

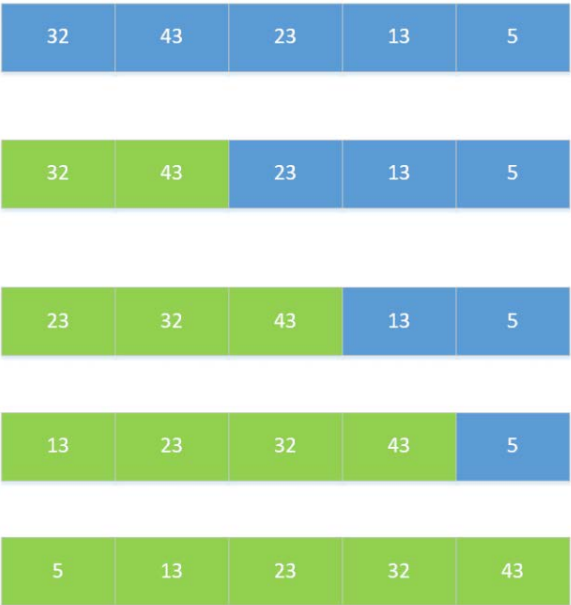


Java常用的八种排序算法与代码实现

排序问题一直是程序员工作与面试的重点，今天特意整理研究下与大家共勉！这里列出8种常见的经典排序，基本涵盖了所有的排序算法。

1.直接插入排序

我们会经常遇到这样一类排序问题：把新的数据插入到已经排好的数据列中。将第一个数和第二个数排序，然后构成一个有序序列将第三个数插入进去，构成一个新的有序序列。对第四个数、第五个数.....直到最后一个数，重复第二步。如题所示：



直接插入排序（Straight Insertion Sorting）的基本思想：在要排序的一组数中，假设前面 $(n-1)$ $[n \geq 2]$ 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。

代码实现：


首先设定插入次数，即循环次数，`for(int i=1;i<length;i++)`，1个数的那次不用插入。

设定插入数和得到已经排好序列的最后一个数的位数。`insertNum`和`j=i-1`。


从最后一个数开始向前循环，如果插入数小于当前数，就将当前数向后移动一位。

将当前数放置到空着的位置，即`j+1`。

代码如下：



```
1 public void insertSort(int [] a){
2     int len=a.length;//单独把数组长度拿出来，提高效率
3     int insertNum;//要插入的数
4     for(int i=1;i<len;i++){//因为第一次不用，所以从1开始
5         insertNum=a[i];
6         int j=i-1;//序列元素个数
7         while(j>=0&& a[j]>insertNum){//从后往前循环，将大于insertNum的数向后移动
8             a[j+1]=a[j];//元素向后移动
9             j--;
10        }
11        a[j+1]=insertNum;//找到位置，插入当前元素
12    }
13 }
```



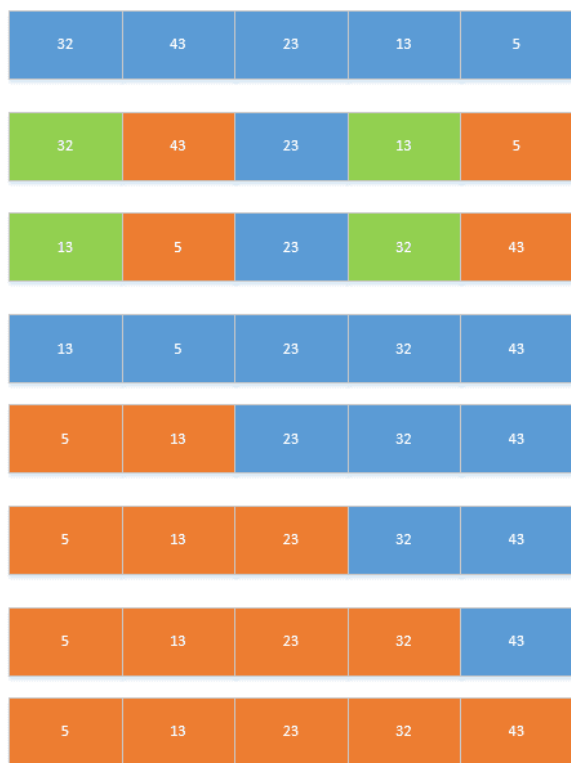
2.希尔排序

针对直接插入排序的下效率问题，有人对次进行了改进与升级，这就是现在的希尔排序。**希尔排序**，也称**递减增量排序算法**，是插入排序的一种更高效的改进版本。希尔排序是非稳定排序算法。

