**直接插入排序**

1. **public** **void** insertSort(**int**[] a) {
2. **for**(**int** i=1;i<a.length;i++){//从头部第一个当做已经排好序的，把后面的一个一个的插到已经排好的列表中去。
3. **int** j;
4. **int** x=a[i];//x为待插入元素
5. **for**( j=i;  j>0 && x<a[j-1];j--){//通过循环，逐个后移一位找到要插入的位置。
6. a[j]=a[j-1];
7. }
8. a[j]=x;//插入
9. }
11. }

## 希尔排序

1. **private** **void** ShellInsertSort(**int**[] a, **int** dk) {//类似插入排序，只是插入排序增量是1，这里增量是dk,把1换成dk就可以了
2. **for**(**int** i=dk;i<a.length;i++){
3. **if**(a[i]<a[i-dk]){
4. **int** j;
5. **int** x=a[i];//x为待插入元素
6. a[i]=a[i-dk];
7. **for**(j=i-dk;  j>=0 && x<a[j];j=j-dk){//通过循环，逐个后移一位找到要插入的位置。
8. a[j+dk]=a[j];
9. }
10. a[j+dk]=x;//插入
11. }
13. }
15. }

## 简单选择排序

1. **private** **void** selectSort(**int**[] a) {
2. **for**(**int** i=0;i<a.length;i++){
3. **int** k=i;//k存放最小值下标。每次循环最小值下标+1
4. **for**(**int** j=i+1;j<a.length;j++){//找到最小值下标
5. **if**(a[k]>a[j])
6. k=j;
7. }
8. swap(a,k,i);//把最小值放到它该放的位置上
9. }
10. }
11. **public**  **void** swap(**int**[] data, **int** i, **int** j) {
12. **if** (i == j) {
13. **return**;
14. }
15. data[i] = data[i] + data[j];
16. data[j] = data[i] - data[j];
17. data[i] = data[i] - data[j];
18. }
19. }

## 冒泡排序

**算法的实现：**

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7776068) [copy](http://blog.csdn.net/hguisu/article/details/7776068)

1. **void** bubbleSort(**int** a[], **int** n){
2. **for**(**int** i =0 ; i< n-1; ++i) {
3. **for**(**int** j = 0; j < n-i-1; ++j) {
4. **if**(a[j] > a[j+1])
5. {
6. **int** tmp = a[j] ; a[j] = a[j+1] ;  a[j+1] = tmp;
7. }
8. }
9. }
10. }

**冒泡排序算法的改进**

对冒泡排序常见的改进方法是加入一标志性变量exchange，用于标志某一趟排序过程中是否有数据交换，如果进行某一趟排序时并没有进行数据交换，则说明数据已经按要求排列好，可立即结束排序，避免不必要的比较过程。本文再提供以下两种改进算法：

1．设置一标志性变量pos,用于记录每趟排序中最后一次进行交换的位置。由于pos位置之后的记录均已交换到位,故在进行下一趟排序时只要扫描到pos位置即可。

改进后算法如下:

1. **void** Bubble\_1 ( **int** r[], **int** n) {
2. **int** i= n -1;  //初始时,最后位置保持不变
3. **while** ( i> 0) {
4. **int** pos= 0; //每趟开始时,无记录交换
5. **for** (**int** j= 0; j< i; j++)
6. **if** (r[j]> r[j+1]) {
7. pos= j; //记录交换的位置
8. **int** tmp = r[j]; r[j]=r[j+1];r[j+1]=tmp;
9. }
10. i= pos; //为下一趟排序作准备
11. }
12. }

