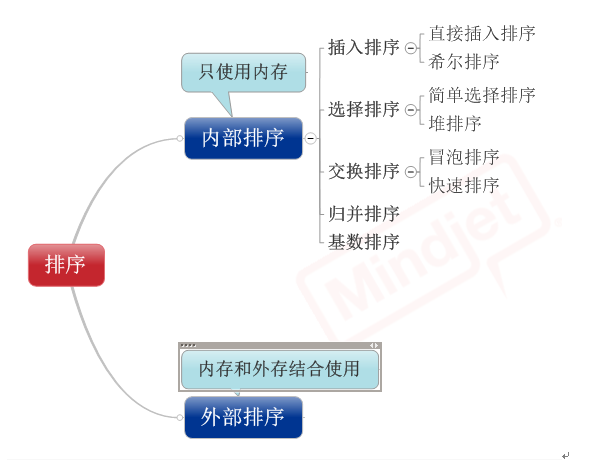
[----JAVA资料免费领取：加QQ群：685430616]

8种排序之间的关系:



**1， 直接插入排序**

   （1）基本思想：在要排序的一组数中，假设前面(n-1)[n>=2] 个数已经是排

好顺序的，现在要把第n个数插到前面的有序数中，使得这n个数

也是排好顺序的。如此反复循环，直到全部排好顺序。

 （2）实例



（3）用java实现

|  |  |  |
| --- | --- | --- |
| 01 | package com.njue; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class insertSort { | |
| 04 | public insertSort(){ |

|  |  |  |  |
| --- | --- | --- | --- |
| 05 | inta[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51}; | | |
| 06 | | int temp=0; |

|  |  |  |
| --- | --- | --- |
| 07 | for(int i=1;i<a.length;i++){ | |
| 08 | int j=i-1; |

|  |  |
| --- | --- |
| 09 | temp=a[i]; |
| 10 | for(;j>=0&&temp<a[j];j--){ | |

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | a[j+1]=a[j];                       //将大于temp的值整体后移一个单位 | | |
| 12 | | } |

|  |  |  |
| --- | --- | --- |
| 13 | a[j+1]=temp; | |
| 14 | } |

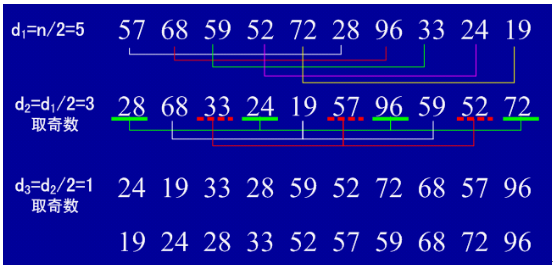
|  |  |
| --- | --- |
| 15 | for(int i=0;i<a.length;i++) |
| 16 | System.out.println(a[i]); | |

|  |  |
| --- | --- |
| 17 | } |
| 18 | } |

**2，           希尔排序（最小增量排序）**

（1）基本思想：算法先将要排序的一组数按某个增量d（n/2,n为要排序数的个数）分成若干组，每组中记录的下标相差d.对每组中全部元素进行直接插入排序，然后再用一个较小的增量（d/2）对它进行分组，在每组中再进行直接插入排序。当增量减到1时，进行直接插入排序后，排序完成。

（2）实例：

****

**（3）用java实现**

|  |  |  |
| --- | --- | --- |
| 01 | public class shellSort { | |
| 02 | public  shellSort(){ |

|  |  |  |
| --- | --- | --- |
| 03 | int a[]={1,54,6,3,78,34,12,45,56,100}; | |
| 04 | double d1=a.length; |

|  |  |
| --- | --- |
| 05 | int temp=0; |
| 06 | while(true){ |

|  |  |  |
| --- | --- | --- |
| 07 | d1= Math.ceil(d1/2); | |
| 08 | int d=(int) d1; |

|  |  |
| --- | --- |
| 09 | for(int x=0;x<d;x++){ |
| 10 | for(int i=x+d;i<a.length;i+=d){ | |

|  |  |  |
| --- | --- | --- |
| 11 | int j=i-d; | |
| 12 | temp=a[i]; |

|  |  |  |
| --- | --- | --- |
| 13 | for(;j>=0&&temp<a[j];j-=d){ | |
| 14 | a[j+d]=a[j]; |

|  |  |
| --- | --- |
| 15 | } |
| 16 | a[j+d]=temp; | |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 | } |

