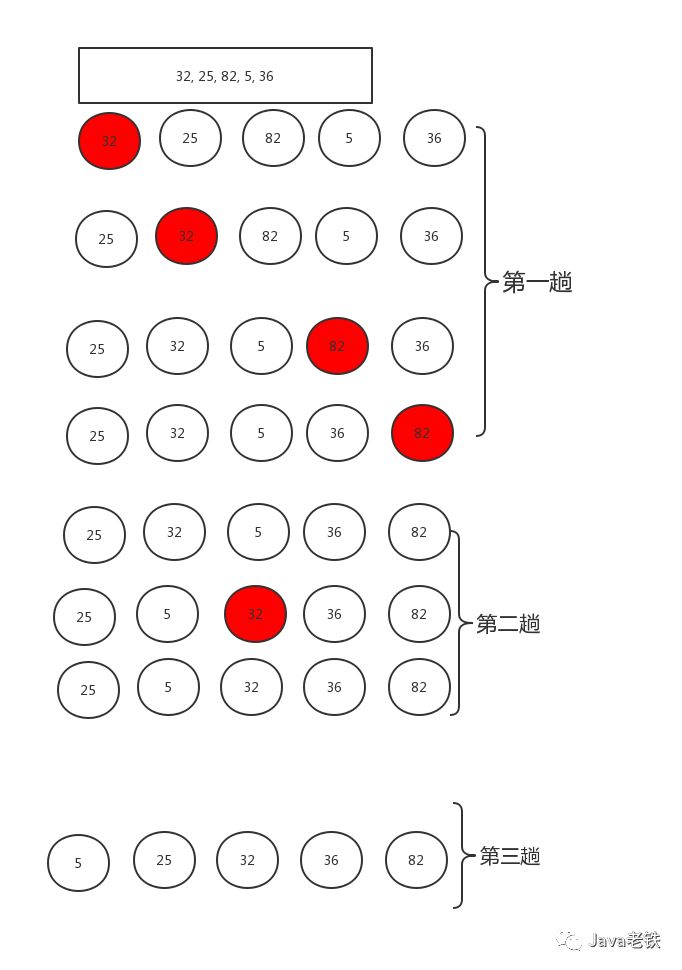
第1次：1 3 6 9 8 4 2 7 第2次：1 2 6 9 8 4 3 7 第3次：1 2 3 9 8 4 6 7 第4次：1 2 3 4 8 9 6 7 第5次：1 2 3 4 6 9 8 7 第6次：1 2 3 4 6 7 8 9 第7次：1 2 3 4 6 7 8 9

思考：从大到小时如何修改上述代码？

02

冒泡排序

基本思路：从头开始扫描待排序的元素，在扫描过程中依次对相邻的元素进行比较，将元素大的后移，就像水底下的气泡一样，逐渐向上冒。



说明：本来需要四趟，但是第三趟排完之后，发现顺序已经排好，则排序提前结束。

代码实现：

*/\*\* \* 冒泡排序 \* @param arry \*/* public static void bubbleSort(int[] arry){​ *//只有一个数 直接退出*

if (arry.length <= 1){ System.out.println("0个"); return ; } *//外层循环控制排序的趟数*

for (int i = 0; i <arry.length ; i++) {

*// 提前退出冒泡循环的标志位,即一次比较中没有交换任何元素，这个数组就已经是有序的了*

boolean flag=false;

*//内层循环控制每一趟需要的次数*

for (int j = 0; j < arry.length-i-1; j++) { if (arry[j]>arry[j+1]){ int num=arry[j]; arry[j]=arry[j+1];*//把大的值交换到后面*

arry[j+1]=num; flag=true; } }

*//没有数据交换 说明已经排好序，直接退出*

if (!flag){ return; } } }

运行结果：

排序前：[32, 25, 82, 5, 36]排序后：[5, 25, 32, 36, 82]