希尔排序是基于插入排序的以下两点性质而提出改进方法的：

- 插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率
- 但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位

如图所示：



对于直接插入排序问题，数据量巨大时。

将数的个数设为 n ，取奇数 $k=n/2$ ，将下标差值为 k 的数分为一组，构成有序序列。

再取 $k=k/2$ ，将下标差值为 k 的书分为一组，构成有序序列。

重复第二步，直到 $k=1$ 执行简单插入排序。

代码实现：

首先确定分的组数。

然后对组中元素进行插入排序。

然后将 $length/2$ ，重复1,2步，直到 $length=0$ 为止。



```
1 public void sheelSort(int [] a){
2     int len=a.length;//单独把数组长度拿出来，提高效率
3     while(len!=0){
4         len=len/2;
5         for(int i=0;i<len;i++){//分组
6             for(int j=i+len;j<a.length;j+=len){//元素从第二个开始
7                 int k=j-len;//k为有序序列最后一位的位数
8                 int temp=a[j];//要插入的元素
9                 /*for(;k>=0&&temp<a[k];k-=len){
10                     a[k+len]=a[k];
11                 }*/
12                 while(k>=0&&temp<a[k]){//从后往前遍历
13                     a[k+len]=a[k];
```

```
14         k--len; //向后移动len位
15     }
16     a[k+len]=temp;
17 }
18 }
19 }
20 }
```

3.简单选择排序

常用于取序列中最大最小的几个数时。

(如果每次比较都交换，那么就是交换排序；如果每次比较完一个循环再交换，就是简单选择排序。)

遍历整个序列，将最小的数放在最前面。

遍历剩下的序列，将最小的数放在最前面。

重复第二步，直到只剩下一个数。



代码实现：

首先确定循环次数，并且记住当前数字和当前位置。

将当前位置后面所有的数与当前数字进行对比，小数赋值给key，并记住小数的位置。

比对完成后，将最小的值与第一个数的值交换。

重复2、3步。

```
1 public void selectSort(int[]a){
2     int len=a.length;
3     for(int i=0;i<len;i++){//
```

循环次数

```

4         int value=a[i];
5         int position=i;
6         for(int j=i+1;j<len;j++){//找到最小的值和位置
7             if(a[j]<value){
8                 value=a[j];
9                 position=j;
10            }
11        }
12        a[position]=a[i];//进行交换
13        a[i]=value;
14    }
15 }

```



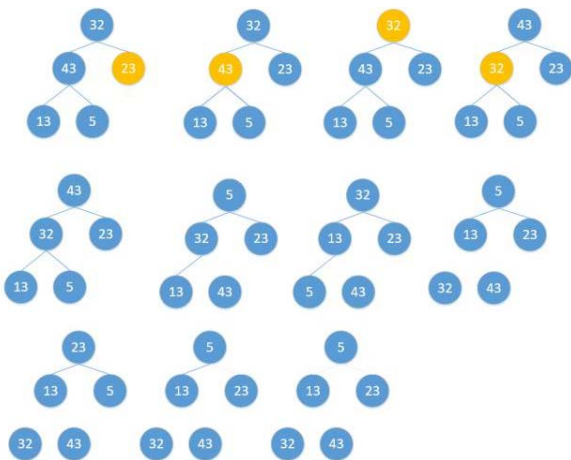
4.堆排序

对简单选择排序的优化。

将序列构建成大顶堆。

将根节点与最后一个节点交换，然后断开最后一个节点。

重复第一、二步，直到所有节点断开。



代码如下：



```

1 public void heapSort(int[] a){
2     int len=a.length;
3     //循环建堆
4     for(int i=0;i<len-1;i++){
5         //建堆
6         buildMaxHeap(a,len-1-i);
7         //交换堆顶和最后一个元素
8         swap(a,0,len-1-i);
9     }