|  |  |
| --- | --- |
| 19 | if(d==1) |
| 20 | break; | |

|  |  |
| --- | --- |
| 21 | } |
| 22 | for(int i=0;i<a.length;i++) | |

|  |  |  |
| --- | --- | --- |
| 23 | System.out.println(a[i]); | |
| 24 | } |

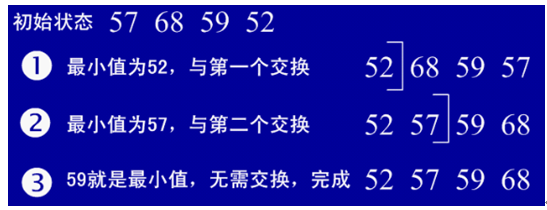
|  |  |
| --- | --- |
| 25 | } |

**3.简单选择排序**

（1）基本思想：在要排序的一组数中，选出最小的一个数与第一个位置的数交换；

然后在剩下的数当中再找最小的与第二个位置的数交换，如此循环到倒数第二个数和最后一个数比较为止。

（2）实例：



（3）用java实现

|  |  |
| --- | --- |
| 01 | public class selectSort { |
| 02 | public selectSort(){ | |

|  |  |  |
| --- | --- | --- |
| 03 | int a[]={1,54,6,3,78,34,12,45}; | |
| 04 | int position=0; |

|  |  |  |
| --- | --- | --- |
| 05 | for(int i=0;i<a.length;i++){ | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | int j=i+1; |
| 08 | position=i; |

|  |  |
| --- | --- |
| 09 | int temp=a[i]; |
| 10 | for(;j<a.length;j++){ | |

|  |  |
| --- | --- |
| 11 | if(a[j]<temp){ |
| 12 | temp=a[j]; | |

|  |  |  |
| --- | --- | --- |
| 13 | position=j; | |
| 14 | } |

|  |  |
| --- | --- |
| 15 | } |
| 16 | a[position]=a[i]; | |

|  |  |  |
| --- | --- | --- |
| 17 | a[i]=temp; | |
| 18 | } |

|  |  |
| --- | --- |
| 19 | for(int i=0;i<a.length;i++) |
| 20 | System.out.println(a[i]); | |

|  |  |  |
| --- | --- | --- |
| 21 | } | |
| 22 | } |

**4，      堆排序**

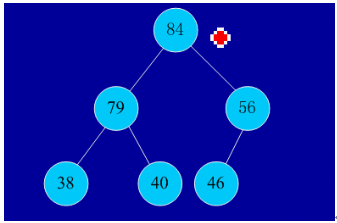
（1）基本思想：堆排序是一种树形选择排序，是对直接选择排序的有效改进。

堆的定义如下：具有n个元素的序列（h1,h2,...,hn),当且仅当满足（hi>=h2i,hi>=2i+1）或（hi<=h2i,hi<=2i+1）(i=1,2,...,n/2)时称之为堆。在这里只讨论满足前者条件的堆。由堆的定义可以看出，堆顶元素（即第一个元素）必为最大项（大顶堆）。完全二叉树可以很直观地表示堆的结构。堆顶为根，其它为左子树、右子树。初始时把要排序的数的序列看作是一棵顺序存储的二叉树，调整它们的存储序，使之成为一个堆，这时堆的根节点的数最大。然后将根节点与堆的最后一个节点交换。然后对前面(n-1)个数重新调整使之成为堆。依此类推，直到只有两个节点的堆，并对它们作交换，最后得到有n个节点的有序序列。从算法描述来看，堆排序需要两个过程，一是建立堆，二是堆顶与堆的最后一个元素交换位置。所以堆排序有两个函数组成。一是建堆的渗透函数，二是反复调用渗透函数实现排序的函数。

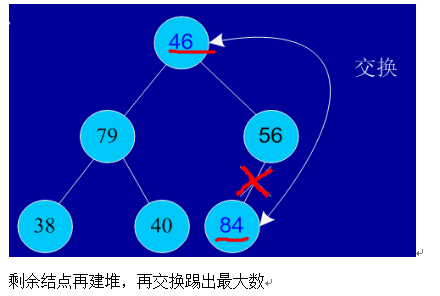
（2）实例：

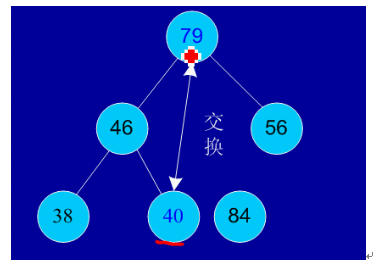
初始序列：46,79,56,38,40,84

建堆：



交换，从堆中踢出最大数





依次类推：最后堆中剩余的最后两个结点交换，踢出一个，排序完成。

（3）用java实现

|  |  |  |
| --- | --- | --- |
| 01 | import java.util.Arrays; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | | public class HeapSort { |
| 04 | int a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51}; | | |