小结：一共需要n-1趟排序，每一趟的排序次数都在减小，如果某一趟的排序中没有一次的交换，说明已经排好序，可以提前结束。

03

插入排序

基本思路：将一组数据分成两个表，假定一张表为有序表，即排好序的表，另一张为无序表，将无序表中的数据插入到有序表中。

原始数据：49 38 65 97 76 13 27

第一趟

        开始前：（49）38 65 97 76 13 27

        结束后：（38 49）65 97 76 13 27

第二趟

        开始前：（38 49）65 97 76 13 27

        结束后：（38 49 65）97 76 13 27

        65和49作比较，65大于49，则65在49后面。

第三趟

开始前：（38 49 65）97 76 13 27

结束后：（38 49 65 97）76 13 27

第四趟

开始前：（38 49 65 97）76 13 27

结束后：（38 49 65 76 97）13 27

第五趟

开始前：（38 49 65 76 97）13 27

结束后：（13 38 49 65 76 97）27

第六趟

开始前：（38 49 65 76 97）13 27

结束后：（13 27 38 49 65 76 97）

说明：第一趟中假定49为排好序的数，即在有序表中，38去和49作比较，38比49小，插入到49左边。后面每一趟依次类推，先和排好序的最大的作比较，括号中的为排好序的数。

代码实现：

*/\*\* \* 插入排序 \* @param arry \*/* public static void insertSort(int[] arry){ for (int i = 1; i < arry.length; i++) { for (int j=i;j>0;j--){ if (arry[j]<arry[j-1]){ int temp=arry[j-1]; arry[j-1]=arry[j]; arry[j]=temp; } } } }

运行结果：

[13, 27, 38, 49, 65, 76, 97]

04

希尔排序

基本思路：希尔排序的本质是分组插入排序。希尔排序也被称为缩小增量排序，先将待排序的记录序列分割成几组，对每一组进行插入排序；然后减少增量，每组的数据将会增多，在进行插入排序；重复上述操作，直到增量为1是，整个数据就会被分在一组，算法就马上终止。

原始数据：8 9 1 7 2 3 5 4 6  0

第一趟增量为5：（8 3）（9 5）（1 4）（7 6）（2 0）

插入排序后：3 5 1 6  0  8 9 4 7 2

第二趟增量为2：（3 1  0  9 7）（5 6 8 4 2）

插入排序后：0  2 1 4 3 5 7 6 9 8

第三趟增量为1：(0  2 1 4 3 5 7 6 9 8)

插入排序后：0  1 2 3 4 5 6 7 8 9

代码实现：

*/\*\* \* 希尔排序 \*/* public static void shellSort(int[] arry){ *//确定增量* for (int gap=arry.length/2;gap > 0;gap /=2) { for (int i = gap; i < arry.length; i++) { for (int j = i - gap; j >= 0; j -= gap) { if (arry[j] > arry[j + gap]) { int temp = arry[j + gap]; arry[j + gap] = arry[j]; arry[j] = temp; } } } } }​

第二种实现方式：

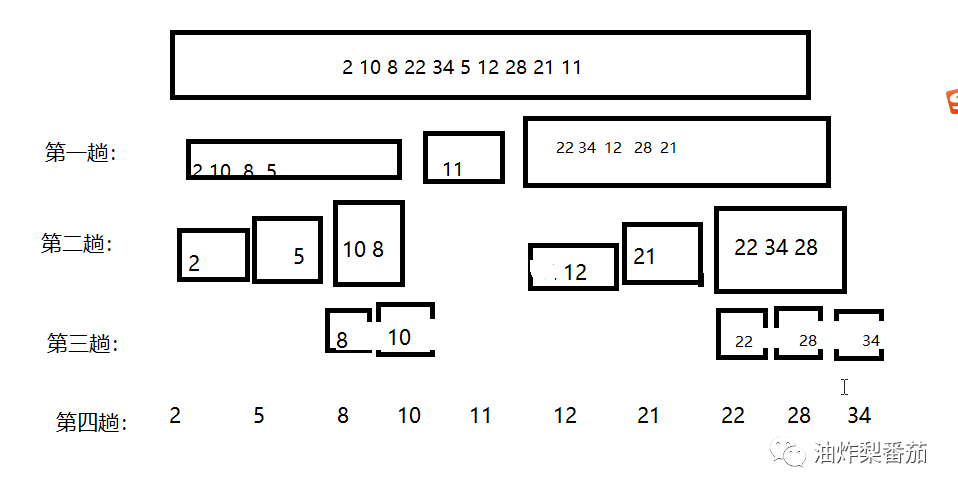
*/\*\* \*希尔排序2 \*/* public static void shellSort2(int[] arry){​ int h=1; while (h <=arry.length/3){ h=h\*3+1; } for (int gap = h; gap > 0; gap=(gap-1)/3) { for (int i = gap; i < arry.length; i++) { for (int j = i; j > gap-1; j -= gap) { if (arry[j] > arry[j - gap]) { int temp = arry[j - gap]; arry[j - gap] = arry[j]; arry[j] = temp; } } } } }​