## 快速排序

1. /\*
2. \* Java实现快速排序算法
3. \* author:wyr
4. \* 2016-7-14
5. \*/
6. **public** **class** QuickSort {
7. **public** **static** **void** main(String[] args) {
9. **int** a[] = {3,1,5,7,2,4,9,6,10,8};
10. QuickSort  obj=**new** QuickSort();
11. System.out.println("初始值：");
12. obj.print(a);
13. **int** h=a.length-1;
14. obj.quickSort(a,0,h);
15. System.out.println("\n排序后：");
16. obj.print(a);
17. }
18. **private**  **void** quickSort(**int**[] a,**int** low, **int** high) {
19. **if**(low<high){ //如果不加这个判断递归会无法退出导致堆栈溢出异常
20. **int** middle=getMiddle(a,low,high);
21. quickSort(a,  0,  middle-1);          //递归对低子表递归排序
22. quickSort(a,   middle + 1, high);        //递归对高子表递归排序
23. }
24. }
25. **public** **int** getMiddle(**int**[] a, **int** low, **int** high){
27. **int** key = a[low];//基准元素，排序中会空出来一个位置
28. **while**(low<high){
29. **while**(low<high && a[high]>=key){//从high开始找比基准小的，与low换位置
30. high--;
31. }
32. a[low]=a[high];
33. **while**(low<high && a[low]<=key){//从low开始找比基准大,放到之前high空出来的位置上
34. low++;
35. }
36. a[high]=a[low];
37. }
38. a[low]=key;//此时low=high 是基准元素的位置，也是空出来的那个位置
39. **return** low;
41. }
42. **public** **void** print(**int** a[]){
43. **for**(**int** i=0;i<a.length;i++){
44. System.out.print(a[i]+" ");
45. }
46. }
47. }

## 归并排序

1. **public** **class** MergeSortTest {
3. **public** **static** **void** main(String[] args) {
4. **int**[] data = **new** **int**[] { 5, 3, 6, 2, 1, 9, 4, 8, 7 };
5. print(data);
6. mergeSort(data);
7. System.out.println("排序后的数组：");
8. print(data);
9. }
11. **public** **static** **void** mergeSort(**int**[] data) {
12. sort(data, 0, data.length - 1);
13. }
15. **public** **static** **void** sort(**int**[] data, **int** left, **int** right) {
16. **if** (left >= right)
17. **return**;
18. // 找出中间索引
19. **int** center = (left + right) / 2;
20. // 对左边数组进行递归
21. sort(data, left, center);
22. // 对右边数组进行递归
23. sort(data, center + 1, right);
24. // 合并
25. merge(data, left, center, right);
26. print(data);
27. }
29. /\*\*
30. \* 将两个数组进行归并，归并前面2个数组已有序，归并后依然有序
31. \*
32. \* @param data
33. \*            数组对象
34. \* @param left
35. \*            左数组的第一个元素的索引
36. \* @param center
37. \*            左数组的最后一个元素的索引，center+1是右数组第一个元素的索引
38. \* @param right
39. \*            右数组最后一个元素的索引
40. \*/
41. **public** **static** **void** merge(**int**[] data, **int** left, **int** center, **int** right) {
42. // 临时数组
43. **int**[] tmpArr = **new** **int**[data.length];
44. // 右数组第一个元素索引
45. **int** mid = center + 1;
46. // third 记录临时数组的索引
47. **int** third = left;
48. // 缓存左数组第一个元素的索引
49. **int** tmp = left;
50. **while** (left <= center && mid <= right) {
51. // 从两个数组中取出最小的放入临时数组
52. **if** (data[left] <= data[mid]) {
53. tmpArr[third++] = data[left++];
54. } **else** {
55. tmpArr[third++] = data[mid++];
56. }
57. }
58. // 剩余部分依次放入临时数组（实际上两个while只会执行其中一个）
59. **while** (mid <= right) {
60. tmpArr[third++] = data[mid++];
61. }
62. **while** (left <= center) {
63. tmpArr[third++] = data[left++];
64. }
65. // 将临时数组中的内容拷贝回原数组中
66. // （原left-right范围的内容被复制回原数组）
67. **while** (tmp <= right) {
68. data[tmp] = tmpArr[tmp++];
69. }
70. }
72. **public** **static** **void** print(**int**[] data) {
73. **for** (**int** i = 0; i < data.length; i++) {
74. System.out.print(data[i] + "\t");
75. }
76. System.out.println();
77. }
79. }