|  |  |  |
| --- | --- | --- |
| 05 | public  HeapSort(){ | |
| 06 | heapSort(a); |

|  |  |
| --- | --- |
| 07 | } |
| 08 | public  void heapSort(int[] a){ | |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.println("开始排序"); | |
| 10 | int arrayLength=a.length; |

|  |  |
| --- | --- |
| 11 | //循环建堆 |
| 12 | for(int i=0;i<arrayLength-1;i++){ | |

|  |  |  |
| --- | --- | --- |
| 13 | //建堆 | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | buildMaxHeap(a,arrayLength-1-i); |
| 16 | //交换堆顶和最后一个元素 | |

|  |  |
| --- | --- |
| 17 | swap(a,0,arrayLength-1-i); |
| 18 | System.out.println(Arrays.toString(a)); | |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 | } |

|  |  |
| --- | --- |
| 21 |  |
| 22 | private  void swap(int[] data, int i, int j) { | |

|  |  |  |
| --- | --- | --- |
| 23 | // TODO Auto-generated method stub | |
| 24 | int tmp=data[i]; |

|  |  |  |
| --- | --- | --- |
| 25 | data[i]=data[j]; | |
| 26 | data[j]=tmp; |

|  |  |
| --- | --- |
| 27 | } |
| 28 | //对data数组从0到lastIndex建大顶堆 | |

|  |  |  |
| --- | --- | --- |
| 29 | private void buildMaxHeap(int[] data, int lastIndex) { | |
| 30 | // TODO Auto-generated method stub |

|  |  |  |
| --- | --- | --- |
| 31 | //从lastIndex处节点（最后一个节点）的父节点开始 | |
| 32 | for(int i=(lastIndex-1)/2;i>=0;i--){ |

|  |  |  |
| --- | --- | --- |
| 33 | //k保存正在判断的节点 | |
| 34 | int k=i; |

|  |  |  |
| --- | --- | --- |
| 35 | //如果当前k节点的子节点存在 | |
| 36 | while(k\*2+1<=lastIndex){ |

|  |  |  |
| --- | --- | --- |
| 37 | //k节点的左子节点的索引 | |
| 38 | int biggerIndex=2\*k+1; |

|  |  |  |
| --- | --- | --- |
| 39 | //如果biggerIndex小于lastIndex，即biggerIndex+1代表的k节点的右子节点存在 | |
| 40 | if(biggerIndex<lastIndex){ |

|  |  |  |
| --- | --- | --- |
| 41 | | //若果右子节点的值较大 |
| 42 | if(data[biggerIndex]<data[biggerIndex+1]){ | | |

|  |  |  |
| --- | --- | --- |
| 43 | //biggerIndex总是记录较大子节点的索引 | |
| 44 | biggerIndex++; |

|  |  |  |
| --- | --- | --- |
| 45 | } | |
| 46 | } |

|  |  |  |
| --- | --- | --- |
| 47 | //如果k节点的值小于其较大的子节点的值 | |
| 48 | if(data[k]<data[biggerIndex]){ |

|  |  |
| --- | --- |
| 49 | //交换他们 |
| 50 | swap(data,k,biggerIndex); | |

|  |  |  |
| --- | --- | --- |
| 51 | //将biggerIndex赋予k，开始while循环的下一次循环，重新保证k节点的值大于其左右子节点的值 | |
| 52 | k=biggerIndex; |

|  |  |
| --- | --- |
| 53 | }else{ |
| 54 | break; | |

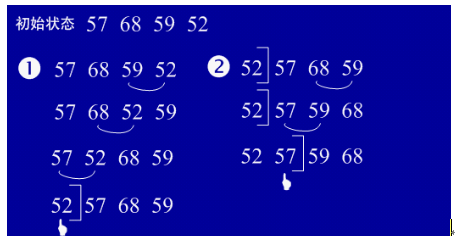
|  |  |
| --- | --- |
| 55 | } |
| 56 | }<p align="left"> <span>   </span>}</p> | |

|  |  |
| --- | --- |
| 57 | <p align="left">    }</p> |
| 58 | <p align="left"> <span style="background-color:white;">}</span></p> | |