小结：记录跳跃式移动导致排序方法是不稳定的。

05

快速排序

基本思路：快速排序是对冒泡排序的改进，在待排序的记录中任取一个作为枢轴，经过一趟排序后，把所有小于枢轴的数排在枢轴的前面，大的排在后面，然后对轴的两则重复上述操作。



说明：在第一趟以11为枢轴，比11小的放在左边，比11大的放在右边。

第二趟对11两侧的数进行排序，左侧以5为枢轴，小的放在左边，大的放在右边，右侧以21为枢轴小的放在左边，大的放在右边。第三趟步骤一样，最终得到结果。

代码实现;

public static void sort(int[] arry,int left,int right){ if (right - left <=0){ return; } int mid = quickSort(arry, left, right); sort(arry,left,mid - 1); sort(arry, mid+1,right); }

*/\*\**

*\* 快速排序*

*\* @param arry*

*\* @param left 待排序列起始索引*

*\* @param right 待排序列结束索引*

*\*/* public static int quickSort(int[] arry, int left, int right){​ int l=left;*//左边边界下标*

int r=right-1;*//右边边界下标*

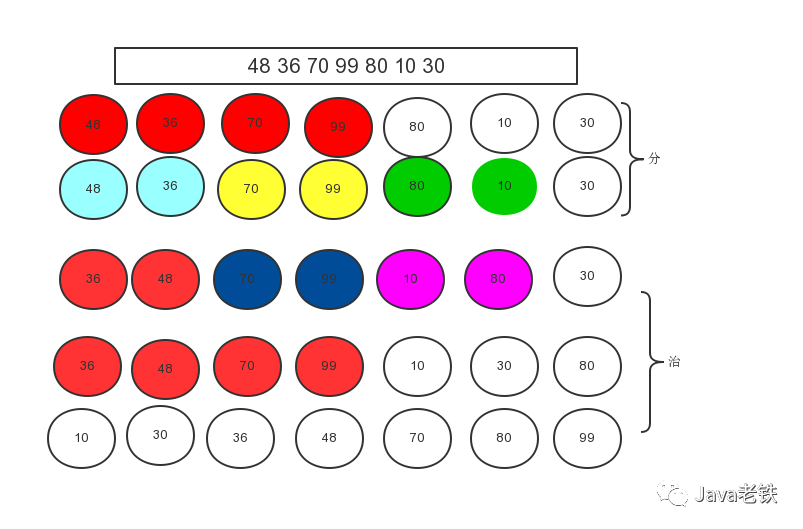
int pivot=arry[right];*//枢轴*​

while (l <= r){ while (l<=r && arry[l] <= pivot){ l++; } while(l<=r && arry[r] > pivot){ r--; } if (l < r){ int temp = arry[l]; arry[l] = arry[r]; arry[r] = temp; } } int temp = arry[l]; arry[l] = arry[right]; arry[right] = temp;​ return l; }

