各种排序算法的分析及java实现

排序一直以来都是让我很头疼的事，以前上《数据结构》打酱油去了，整个学期下来才勉强能写出个冒泡排序。由于下半年要准备工作了，也知道排序算法的重要性（据说是面试必问的知识点），所以又花了点时间重新研究了一下。

排序大的分类可以分为两种：内排序和外排序。在排序过程中，全部记录存放在内存，则称为内排序，如果排序过程中需要使用外存，则称为外排序。下面讲的排序都是属于内排序。

内排序有可以分为以下几类：

(1)、插入排序：直接插入排序、二分法插入排序、希尔排序。

(2)、选择排序：简单选择排序、堆排序。

(3)、交换排序：冒泡排序、快速排序。

(4)、归并排序

(5)、基数排序

一、插入排序

•思想：每步将一个待排序的记录，按其顺序码大小插入到前面已经排序的字序列的合适位置，直到全部插入排序完为止。
•关键问题：在前面已经排好序的序列中找到合适的插入位置。
•方法：
–直接插入排序
–二分插入排序
–希尔排序

①直接插入排序（从后向前找到合适位置后插入）

1、基本思想：每步将一个待排序的记录，按其顺序码大小插入到前面已经排序的字序列的合适位置（从后向前找到合适位置后），直到全部插入排序完为止。

2、实例

3、java实现

```
1 package com.sort;
2
3 public class 直接插入排序 {
4
5     public static void main(String[] args) {
6         int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1};
7         System.out.println("排序之前：");
8         for (int i = 0; i < a.length; i++) {
9             System.out.print(a[i]+" ");
10        }
11        //直接插入排序
12        for (int i = 1; i < a.length; i++) {
13            //待插入元素
14            int temp = a[i];
```

```
15              int j;
16              /*for (j = i-1; j>=0 && a[j]>temp; j--) {
17                  //将大于temp的往后移动一位
18                  a[j+1] = a[j];
19              }*/
20              for (j = i-1; j>=0; j--) {
21                  //将大于temp的往后移动一位
22                  if(a[j]>temp){
23                      a[j+1] = a[j];
24                  }else{
25                      break;
26                  }
27              }
28              a[j+1] = temp;
29          }
30          System.out.println();
31          System.out.println("排序之后: ");
32          for (int i = 0; i < a.length; i++) {
33              System.out.print(a[i]+" ");
34          }
35      }
36
37  }
```

4、分析

直接插入排序是稳定的排序。关于各种算法的稳定性分析可以参考http://www.cnblogs.com/Braveliu/archive/2013/01/15/2861201.html

文件初态不同时，直接插入排序所耗费的时间有很大差异。若文件初态为正序，则每个待插入的记录只需要比较一次就能够找到合适的位置插入，故算法的时间复杂度为O(n)，这时最好的情况。若初态为反序，则第i个待插入记录需要比较i+1次才能找到合适位置插入，故时间复杂度为$O(n^2)$，这时最坏的情况。

直接插入排序的平均时间复杂度为O(n2)。

②二分法插入排序（按二分法找到合适位置插入）

1、基本思想：二分法插入排序的思想和直接插入一样，只是找合适的插入位置的方式不同，这里是按二分法找到合适的位置，可以减少比较的次数。

2、实例

3、java实现

```java
package com.sort;

public class 二分插入排序 {
    public static void main(String[] args) {
        int[] a={49,38,65,97,176,213,227,49,78,34,12,164,11,18,1};
        System.out.println("排序之前：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
        //二分插入排序
        sort(a);
        System.out.println();
        System.out.println("排序之后：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
    }

    private static void sort(int[] a) {
        for (int i = 0; i < a.length; i++) {
            int temp = a[i];
            int left = 0;
            int right = i-1;
            int mid = 0;
            while(left<=right){
                mid = (left+right)/2;
                if(temp<a[mid]){
                    right = mid-1;
                }else{
                    left = mid+1;
                }
            }
            for (int j = i-1; j >= left; j--) {
                a[j+1] = a[j];
            }
            if(left != i){
                a[left] = temp;
            }
        }
    }
}
```