```

```

10     }
11     //交换方法
12     private void swap(int[] data, int i, int j) {
13         int tmp=data[i];
14         data[i]=data[j];
15         data[j]=tmp;
16     }
17     //对data数组从0到lastIndex建大顶堆
18     private void buildMaxHeap(int[] data, int lastIndex) {
19         //从lastIndex处节点（最后一个节点）的父节点开始
20         for(int i=(lastIndex-1)/2;i>=0;i--){
21             //k保存正在判断的节点
22             int k=i;
23             //如果当前k节点的子节点存在
24             while(k*2+1<=lastIndex){
25                 //k节点的左子节点的索引
26                 int biggerIndex=2*k+1;
27                 //如果biggerIndex小于lastIndex, 即biggerIndex+1代表的k节点的右子节点存在
28                 if(biggerIndex<lastIndex){
29                     //若果右子节点的值较大
30                     if(data[biggerIndex]<data[biggerIndex+1]){
31                         //biggerIndex总是记录较大子节点的索引
32                         biggerIndex++;
33                     }
34                 }
35                 //如果k节点的值小于其较大的子节点的值
36                 if(data[k]<data[biggerIndex]){
37                     //交换他们
38                     swap(data,k,biggerIndex);
39                     //将biggerIndex赋予k, 开始while循环的下一轮循环, 重新保证k节点的值大于其左右子节点
40                     k=biggerIndex;
41                 }else{
42                     break;
43                 }
44             }
45         }
46     }

```



5.冒泡排序

很简单，用到的很少，据了解，面试的时候问的比较多！

将序列中所有元素两两比较，将最大的放在最后面。

将剩余序列中所有元素两两比较，将最大的放在最后面。

重复第二步，直到只剩下一个数。

32	43	23	13	5
32	43	23	13	5
32	23	43	13	5
32	23	13	43	5
32	23	13	5	43
23	32	13	5	43
23	13	32	5	43
23	13	5	32	43
13	23	5	32	43
13	5	23	32	43
5	13	23	32	43

代码实现：

设置循环次数。

设置开始比较的位数，和结束的位数。

两两比较，将最小的放到前面去。

重复2、3步，直到循环次数完毕。



```
1 public void bubbleSort(int []a){
2     int len=a.length;
3     for(int i=0;i<len;i++){
4         for(int j=0;j<len-i-1;j++){//注意第二重循环的条件
5             if(a[j]>a[j+1]){
6                 int temp=a[j];
7                 a[j]=a[j+1];
8                 a[j+1]=temp;
9             }
10        }
11    }
12 }
```



6.快速排序

要求时间最快时。

选择第一个数为p，小于p的数放在左边，大于p的数放在右边。

递归的将p左边和右边的数都按照第一步进行，直到不能递归。

32	43	23	13	5
----	----	----	----	---

23	13	5	32	43
----	----	---	----	----

13	5	23	32	43
----	---	----	----	----

5	13	23	32	43
---	----	----	----	----



```

1 public void quickSort(int[] a,int start,int end){
2     if(start<end){
3         int baseNum=a[start]; //选基准值
4         int midNum; //记录中间值
5         int i=start;
6         int j=end;
7         do{
8             while((a[i]<baseNum)&&i<end){
9                 i++;
10            }
11            while((a[j]>baseNum)&&j>start){
12                j--;
13            }
14            if(i<=j){
15                midNum=a[i];
16                a[i]=a[j];
17                a[j]=midNum;
18                i++;
19                j--;
20            }
21        }while(i<=j);
22        if(start<j){
23            quickSort(a,start,j);
24        }
25        if(end>i){
26            quickSort(a,i,end);
27        }
28    }