06

归并排序

基本思路：就是讲两个或两个以上的有序表合并成一个有序表的过程。



说明：第一行为原始序列，第二行将原始数据分为4部分，第三行对每一部分进行排序，第四行进行合并，合并之后的两部分分别为一个有序序列，然后对这两个有序序列在合并。

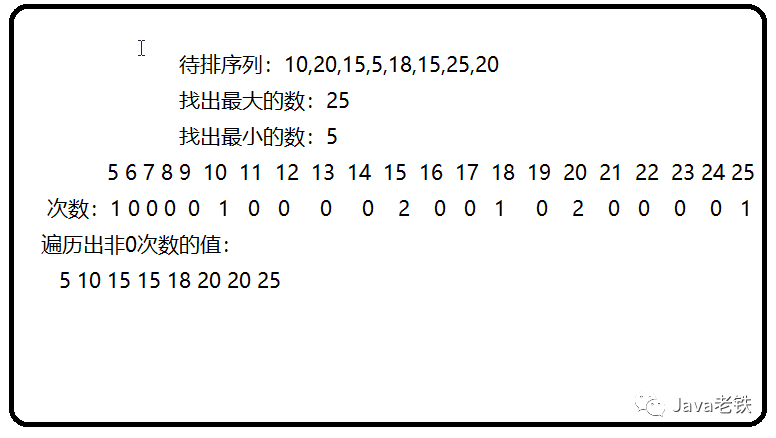
代码实现：

public static void main(String[] args) { int[] arr={48, 36, 70, 99, 80, 10 ,30}; *//归并排序需要一个额外空间* int temp[] = new int[arr.length]; mergeSort(arr, 0, arr.length - 1, temp);​ System.out.println(Arrays.toString(arr)); }​*//分+合方法* public static void mergeSort(int[] arr, int left, int right, int[] temp) { if(left < right) { int mid = (left + right) / 2; *//中间索引* *//向左递归进行分解* mergeSort(arr, left, mid, temp); *//向右递归进行分解* mergeSort(arr, mid + 1, right, temp); *//合并* merge(arr, left, mid, right, temp); } }​ *//合并的方法* */\*\* \* \* @param arr 排序的原始数组 \* @param left 左边有序序列的初始索引 \* @param mid 中间索引 \* @param right 右边索引 \* @param temp 做中转的数组 \*/* public static void merge(int[] arr, int left, int mid, int right, int[] temp) {​ int i = left; *// 初始化i, 左边有序序列的初始索引* int j = mid + 1; *//初始化j, 右边有序序列的初始索引* int t = 0; *// 指向temp数组的当前索引*​ *//先把左右两边(有序)的数据按照规则填充到temp数组* *//直到左右两边的有序序列，有一边处理完毕为止* while (i <= mid && j <= right) {*//继续* *//如果左边的有序序列的当前元素，小于等于右边有序序列的当前元素* *//即将左边的当前元素，填充到 temp数组* *//然后 t++, i++* if(arr[i] <= arr[j]) { temp[t] = arr[i]; t += 1; i += 1; } else { *//反之,将右边有序序列的当前元素，填充到temp数组* temp[t] = arr[j]; t += 1; j += 1; } }​ *//把有剩余数据的一边的数据依次全部填充到temp* while( i <= mid) { *//左边的有序序列还有剩余的元素，就全部填充到temp* temp[t] = arr[i]; t += 1; i += 1; }​ while( j <= right) { *//右边的有序序列还有剩余的元素，就全部填充到temp* temp[t] = arr[j]; t += 1; j += 1; }​ *//将temp数组的元素拷贝到arr* *//注意，并不是每次都拷贝所有* t = 0; int tempLeft = left; *//* *//第一次合并 tempLeft = 0 , right = 1 // tempLeft = 2 right = 3 // tL=0 ri=3* *//最后一次 tempLeft = 0 right = 7* while(tempLeft <= right) { arr[tempLeft] = temp[t]; t += 1; tempLeft += 1; }​ }​