**5.冒泡排序**

（1）基本思想：在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。

（2）实例：



（3）用java实现

|  |  |  |
| --- | --- | --- |
| 01 | public class bubbleSort { | |
| 02 | public  bubbleSort(){ |

|  |  |  |  |
| --- | --- | --- | --- |
| 03 | int a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51}; | | |
| 04 | | int temp=0; |

|  |  |
| --- | --- |
| 05 | for(int i=0;i<a.length-1;i++){ |
| 06 | for(int j=0;j<a.length-1-i;j++){ | |

|  |  |
| --- | --- |
| 07 | if(a[j]>a[j+1]){ |
| 08 | temp=a[j]; | |

|  |  |
| --- | --- |
| 09 | a[j]=a[j+1]; |
| 10 | a[j+1]=temp; |

|  |  |
| --- | --- |
| 11 | } |
| 12 | } |

|  |  |
| --- | --- |
| 13 | } |
| 14 | for(int i=0;i<a.length;i++) | |

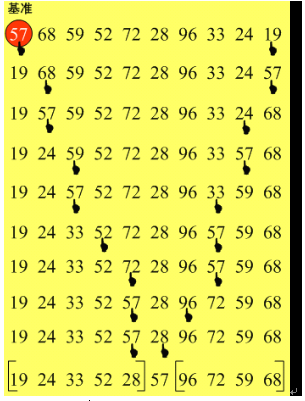
|  |  |  |
| --- | --- | --- |
| 15 | System.out.println(a[i]); | |
| 16 | } |

|  |  |
| --- | --- |
| 17 | } |

**6.快速排序**

（1）基本思想：选择一个基准元素,通常选择第一个元素或者最后一个元素,通过一趟扫描，将待排序列分成两部分,一部分比基准元素小,一部分大于等于基准元素,此时基准元素在其排好序后的正确位置,然后再用同样的方法递归地排序划分的两部分。

（2）实例：



（3）用java实现

|  |  |  |
| --- | --- | --- |
| 01 | | public class quickSort { |
| 02 | int a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51}; | | |

|  |  |  |
| --- | --- | --- |
| 03 | public  quickSort(){ | |
| 04 | quick(a); |

|  |  |
| --- | --- |
| 05 | for(int i=0;i<a.length;i++) |
| 06 | System.out.println(a[i]); | |

|  |  |
| --- | --- |
| 07 | } |
| 08 | public int getMiddle(int[] list, int low, int high) { | |

|  |  |  |
| --- | --- | --- |
| 09 | int tmp = list[low];    //数组的第一个作为中轴 | |
| 10 | while (low < high) { |

|  |  |  |
| --- | --- | --- |
| 11 | while (low < high && list[high] >= tmp) { | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | high--; |
| 14 | } | |

|  |  |
| --- | --- |
| 15 | list[low] = list[high];   //比中轴小的记录移到低端 |
| 16 | while (low < high && list[low] <= tmp) { |

|  |  |  |
| --- | --- | --- |
| 17 | low++; | |
| 18 | } |

|  |  |  |
| --- | --- | --- |
| 19 | list[high] = list[low];   //比中轴大的记录移到高端 | |
| 20 | } |

|  |  |  |
| --- | --- | --- |
| 21 | | list[low] = tmp;              //中轴记录到尾 |
| 22 | return low;                   //返回中轴的位置 | |

|  |  |
| --- | --- |
| 23 | } |
| 24 | public void \_quickSort(int[] list, int low, int high) { | |

|  |  |
| --- | --- |
| 25 | if (low < high) { |
| 26 | int middle = getMiddle(list, low, high);  //将list数组进行一分为二 | |

|  |  |
| --- | --- |
| 27 | \_quickSort(list, low, middle - 1);        //对低字表进行递归排序 |
| 28 | \_quickSort(list, middle + 1, high);       //对高字表进行递归排序 |

|  |  |  |
| --- | --- | --- |
| 29 | } | |
| 30 | } |

|  |  |
| --- | --- |
| 31 | public void quick(int[] a2) { |
| 32 | if (a2.length > 0) {    //查看数组是否为空 | |