```


29 }



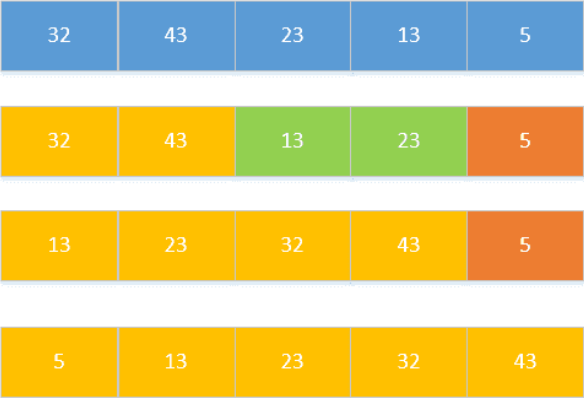
7.归并排序

速度仅次于快速排序，内存少的时候使用，可以进行并行计算的时候使用。

选择相邻两个数组成一个有序序列。

选择相邻的两个有序序列组成一个有序序列。

重复第二步，直到全部组成一个有序序列。



```
1 public void mergeSort(int[] a, int left, int right) {
2     int t = 1; // 每组元素个数
3     int size = right - left + 1;
4     while (t < size) {
5         int s = t; // 本次循环每组元素个数
6         t = 2 * s;
7         int i = left;
8         while (i + (t - 1) < size) {
9             merge(a, i, i + (s - 1), i + (t - 1));
10            i += t;
11        }
12        if (i + (s - 1) < right)
13            merge(a, i, i + (s - 1), right);
14    }
15 }
16
17 private static void merge(int[] data, int p, int q, int r) {
18     int[] B = new int[data.length];
19     int s = p;
20     int t = q + 1;
21     int k = p;
22     while (s <= q && t <= r) {
23         if (data[s] <= data[t]) {
24             B[k] = data[s];
25             s++;
```

```

26         } else {
27             B[k] = data[t];
28             t++;
29         }
30         k++;
31     }
32     if (s == q + 1)
33         B[k++] = data[t++];
34     else
35         B[k++] = data[s++];
36     for (int i = p; i <= r; i++)
37         data[i] = B[i];
38 }

```



8.基数排序

用于大量数，很长的数进行排序时。

将所有的数的个位数取出，按照个位数进行排序，构成一个序列。

将新构成的所有的数的十位数取出，按照十位数进行排序，构成一个序列。

代码实现：



```

1 public void baseSort(int[] a) {
2     //首先确定排序的趟数;
3     int max = a[0];
4     for (int i = 1; i < a.length; i++) {
5         if (a[i] > max) {
6             max = a[i];
7         }
8     }
9     int time = 0;
10    //判断位数;
11    while (max > 0) {
12        max /= 10;
13        time++;
14    }
15    //建立10个队列;
16    List<ArrayList<Integer>> queue = new ArrayList<ArrayList<Integer>>();
17    for (int i = 0; i < 10; i++) {
18        ArrayList<Integer> queue1 = new ArrayList<Integer>();
19        queue.add(queue1);
20    }
21    //进行time次分配和收集;
22    for (int i = 0; i < time; i++) {

```

```

23         //分配数组元素;
24         for (int j = 0; j < a.length; j++) {
25             //得到数字的第time+1位数;
26             int x = a[j] % (int) Math.pow(10, i + 1) / (int) Math.pow(10, i);
27             ArrayList<Integer> queue2 = queue.get(x);
28             queue2.add(a[j]);
29             queue.set(x, queue2);
30         }
31         int count = 0; //元素计数器;
32         //收集队列元素;
33         for (int k = 0; k < 10; k++) {
34             while (queue.get(k).size() > 0) {
35                 ArrayList<Integer> queue3 = queue.get(k);
36                 a[count] = queue3.get(0);
37                 queue3.remove(0);
38                 count++;
39             }
40         }
41     }
42 }

```



新建测试类进行测试



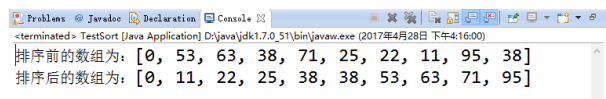
```

1 public class TestSort {
2     public static void main(String[] args) {
3         int []a=new int[10];
4         for(int i=1;i<a.length;i++){
5             //a[i]=(int)(new Random().nextInt(100));
6             a[i]=(int)(Math.random()*100);
7         }
8         System.out.println("排序前的数组为："+Arrays.toString(a));
9         Sort s=new Sort();
10        //排序方法测试
11        //s.insertSort(a);
12        //s.sheelSort(a);
13        //s.selectSort(a);
14        //s.heapSort(a);
15        //s.bubbleSort(a);
16        //s.quickSort(a, 1, 9);
17        //s.mergeSort(a, 3, 7);
18        s.baseSort(a);
19        System.out.println("排序后的数组为："+Arrays.toString(a));
20    }
21
22 }

```



部分结果如下:



如果要进行比较可已加入时间，输出排序时间，从而比较各个排序算法的优缺点，这里不再做介绍。

8.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。

$O(n\log n)$: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

各算法复杂度如下：

排序法	最差时间分析	平均时间复杂度	稳定度	空间复杂度
选择排序	$O(n^2)$	$O(n^2)$	稳定	$O(1)$
插入排序	$O(n^2)$	$O(n^2)$	稳定	$O(1)$
冒泡排序	$O(n^2)$	$O(n^2)$	稳定	$O(1)$
快速排序	$O(n^2)$	$O(n*\log_2n)$	不稳定	$O(\log_2n) \sim O(n)$
归并排序	$O(n^2)$	$O(n*\log n)$	稳定	不一定
希尔排序	$O(n*(\log n)^2)$	$O(n*(\log n)^2)$	不稳定	$O(1)$
堆排序	$O(n*\log_2n)$	$O(n*\log_2n)$	不稳定	$O(1)$
基数排序	$O(kn)$	$O(n\log(r)m)$	稳定	$O(kn)$