07

计数排序

基本思路：找出待排数组中的最大和最小值，

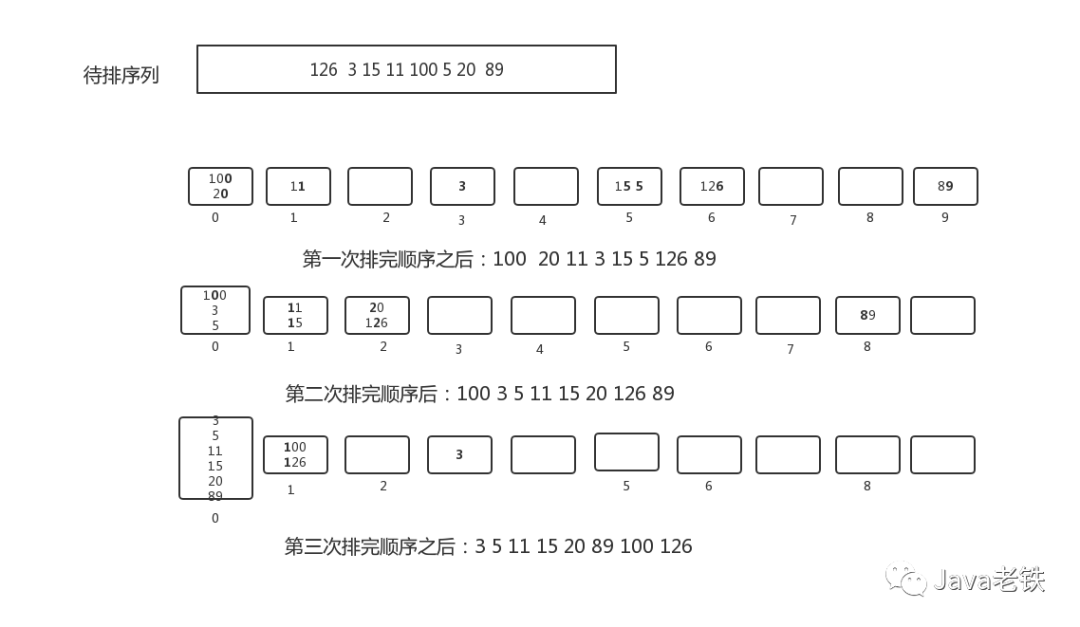


*/\*\* \* 找出最大的数 \* @param arr \* @return 最大值 \*/* private static int getMax(int[] arr){ int max=arr[0]; for (int i = 1; i < arr.length; i++) { if (max<arr[i]){ max=arr[i]; } } return max; }​​ */\*\* \* 找出最小值 \* @param arr \* @return 最小值 \*/* private static int getMin(int[] arr){ int min=arr[0]; for (int i = 1; i < arr.length; i++) { if (min>arr[i]){ min=arr[i]; } } return min; }​​ */\*\* \* 计数排序 \* @param arr \*/* public static void countSort(int[] arr){ int max = getMax(arr); int min = getMin(arr); *//该数组用来存取次数* int[] countArr=new int[max - min +1]; for (int i = 0; i < arr.length; i++) { countArr[arr[i]-min]++; }​ int index=0; for(int i=0;i<countArr.length;) { if(countArr[i]!=0) { arr[index++]=i+min; countArr[i]--; continue; }else { i++; } }​ *//遍历排好序的数组* for (int i = 0; i < arr.length; i++) { System.out.println(i); *// System.out.println(arr[i]);* } }

小结：适合数据量较小的序列  
08

基数排序

思路：先找出最大数的位数，序列中其他的不够该位数的在前面用0补齐，然后从最低位依次排序，最后就会得到一个有序的序列。



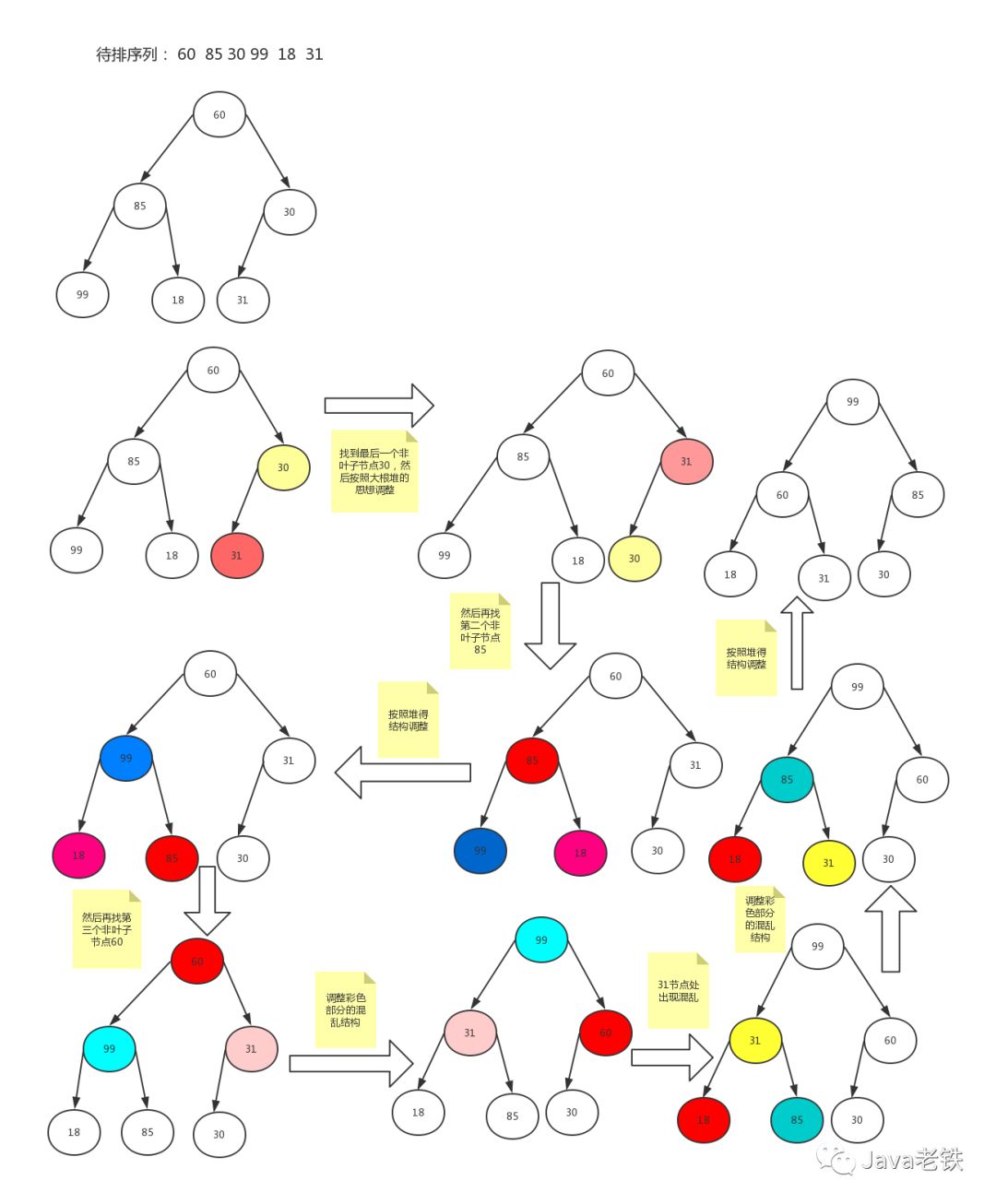
说明：先对待排序列的个位数进行排序，从个位数为0开始，依次排出100和20，个位数为1的11，个位数为3的3，个位数为5的15和5，个位数为6的126，个位数为9的89；之后在对第一次排完之后的序列进行十位数的排序，按照规律在对最后一位进行排序。