4、分析

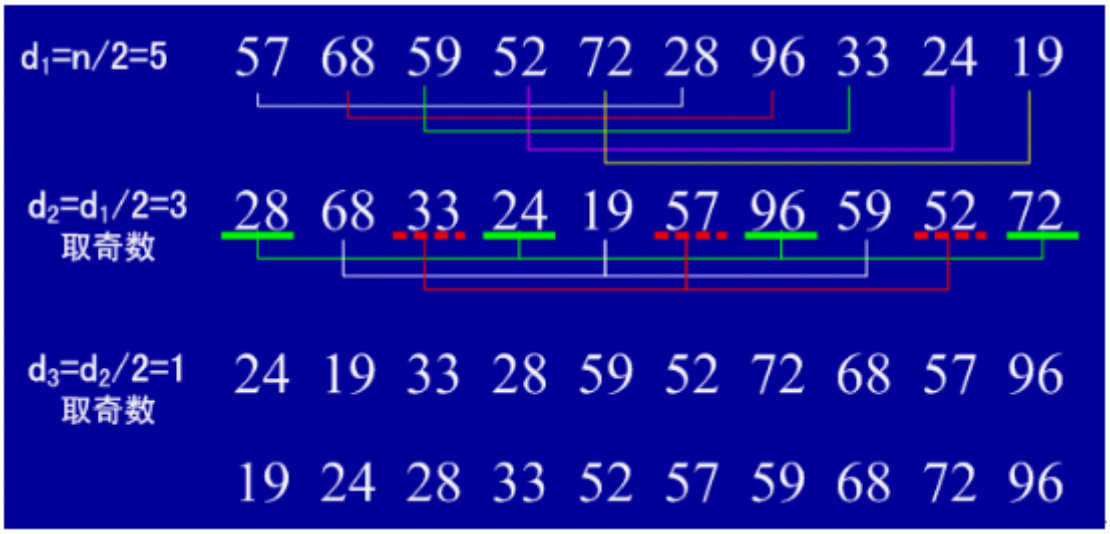
当然，二分法插入排序也是稳定的。

二分插入排序的比较次数与待排序记录的初始状态无关，仅依赖于记录的个数。当n较大时，比直接插入排序的最大比较次数少得多。但大于直接插入排序的最小比较次数。算法的移动次数与直接插入排序算法的相同，最坏的情况为n2/2，最好的情况为n，平均移动次数为O(n2)。

③希尔排序

1、基本思想：先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。所有距离为d1的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量d2<d1重复上述的分组和排序，直至所取的增量dt=1(dt<dt-l<…<d2<d1)，即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

2、实例



3、java实现

```
package com.sort;

//不稳定
public class 希尔排序 {


    public static void main(String[] args) {
        int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1};
        System.out.println("排序之前：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
        //希尔排序
        int d = a.length;
        while(true){
            d = d / 2;
            for(int x=0;x<d;x++){
                for(int i=x+d;i<a.length;i=i+d){
                    int temp = a[i];
                    int j;
                    for(j=i-d;j>=0&&a[j]>temp;j=j-d){
                        a[j+d] = a[j];
                    }
                    a[j+d] = temp;
                }
            }
            if(d == 1){
                break;
            }
        }
        System.out.println();
        System.out.println("排序之后：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
    }

}
```

4、分析

我们知道一次插入排序是稳定的，但在不同的插入排序过程中，相同的元素可能在各自的插入排序中移动，最后其稳定性就会被打乱，所以希尔排序是不稳定的。

希尔排序的时间性能优于直接插入排序，原因如下：

（1）当文件初态基本有序时直接插入排序所需的比较和移动次数均较少。

（2）当n值较小时，n和n2的差别也较小，即直接插入排序的最好时间复杂度O(n)和最坏时间复杂度0(n2)差别不大。

（3）在希尔排序开始时增量较大，分组较多，每组的记录数目少，故各组内直接插入较快，后来增量di逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多，但由于已经按di-1作为距离排过序，使文件较接近于有序状态，所以新的一趟排序过程也较快。