**归并的迭代算法**

1 个元素的表总是有序的。所以对n 个元素的待排序列，每个元素可看成1 个有序子表。对子表两两合并生成n/2个子表，所得子表除最后一个子表长度可能为1 外，其余子表长度均为2。再进行两两合并，直到生成n 个元素按关键码有序的表。

**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7776068) [copy](http://blog.csdn.net/hguisu/article/details/7776068)

1. **void** print(**int** a[], **int** n){
2. **for**(**int** j= 0; j<n; j++){
3. cout<<a[j] <<"  ";
4. }
5. cout<<endl;
6. }
8. //将r[i…m]和r[m +1 …n]归并到辅助数组rf[i…n]
9. **void** Merge(ElemType \*r,ElemType \*rf, **int** i, **int** m, **int** n)
10. {
11. **int** j,k;
12. **for**(j=m+1,k=i; i<=m && j <=n ; ++k){
13. **if**(r[j] < r[i]) rf[k] = r[j++];
14. **else** rf[k] = r[i++];
15. }
16. **while**(i <= m)  rf[k++] = r[i++];
17. **while**(j <= n)  rf[k++] = r[j++];
18. print(rf,n+1);
19. }
21. **void** MergeSort(ElemType \*r, ElemType \*rf, **int** lenght)
22. {
23. **int** len = 1;
24. ElemType \*q = r ;
25. ElemType \*tmp ;
26. **while**(len < lenght) {
27. **int** s = len;
28. len = 2 \* s ;
29. **int** i = 0;
30. **while**(i+ len <lenght){
31. Merge(q, rf,  i, i+ s-1, i+ len-1 ); //对等长的两个子表合并
32. i = i+ len;
33. }
34. **if**(i + s < lenght){
35. Merge(q, rf,  i, i+ s -1, lenght -1); //对不等长的两个子表合并
36. }
37. tmp = q; q = rf; rf = tmp; //交换q,rf，以保证下一趟归并时，仍从q 归并到rf
38. }
39. }

42. **int** main(){
43. **int** a[10] = {3,1,5,7,2,4,9,6,10,8};
44. **int** b[10];
45. MergeSort(a, b, 10);
46. print(b,10);
47. cout<<"结果：";
48. print(a,10);
50. }

**两路归并的递归算法**

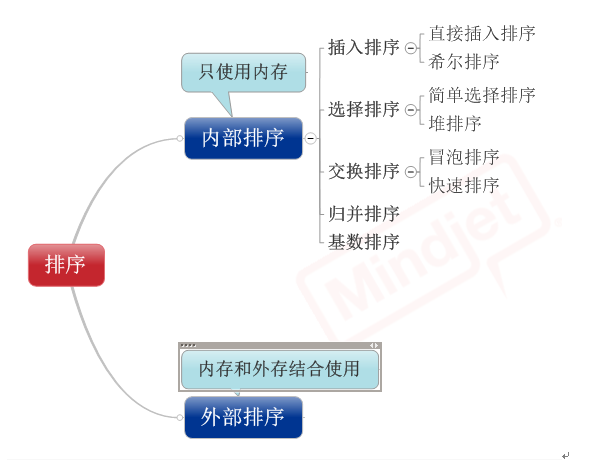
**[cpp]** [view plain](http://blog.csdn.net/hguisu/article/details/7776068) [copy](http://blog.csdn.net/hguisu/article/details/7776068)

1. **void** MSort(ElemType \*r, ElemType \*rf,**int** s, **int** t)
2. {
3. ElemType \*rf2;
4. **if**(s==t) r[s] = rf[s];
5. **else**
6. {
7. **int** m=(s+t)/2;          /\*平分\*p 表\*/
8. MSort(r, rf2, s, m);        /\*递归地将p[s…m]归并为有序的p2[s…m]\*/
9. MSort(r, rf2, m+1, t);      /\*递归地将p[m+1…t]归并为有序的p2[m+1…t]\*/
10. Merge(rf2, rf, s, m+1,t);   /\*将p2[s…m]和p2[m+1…t]归并到p1[s…t]\*/
11. }
12. }
13. **void** MergeSort\_recursive(ElemType \*r, ElemType \*rf, **int** n)
14. {   /\*对顺序表\*p 作归并排序\*/
15. MSort(r, rf,0, n-1);
16. }