*/\*\* \* 找出最大的数 \* @param arr \* @return 最大值 \*/* private static int getMax(int[] arr){ int max=arr[0]; for (int i = 1; i < arr.length; i++) { if (max<arr[i]){ max=arr[i]; } } return max; } */\*\* \* 基数排序 \* @param arr 待排序列 \*/* public static void baseSort(int arr[]) { int max = getMax(arr); String s = Integer.toString(max); int length = s.length();​ *//二维数组 表示10个桶* int[][] bucket = new int[10][arr.length]; *//记录每一个桶中存放的数据* int[] perBucket = new int[10];​ *//循环每一位上的数，i=0为个位数 i=1为十位数* for (int i = 0,n=1; i < length; n \*= 10,i++) { *//循环待排序列* for (int j = 0; j < arr.length; j++) { *//取出每一位上的元素* int perElement = arr[j] / n % 10; *//数据放到对应的桶中* bucket[perElement][perBucket[perElement]] = arr[j]; perBucket[perElement]++; }​​ int index = 0; *//遍历每一个桶中的数据 然后放回到原来的数组中* for (int k = 0; k < perBucket.length; k++) { if (perBucket[k] != 0) { for (int m = 0; m < perBucket[k]; m++) { *//取出元素放入到arr* arr[index++] = bucket[k][m]; } } perBucket[k] = 0; } System.out.println("第"+(i+1)+"次："+Arrays.toString(arr)); } }