因此，希尔排序在效率上较直接插人排序有较大的改进。

希尔排序的平均时间复杂度为O(nlogn)。

二、选择排序

•思想：每趟从待排序的记录序列中选择关键字最小的记录放置到已排序表的最前位置，直到全部排完。

•关键问题：在剩余的待排序记录序列中找到最小关键码记录。

•方法：

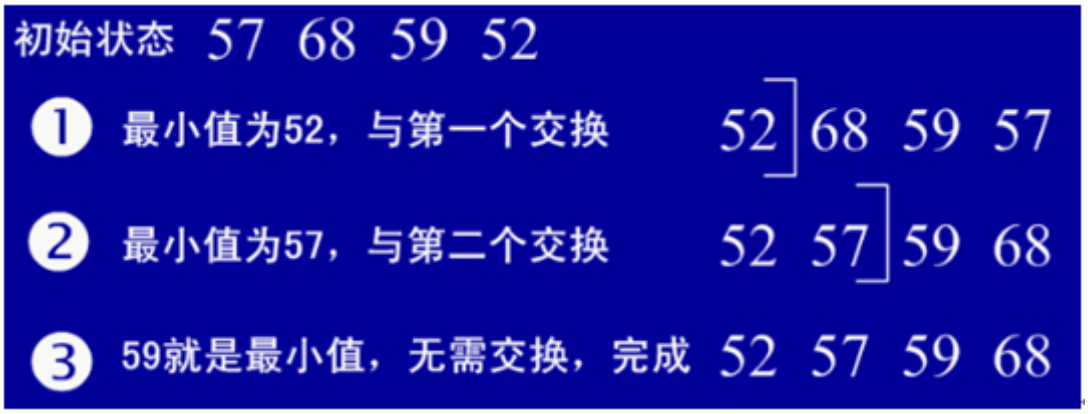
–直接选择排序

–堆排序

①简单的选择排序

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、java实现

```java
package com.sort;

//不稳定
public class 简单的选择排序 {

    public static void main(String[] args) {
        int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1,8};
        System.out.println("排序之前：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
        //简单的选择排序
        for (int i = 0; i < a.length; i++) {
            int min = a[i];
            int n=i; //最小数的索引
            for(int j=i+1;j<a.length;j++){
                if(a[j]<min){   //找出最小的数
                    min = a[j];
                    n = j;
                }
            }
            a[n] = a[i];
            a[i] = min;

        }
        System.out.println();
        System.out.println("排序之后：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
    }

}
```

4、分析

简单选择排序是不稳定的排序。

时间复杂度：T(n)=O(n2)。

②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。

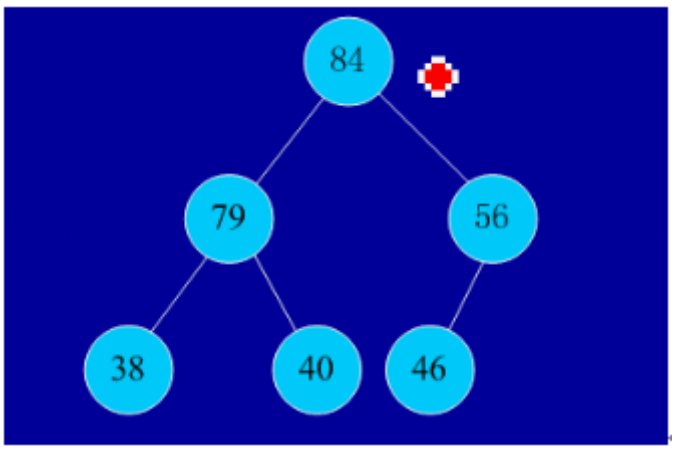
堆的定义下：具有n个元素的序列 （h1,h2,...,hn),当且仅当满足（hi>=h2i,hi>=2i+1）或（hi<=h2i,hi<=2i+1） (i=1,2,...,n/2)时称之为堆。在这里只讨论满足前者条件的堆。由堆的定义可以看出，堆顶元素（即第一个元素）必为最大项（大顶堆）。完全二叉树可以很直观地表示堆的结构。堆顶为根，其它为左子树、右子树。