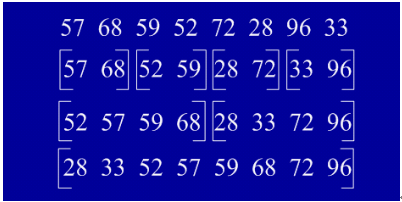
|  |  |  |
| --- | --- | --- |
| 33 | \_quickSort(a2, 0, a2.length - 1); | |
| 34 | } |

|  |  |  |
| --- | --- | --- |
| 35 | } | |
| 36 | } |

**7、归并排序**

（1）基本排序：归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。

（2）实例：

****

**（3）用java实现**

|  |  |  |
| --- | --- | --- |
| 01 | import java.util.Arrays; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | | public class mergingSort { |
| 04 | int a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,34,15,35,25,53,51}; | | |

|  |  |
| --- | --- |
| 05 | public  mergingSort(){ |
| 06 | sort(a,0,a.length-1); | |

|  |  |
| --- | --- |
| 07 | for(int i=0;i<a.length;i++) |
| 08 | System.out.println(a[i]); | |

|  |  |
| --- | --- |
| 09 | } |
| 10 | public void sort(int[] data, int left, int right) { | |

|  |  |  |
| --- | --- | --- |
| 11 | // TODO Auto-generated method stub | |
| 12 | if(left<right){ |

|  |  |
| --- | --- |
| 13 | //找出中间索引 |
| 14 | int center=(left+right)/2; | |

|  |  |
| --- | --- |
| 15 | //对左边数组进行递归 |
| 16 | sort(data,left,center); | |

|  |  |
| --- | --- |
| 17 | //对右边数组进行递归 |
| 18 | sort(data,center+1,right); | |

|  |  |
| --- | --- |
| 19 | //合并 |
| 20 | merge(data,left,center,right); | |

|  |  |  |
| --- | --- | --- |
| 21 |  | |
| 22 | } |

|  |  |
| --- | --- |
| 23 | } |
| 24 | public void merge(int[] data, int left, int center, int right) { | |

|  |  |
| --- | --- |
| 25 | // TODO Auto-generated method stub |
| 26 | int [] tmpArr=new int[data.length]; | |

|  |  |
| --- | --- |
| 27 | int mid=center+1; |
| 28 | //third记录中间数组的索引 | |

|  |  |  |
| --- | --- | --- |
| 29 | int third=left; | |
| 30 | int tmp=left; |

|  |  |  |
| --- | --- | --- |
| 31 | while(left<=center&&mid<=right){ | |
| 32 |  |

|  |  |
| --- | --- |
| 33 | //从两个数组中取出最小的放入中间数组 |
| 34 | if(data[left]<=data[mid]){ |

|  |  |  |
| --- | --- | --- |
| 35 | tmpArr[third++]=data[left++]; | |
| 36 | }else{ |

|  |  |  |
| --- | --- | --- |
| 37 | tmpArr[third++]=data[mid++]; | |
| 38 | } |

|  |  |
| --- | --- |
| 39 | } |
| 40 | //剩余部分依次放入中间数组 | |

|  |  |
| --- | --- |
| 41 | while(mid<=right){ |
| 42 | tmpArr[third++]=data[mid++]; | |

|  |  |
| --- | --- |
| 43 | } |
| 44 | while(left<=center){ | |

|  |  |  |
| --- | --- | --- |
| 45 | tmpArr[third++]=data[left++]; | |
| 46 | } |

|  |  |  |
| --- | --- | --- |
| 47 | //将中间数组中的内容复制回原数组 | |
| 48 | while(tmp<=right){ |

|  |  |  |
| --- | --- | --- |
| 49 | data[tmp]=tmpArr[tmp++]; | |
| 50 | } |

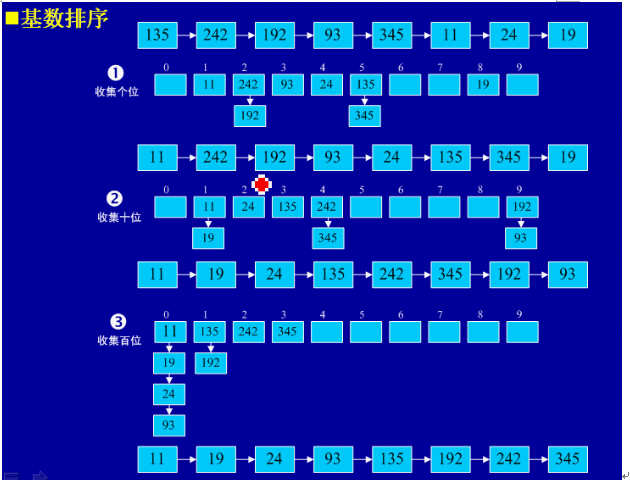
|  |  |  |
| --- | --- | --- |
| 51 | System.out.println(Arrays.toString(data)); | |
| 52 | } |