## 桶排序/基数排序

1. Void RadixSort(Node L[],length,maxradix)
2. {
3. **int** m,n,k,lsp;
4. k=1;m=1;
5. **int** temp[10][length-1];
6. Empty(temp); //清空临时空间
7. **while**(k<maxradix) //遍历所有关键字
8. {
9. **for**(**int** i=0;i<length;i++) //分配过程
10. {
11. **if**(L[i]<m)
12. Temp[0][n]=L[i];
13. **else**
14. Lsp=(L[i]/m)%10; //确定关键字
15. Temp[lsp][n]=L[i];
16. n++;
17. }
18. CollectElement(L,Temp); //收集
19. n=0;
20. m=m\*10;
21. k++;
22. }
23. }

## 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

package com.njue;

public class insertSort {

public insertSort(){

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};

int temp=0;

for(int i=1;i<a.length;i++){

int j=i-1;

temp=a[i];

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

a[j+1]=a[j]; //将大于temp的值整体后移一个单位

}

a[j+1]=temp;

}

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

System.out.println(a[i]);

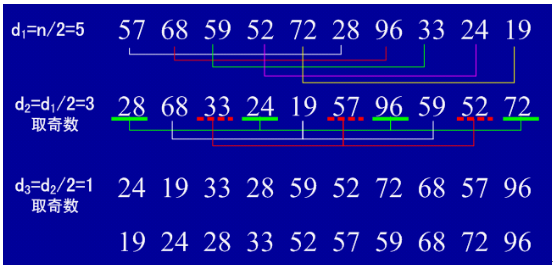
}

}

## 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

public class shellSort {

public shellSort(){

int a[]={1,54,6,3,78,34,12,45,56,100};

double d1=a.length;

int temp=0;

while(true){

d1= Math.ceil(d1/2);

int d=(int) d1;

for(int x=0;x<d;x++){

for(int i=x+d;i<a.length;i+=d){

int j=i-d;

temp=a[i];

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

a[j+d]=a[j];

}

a[j+d]=temp;

}

}

if(d==1)

break;

}

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

System.out.println(a[i]);

}

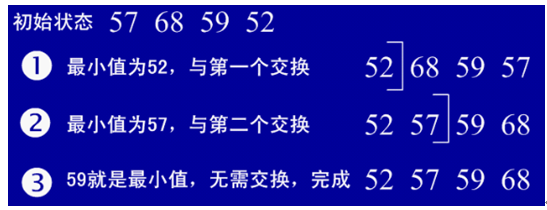
}

## 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

public class selectSort {

public selectSort(){

int a[]={1,54,6,3,78,34,12,45};

int position=0;

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

int j=i+1;

position=i;

int temp=a[i];

for(;j<a.length;j++){

if(a[j]<temp){

temp=a[j];

position=j;

}

}

a[position]=a[i];

a[i]=temp;

}

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

System.out.println(a[i]);

}

}

## 4，堆排序

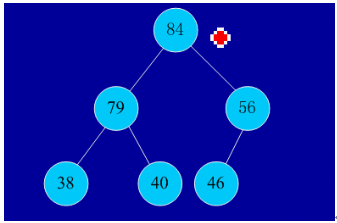
（1）基本思想：堆排序是一种树形选择排序，是对直接选择排序的有效改进。

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& gt;=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。

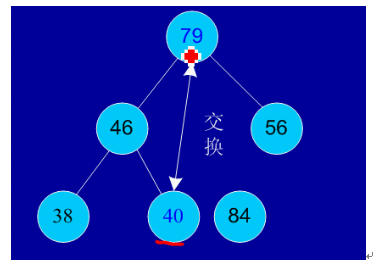
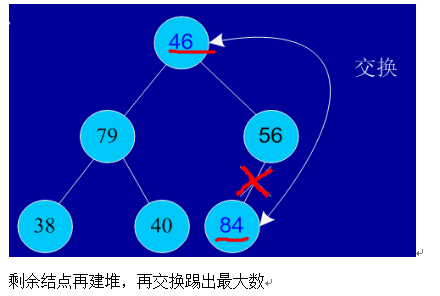
（2）实例：