思想: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。

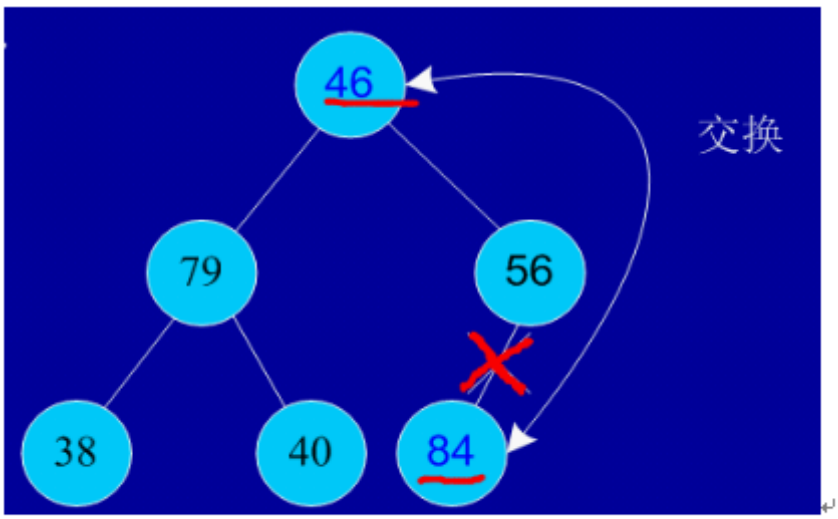
2、实例

初始序列：46,79,56,38,40,84

建堆：



交换，从堆中踢出最大数



剩余结点再建堆，再交换踢出最大数

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、java实现

```java
package com.sort;
//不稳定
import java.util.Arrays;

public class HeapSort {
    public static void main(String[] args) {
        int[] a={49,38,65,97,76,13,27,49,78,34,12,64};
        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));
        }
    }
    //对data数组从0到lastIndex建大顶堆
    public static void buildMaxHeap(int[] data, int lastIndex){
         //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;
                }
            }
        }
    }
    //交换
    private static void swap(int[] data, int i, int j) {
        int tmp=data[i];
        data[i]=data[j];
        data[j]=tmp;
    }
}
```

4、分析

堆排序也是一种不稳定的排序算法。

堆排序优于简单选择排序的原因：

直接选择排序中，为了从R[1..n]中选出关键字最小的记录，必须进行n-1次比较，然后在R[2..n]中选出关键字最小的记录，又需要做n-2次比较。事实上，后面的n-2次比较中，有许多比较可能在前面的n-1次比较中已经做过，但由于前一趟排序时未保留这些比较结果，所以后一趟排序时又重复执行了这些比较操作。