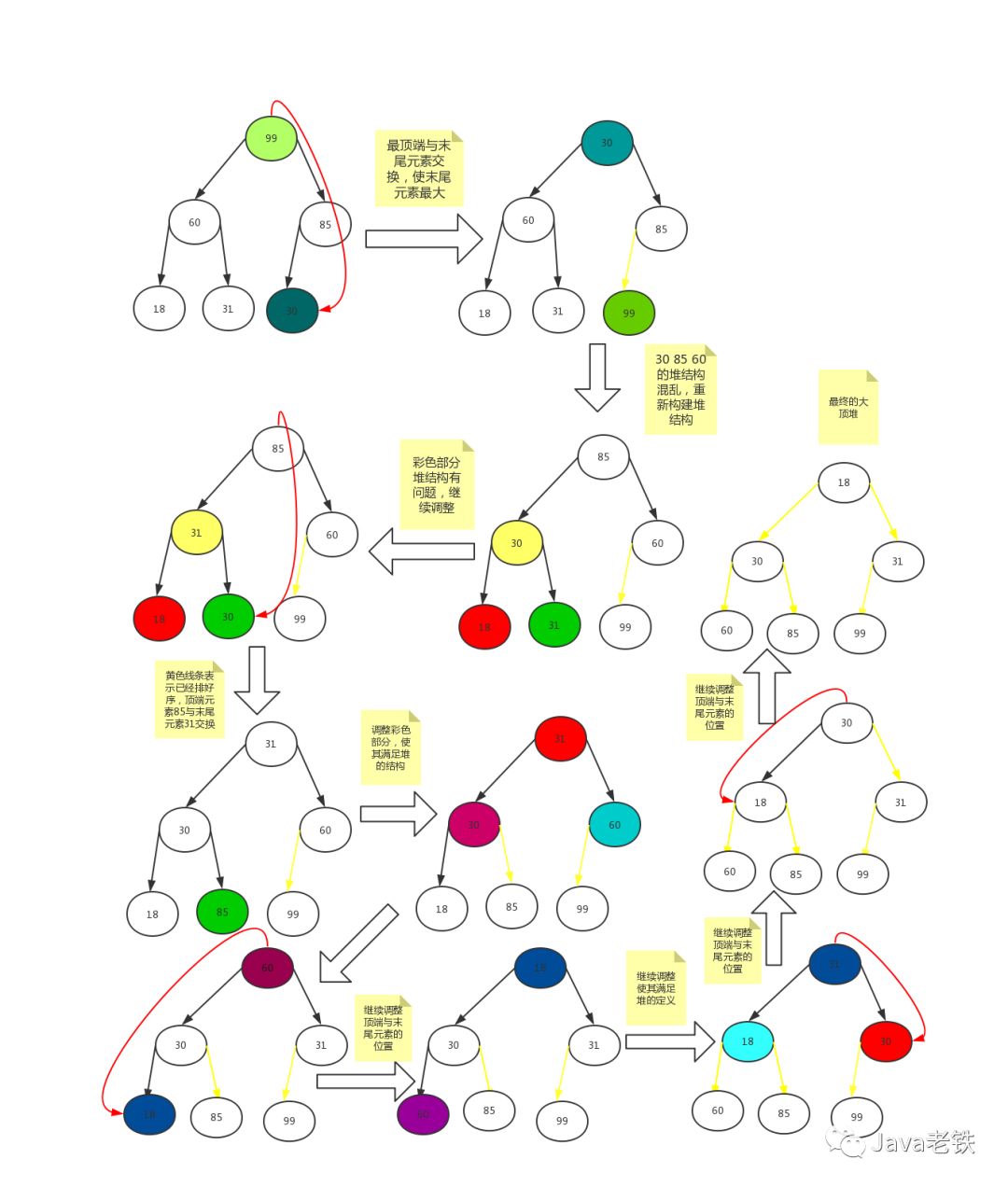
10

堆排序

基本思路：先将待排序列构建成一个堆，再根据需求选择大顶堆还是小顶堆（升序采用大根堆，降序为小根堆），接着，最顶端与末尾元素交换，重复此操作，直到排出顺序。  
图文详解：第一步：构建堆



第二步:最顶端与末尾元素交换，重复此操作，直到排出顺序。



代码实现：

*/\*\* \* 堆排序 \* @param arr \*/* public static void headSort(int[] arr){ int temp=0; *//将待排的数组构建为一个堆* for (int i =arr.length/2 -1; i>=0;i--) { adjustHeap(arr,i,arr.length); }​ *//交换顶端与末尾元素的值 然后根据大小 调整堆的结构* for (int j = arr.length - 1; j > 0; j--) { temp=arr[j]; arr[j]=arr[0]; arr[0]=temp; adjustHeap(arr,0,j); } System.out.println(Arrays.toString(arr)); } */\*\* \* 构建堆 \* @param arr 待调整的数组 \* @param i 非叶子节点在数组中的索引 \* @param length 需要调整的元素的数量 \*/* public static void adjustHeap(int arr[],int i,int length){ int temp=arr[i]; for (int k = i\*2+1; k < length; k=k\*2+1) { *//左子节点的值小于右子节点的值* if (k+1 < length && arr[k] < arr[k+1]){ k++; } *//如果子节点大于父节点* if (arr[k] > temp ){ arr[i]=arr[k]; i=k; }else{ break; } } arr[i]=temp; }

欢迎识别下方二维码，关注小编微信公众号，可以获取跟多Java资料：

​