|  |  |  |
| --- | --- | --- |
| 53 |  | |
| 54 | } |

**8、基数排序**

（1）基本思想：将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后,数列就变成一个有序序列。

（2）实例：

  
（3）用java实现

[查看源码](http://www.open-open.com/lib/view/open1340852024936.html" \l "viewSource" \o "查看源码)

[打印](http://www.open-open.com/lib/view/open1340852024936.html" \l "printSource" \o "打印)[?](http://www.open-open.com/lib/view/open1340852024936.html" \l "about" \o "?)

|  |  |  |
| --- | --- | --- |
| 01 | import java.util.ArrayList; | |
| 02 | import java.util.List; |

|  |  |
| --- | --- |
| 03 |  |
| 04 | public class radixSort { | |

|  |  |  |  |
| --- | --- | --- | --- |
| 05 | int a[]={49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,101,56,17,18,23,34,15,35,25,53,51}; | | |
| 06 | | public radixSort(){ |

|  |  |
| --- | --- |
| 07 | sort(a); |
| 08 | for(int i=0;i<a.length;i++) | |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.println(a[i]); | |
| 10 | } |

|  |  |  |
| --- | --- | --- |
| 11 | public  void sort(int[] array){ | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | //首先确定排序的趟数; | |
| 14 | int max=array[0]; |

|  |  |  |
| --- | --- | --- |
| 15 | for(int i=1;i<array.length;i++){ | |
| 16 | if(array[i]>max){ |

|  |  |  |
| --- | --- | --- |
| 17 | max=array[i]; | |
| 18 | } |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 |  |

|  |  |
| --- | --- |
| 21 | int time=0; |
| 22 | //判断位数; | |

|  |  |
| --- | --- |
| 23 | while(max>0){ |
| 24 | max/=10; | |

|  |  |  |
| --- | --- | --- |
| 25 | time++; | |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |
| 28 | //建立10个队列; | |

|  |  |  |
| --- | --- | --- |
| 29 | List<ArrayList> queue=new ArrayList<ArrayList>(); | |
| 30 | for(int i=0;i<10;i++){ |

|  |  |  |
| --- | --- | --- |
| 31 | ArrayList<Integer> queue1=new ArrayList<Integer>(); | |
| 32 | queue.add(queue1); |

|  |  |
| --- | --- |
| 33 | } |
| 34 |  | |

|  |  |
| --- | --- |
| 35 | //进行time次分配和收集; |
| 36 | for(int i=0;i<time;i++){ | |

|  |  |
| --- | --- |
| 37 |  |
| 38 | //分配数组元素; | |

|  |  |  |
| --- | --- | --- |
| 39 | for(int j=0;j<array.length;j++){ | |
| 40 | //得到数字的第time+1位数; |

|  |  |  |
| --- | --- | --- |
| 41 | int x=array[j]%(int)Math.pow(10, i+1)/(int)Math.pow(10, i); | |
| 42 | | ArrayList<Integer> queue2=queue.get(x); |

|  |  |
| --- | --- |
| 43 | queue2.add(array[j]); |
| 44 | queue.set(x, queue2); |

|  |  |
| --- | --- |
| 45 | } |
| 46 | int count=0;//元素计数器; | |

|  |  |
| --- | --- |
| 47 | //收集队列元素; |
| 48 | for(int k=0;k<10;k++){ | |

|  |  |
| --- | --- |
| 49 | while(queue.get(k).size()>0){ |
| 50 | ArrayList<Integer> queue3=queue.get(k); | |

|  |  |  |  |
| --- | --- | --- | --- |
| 51 | array[count]=queue3.get(0); | | |
| 52 | | queue3.remove(0); |

|  |  |  |
| --- | --- | --- |
| 53 | count++; | |
| 54 | } |

|  |  |  |
| --- | --- | --- |
| 55 | } | |
| 56 | } |

|  |  |  |
| --- | --- | --- |
| 57 |  | |
| 58 | } |

|  |  |  |
| --- | --- | --- |
| 59 |  | |
| 60 | } |