堆排序可通过树形结构保存部分比较结果，可减少比较次数。

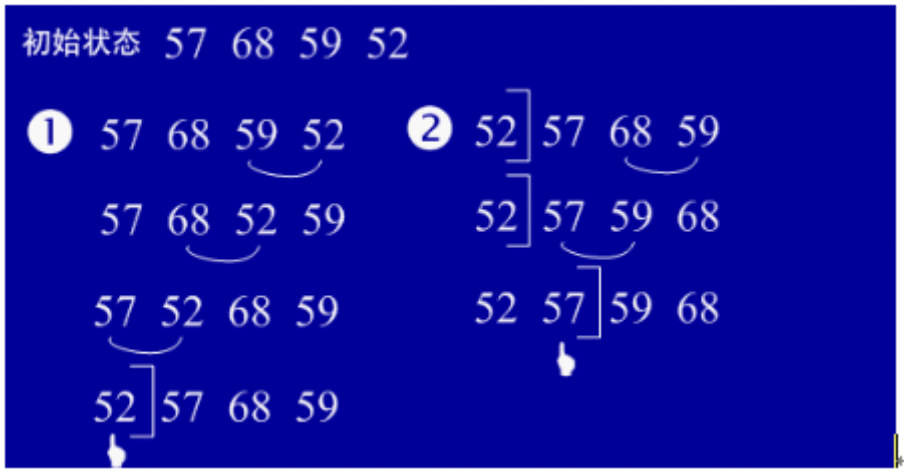
堆排序的最坏时间复杂度为**O(nlogn)**。堆序的平均性能较接近于最坏性能。由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的文件。

三、交换排序

①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、java实现

```
1 package com.sort;
2
3 //稳定
4 public class 冒泡排序 {
5     public static void main(String[] args) {
6         int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1,8};
7         System.out.println("排序之前：");
8         for (int i = 0; i < a.length; i++) {
9             System.out.print(a[i]+" ");
10        }
11        //冒泡排序
12        for (int i = 0; i < a.length; i++) {
13            for(int j = 0; j<a.length-i-1; j++){
14                //这里-i主要是每遍历一次都把最大的i个数沉到最底下去了，没有必要再替换了
15                if(a[j]>a[j+1]){
16                    int temp = a[j];
17                    a[j] = a[j+1];
18                    a[j+1] = temp;
19                }
20            }
21        }
22        System.out.println();
23        System.out.println("排序之后：");
24        for (int i = 0; i < a.length; i++) {
25            System.out.print(a[i]+" ");
26        }
27    }
28 }
```

4、分析

冒泡排序是一种稳定的排序方法。

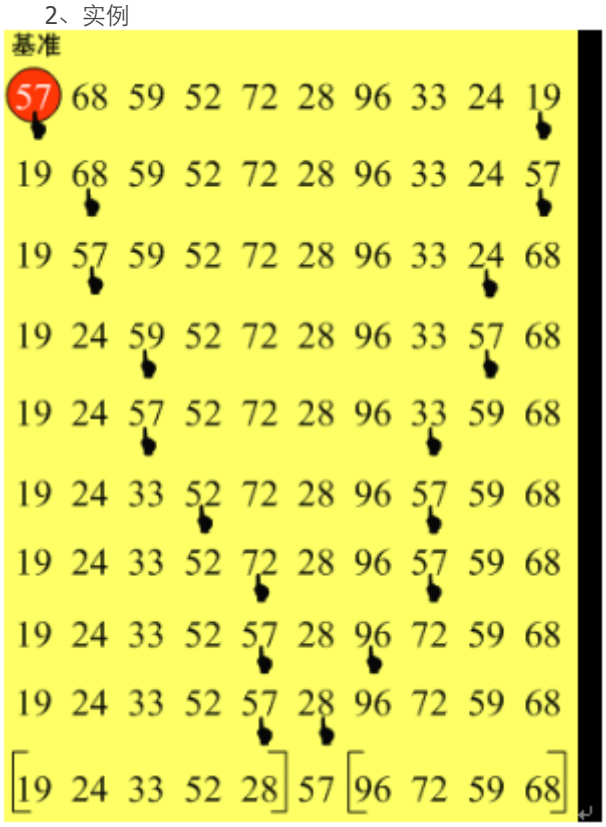
•若文件初状为正序，则一趟起泡就可完成排序，排序码的比较次数为n-1，且没有记录移动，时间复杂度是O(n)
•若文件初态为逆序，则需要n-1趟起泡，每趟进行n-i次排序码的比较，且每次比较都移动三次，比较和移动次数均达到最大值：O(n2)