部分参考资料来源于：

<http://blog.csdn.net/without0815/article/details/7697916>

分类： 经典算法

好文要顶

关注我

收藏该文



我心自在
关注 - 2
粉丝 - 18

+加关注

2

推荐

1

反对

- « 上一篇：大数据及hadoop相关知识介绍
- » 下一篇：hadoop集群搭建--CentOS部署Hadoop服务

posted @ 2017-04-28 16:25 我心自在 阅读(18829) 评论(12) 编辑 收藏

评论列表

#1楼 2018-03-03 22:11 Young2017

归并排序和基数排序的标题都写的是7

支持(0) 反对(0)

#2楼 2018-03-06 14:12 Young2017

博主您好，你的MergeSort算法有bug,在使用测试用例{4,3,2}进行排序时会出现2 3 0的结果

支持(0) 反对(0)

#3楼[楼主] 2018-03-21 22:56 我心自在 

@ Young2017

非常感谢，我会尽快更正

支持(0) 反对(0)

#4楼 2018-04-04 20:15 _xiao鱼 

大哥，你写完都不自己测的么

支持(0) 反对(0)

#5楼 2018-04-06 10:05 _xiao鱼 

除了第一个插入排序while中j > 0 需要改为 j>=0，其余的都可以用，感谢博主，学习了。

支持(2) 反对(0)

#6楼 2018-05-04 14:00 三号小学生 

支持

支持(0) 反对(0)

#7楼 2018-05-11 10:28 <虎口脱险> 

个人觉得，代码如果不经测试，就发出来分享，对于程序员来说，太不严谨，算法里面bug太多，建议修正之后在发表出来

支持(2) 反对(0)

#8楼 2018-06-02 15:32 MartinBockZhu 

@ _xiao鱼

引用

除了第一个插入排序while中j > 0 需要改为 j>=0，其余的都可以用，感谢博主，学习了。

对 难怪我测试的一直不对，博主还没有修改 这很误导人啊

支持(0) 反对(0)

#9楼 2018-06-04 11:31 一生爱你哦 

第一个就有问题，你那个赋值是什么东西？insertNum=a[i]; 这是啥意思，将传入的数字覆盖了

支持(0) 反对(0)

#10楼 2018-06-04 23:34 MartinBockZhu 

@ 一生爱你哦

insertNum;//是要插入的数

支持(0) 反对(0)

#11楼 2018-07-01 11:05 打包爱 

希尔排序例子 32 43 23 13 5

不是将 32 23 5 直接构成一组了吗

支持(0) 反对(0)

#12楼 2018-07-01 11:36 打包爱 

选择排序稳定性到底是稳定还是不稳定，表格跟上面一处地方不一致

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

努力加载评论框中...

最新IT新闻:

- 蓝色起源第九次试射火箭 测试极限逃生技术
- 还记得三岁前发生过什么吗？最初的记忆也许从未发生
- 爱立信第二财季亏损扩大：当季裁员2000人
- 京东向技术转型背后 挑战和忧虑是什么？
- 马斯克向英国潜水员道歉：这都是我的错
- » 更多新闻...

最新知识库文章:

- 危害程序员职业生涯的三大观念
- 断点单步跟踪是一种低效的调试方法
- 测试 | 让每一粒尘埃有的放矢
- 从Excel到微服务
- 如何提升你的能力？给年轻程序员的几条建议
- » 更多知识库文章...

公告

昵称：我心自在
园龄：1年2个月
粉丝：18

关注：2

<	2018年7月						>
日	一	二	三	四	五	六	
24	25	26	27	28	29	30	
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30	31	1	2	3	4	

搜索

随笔分类

HttpClient研究学习总结(1)

javaEE项目(1)

javaWeb开发研究学习(1)

Java多线程(2)

Linux(1)

netty(1)

servlet学习总结(1)

WebService学习总结(3)

大数据/hadoop(6)

经典算法(2)

项目经验(12)

随笔档案

2018年4月 (1)

2018年3月 (2)

2018年2月 (3)

2018年1月 (6)

2017年6月 (2)

2017年5月 (2)

2017年4月 (8)

最新评论

阅读排行榜

评论排行榜

推荐排行榜