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

import java.util.Arrays;

public class HeapSort {

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};

public HeapSort(){

heapSort(a);

}

public void heapSort(int[] a){

System.out.println("开始排序");

int arrayLength=a.length;

//循环建堆

for(int i=0;i<arrayLength-1;i++){

//建堆

buildMaxHeap(a,arrayLength-1-i);

//交换堆顶和最后一个元素

swap(a,0,arrayLength-1-i);

System.out.println(Arrays.toString(a));

}

}

private void swap(int[] data, int i, int j) {

// TODO Auto-generated method stub

int tmp=data[i];

data[i]=data[j];

data[j]=tmp;

}

//对data数组从0到lastIndex建大顶堆

private void buildMaxHeap(int[] data, int lastIndex) {

// TODO Auto-generated method stub

//从lastIndex处节点（最后一个节点）的父节点开始

for(int i=(lastIndex-1)/2;i>=0;i--){

//k保存正在判断的节点

int k=i;

//如果当前k节点的子节点存在

while(k\*2+1<=lastIndex){

//k节点的左子节点的索引

int biggerIndex=2\*k+1;

//如果biggerIndex小于lastIndex，即biggerIndex+1代表的k节点的右子节点存在

if(biggerIndex<lastIndex){

//若果右子节点的值较大

if(data[biggerIndex]<data[biggerIndex+1]){

//biggerIndex总是记录较大子节点的索引

biggerIndex++;

}

}

//如果k节点的值小于其较大的子节点的值

if(data[k]<data[biggerIndex]){

//交换他们

swap(data,k,biggerIndex);

//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

k=biggerIndex;

}else{

break;

}

}

}

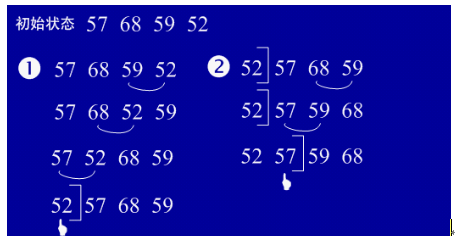
}

}

## 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

public class bubbleSort {

public bubbleSort(){

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};

int temp=0;

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

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

if(a[j]>a[j+1]){

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

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

System.out.println(a[i]);

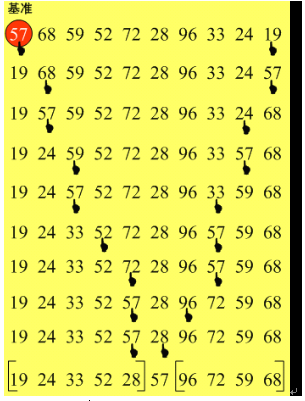
}

}

## 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

public class quickSort {

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};

public quickSort(){

quick(a);

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

System.out.println(a[i]);

}

public int getMiddle(int[] list, int low, int high) {

int tmp = list[low]; //数组的第一个作为中轴

while (low < high) {

while (low < high && list[high] >= tmp) {

high--;

}

list[low] = list[high]; //比中轴小的记录移到低端

while (low < high && list[low] <= tmp) {

low++;

}

list[high] = list[low]; //比中轴大的记录移到高端

}

list[low] = tmp; //中轴记录到尾

return low; //返回中轴的位置

}

public void \_quickSort(int[] list, int low, int high) {

if (low < high) {

int middle = getMiddle(list, low, high); //将list数组进行一分为二

\_quickSort(list, low, middle - 1); //对低字表进行递归排序

\_quickSort(list, middle + 1, high); //对高字表进行递归排序

}

}

public void quick(int[] a2) {

if (a2.length > 0) { //查看数组是否为空

\_quickSort(a2, 0, a2.length - 1);