•起泡排序平均时间复杂度为O(n2)

②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、java实现

```
package com.sort;

//不稳定
public class 快速排序 {
    public static void main(String[] args) {
        int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1,8};
        System.out.println("排序之前：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
        //快速排序
        quick(a);
        System.out.println();
        System.out.println("排序之后：");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]+" ");
        }
    }

    private static void quick(int[] a) {
        if(a.length>0){
            quickSort(a,0,a.length-1);
        }
    }

    private static void quickSort(int[] a, int low, int high) {
        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
            int middle = getMiddle(a,low,high);
            quickSort(a, 0, middle-1);
            quickSort(a, middle+1, high);
        }
    }
```

```
    private static int getMiddle(int[] a, int low, int high) {
        int temp = a[low];//基准元素
        while(low<high){
            //找到比基准元素小的元素位置
            while(low<high && a[high]>=temp){
                high--;
            }
            a[low] = a[high];
            while(low<high && a[low]<=temp){
                low++;
            }
            a[high] = a[low];
        }
        a[low] = temp;
        return low;
    }
}
```

4、分析

快速排序是不稳定的排序。

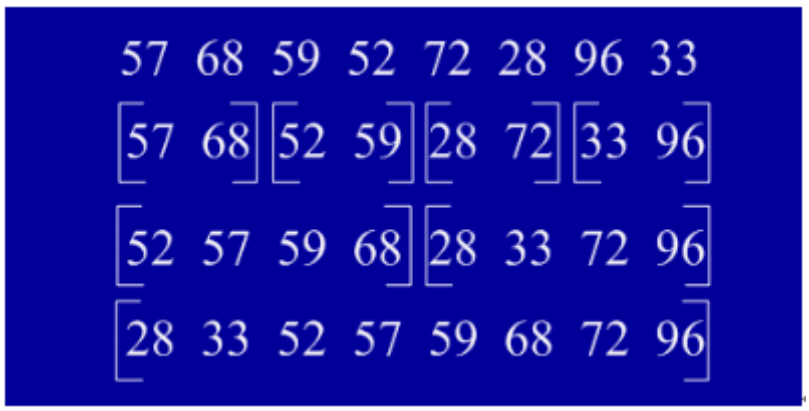
快速排序的时间复杂度为O(nlogn)。

当n较大时使用快排比较好，当序列基本有序时用快排反而不好。

四、归并排序

1、基本思想: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、java实现

```
1 package com.sort;
2
3 //稳定
4 public class 归并排序 {
5     public static void main(String[] args) {
6         int[] a={49,38,65,97,76,13,27,49,78,34,12,64,1,8};
7         System.out.println("排序之前：");
8         for (int i = 0; i < a.length; i++) {
9             System.out.print(a[i]+" ");
10        }
11        //归并排序
12        mergeSort(a,0,a.length-1);
13        System.out.println();
14        System.out.println("排序之后：");
15        for (int i = 0; i < a.length; i++) {
16            System.out.print(a[i]+" ");
17        }
```

```
18      }
19
20      private static void mergeSort(int[] a, int left, int right) {
21          if(left<right){
22              int middle = (left+right)/2;
23              //对左边进行递归
24              mergeSort(a, left, middle);
25              //对右边进行递归
26              mergeSort(a, middle+1, right);
27              //合并
28              merge(a,left,middle,right);
29          }
30      }
31
32      private static void merge(int[] a, int left, int middle, int right) {
33          int[] tmpArr = new int[a.length];
34          int mid = middle+1; //右边的起始位置
35          int tmp = left;
36          int third = left;
37          while(left<=middle && mid<=right){
38              //从两个数组中选取较小的数放入中间数组
39              if(a[left]<=a[mid]){
40                  tmpArr[third++] = a[left++];
41              }else{
42                  tmpArr[third++] = a[mid++];
43              }
44          }
45          //将剩余的部分放入中间数组
46          while(left<=middle){
47              tmpArr[third++] = a[left++];
48          }
49          while(mid<=right){
50              tmpArr[third++] = a[mid++];
51          }
52          //将中间数组复制回原数组
53          while(tmp<=right){
54              a[tmp] = tmpArr[tmp++];
55          }
56      }
57  }
```

4、分析

归并排序是稳定的排序方法。

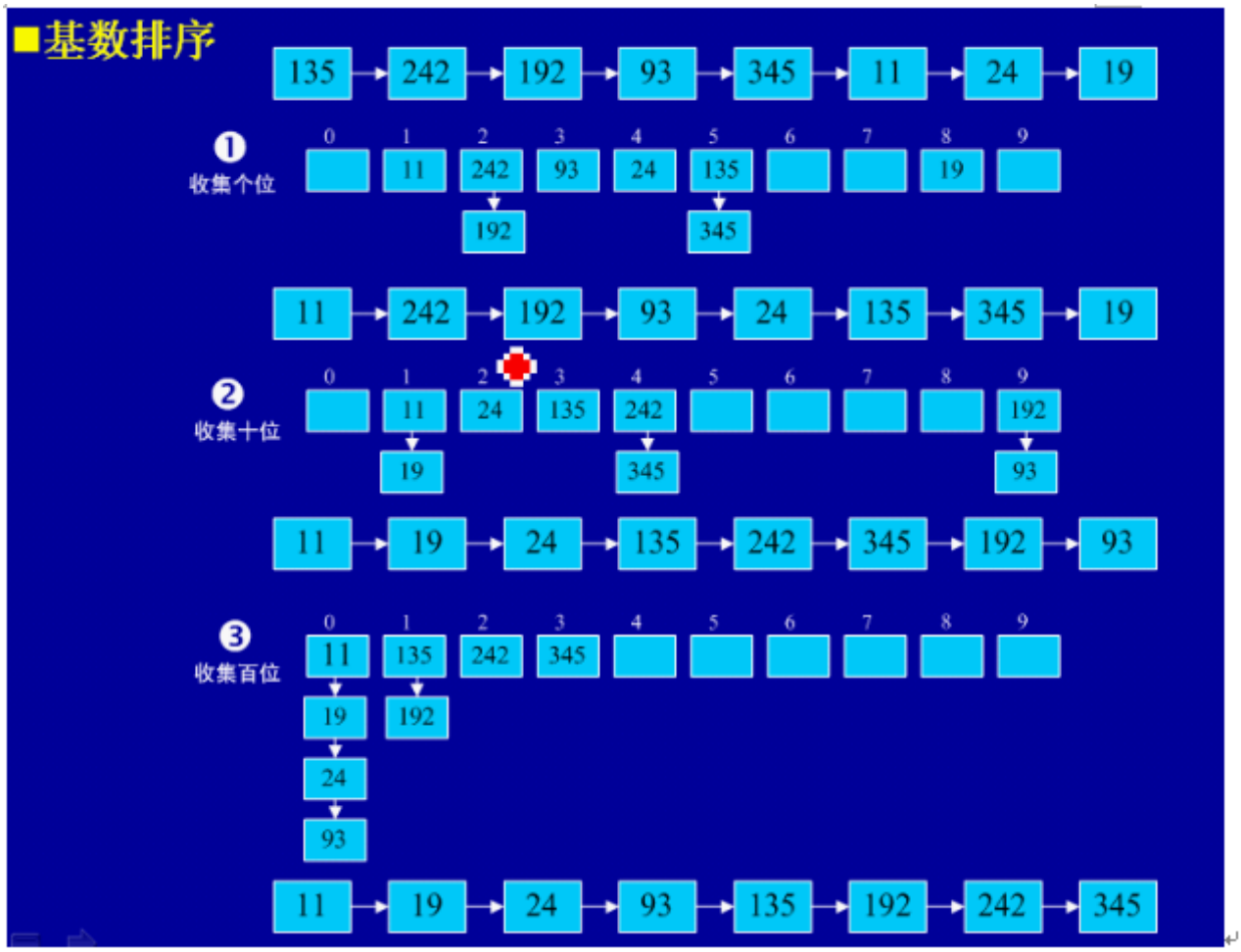
归并排序的时间复杂度为O(nlogn)。

速度仅次于快速排序，为稳定排序算法，一般用于对总体无序，但是各子项相对有序的数列。

五、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、java实现

```
1  package com.sort;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  //稳定
6  public class 基数排序 {
7      public static void main(String[] args) {
8          int[] a={49,38,65,97,176,213,227,49,78,34,12,164,11,18,1};
9          System.out.println("排序之前：");
10         for (int i = 0; i < a.length; i++) {
11             System.out.print(a[i]+" ");
12         }
13         //基数排序
14         sort(a);
15         System.out.println();
16         System.out.println("排序之后：");
17         for (int i = 0; i < a.length; i++) {
18             System.out.print(a[i]+" ");
19         }
20     }
21
22     private static void sort(int[] array) {
23         //找到最大数，确定要排序几趟
24         int max = 0;
25         for (int i = 0; i < array.length; i++) {
26             if(max<array[i]){
27                 max = array[i];
28             }
29         }
30         //判断位数
31         int times = 0;
32         while(max>0){
33             max = max/10;
34             times++;
```

```
35          }
36          //建立十个队列
37          List<ArrayList> queue = new ArrayList<ArrayList>();
38          for (int i = 0; i < 10; i++) {
39              ArrayList queue1 = new ArrayList();
40              queue.add(queue1);