}

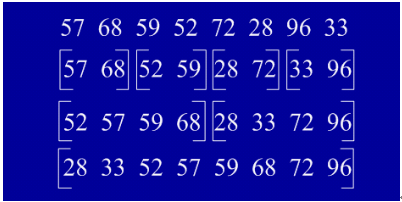
}

}

## 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**

import java.util.Arrays;

public class mergingSort {

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};

public mergingSort(){

sort(a,0,a.length-1);

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

System.out.println(a[i]);

}

public void sort(int[] data, int left, int right) {

// TODO Auto-generated method stub

if(left<right){

//找出中间索引

int center=(left+right)/2;

//对左边数组进行递归

sort(data,left,center);

//对右边数组进行递归

sort(data,center+1,right);

//合并

merge(data,left,center,right);

}

}

public void merge(int[] data, int left, int center, int right) {

// TODO Auto-generated method stub

int [] tmpArr=new int[data.length];

int mid=center+1;

//third记录中间数组的索引

int third=left;

int tmp=left;

while(left<=center&&mid<=right){

//从两个数组中取出最小的放入中间数组

if(data[left]<=data[mid]){

tmpArr[third++]=data[left++];

}else{

tmpArr[third++]=data[mid++];

}

}

//剩余部分依次放入中间数组

while(mid<=right){

tmpArr[third++]=data[mid++];

}

while(left<=center){

tmpArr[third++]=data[left++];

}

//将中间数组中的内容复制回原数组

while(tmp<=right){

data[tmp]=tmpArr[tmp++];

}

System.out.println(Arrays.toString(data));

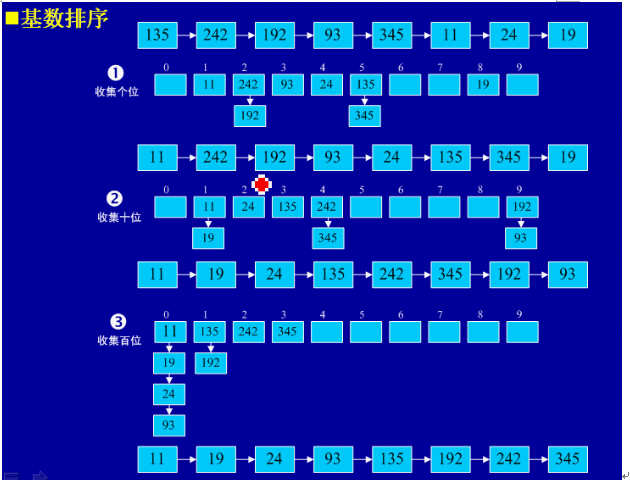
}

}

## 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

import java.util.ArrayList;

import java.util.List;

public class radixSort {

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};

public radixSort(){

sort(a);

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

System.out.println(a[i]);

}

public void sort(int[] array){

//首先确定排序的趟数;

int max=array[0];

for(int i=1;i<array.length;i++){

if(array[i]>max){

max=array[i];

}

}

int time=0;

//判断位数;

while(max>0){

max/=10;

time++;

}

//建立10个队列;

List<ArrayList> queue=new ArrayList<ArrayList>();

for(int i=0;i<10;i++){

ArrayList<Integer> queue1=new ArrayList<Integer>();

queue.add(queue1);

}

//进行time次分配和收集;

for(int i=0;i<time;i++){

//分配数组元素;

for(int j=0;j<array.length;j++){

//得到数字的第time+1位数;

int x=array[j]%(int)Math.pow(10, i+1)/(int)Math.pow(10, i);

ArrayList<Integer> queue2=queue.get(x);

queue2.add(array[j]);

queue.set(x, queue2);

}

int count=0;//元素计数器;

//收集队列元素;

for(int k=0;k<10;k++){

while(queue.get(k).size()>0){

ArrayList<Integer> queue3=queue.get(k);

array[count]=queue3.get(0);

queue3.remove(0);

count++;

}

}

}

}

}