41          }
42          //进行times次分配和收集
43          for (int i = 0; i < times; i++) {
44              //分配
45              for (int j = 0; j < array.length; j++) {
46                  int x = array[j]%(int)Math.pow(10, i+1)/(int)Math.pow(10, i);
47                  ArrayList queue2 = queue.get(x);
48                  queue2.add(array[j]);
49                  queue.set(x,queue2);
50              }
51              //收集
52              int count = 0;
53              for (int j = 0; j < 10; j++) {
54                  while(queue.get(j).size()>0){
55                      ArrayList<Integer> queue3 = queue.get(j);
56                      array[count] = queue3.get(0);
57                      queue3.remove(0);
58                      count++;
59                  }
60              }
61          }
62      }
63  }
```

4、分析

基数排序是稳定的排序算法。

基数排序的时间复杂度为O(d(n+r)),d为位数，r为基数。

总结:

一、稳定性:

稳定：冒泡排序、插入排序、归并排序和基数排序

不稳定：选择排序、快速排序、希尔排序、堆排序

二、平均时间复杂度

O(n^2):直接插入排序，简单选择排序，冒泡排序。

在数据规模较小时（9W内），直接插入排序，简单选择排序差不多。当数据较大时，冒泡排序算法的时间代价最高。性能为O(n^2)的算法基本上是相邻元素进行比较，基本上都是稳定的。

O(nlogn):快速排序，归并排序，希尔排序，堆排序。

其中，快排是最好的， 其次是归并和希尔，堆排序在数据量很大时效果明显。

三、排序算法的选择

1.数据规模较小

（1）待排序列基本序的情况下，可以选择直接插入排序；

（2）对稳定性不作要求宜用简单选择排序，对稳定性有要求宜用插入或冒泡

2.数据规模不是很大

（1）完全可以用内存空间，序列杂乱无序，对稳定性没有要求，快速排序，此时要付出log（N）的额外空间。

（2）序列本身可能有序，对稳定性有要求，空间允许下，宜用归并排序

3.数据规模很大

（1）对稳定性有求，则可考虑归并排序。

（2）对稳定性没要求，宜用堆排序

4.序列初始基本有序（正序），宜用直接插入，冒泡