Java数据结构和算法（九）——高级排序

目录

- 1、希尔排序
  - ①、直接插入排序
  - ②、希尔排序图解
  - ③、排序间隔选取
  - ④、knuth间隔序列的希尔排序算法实现
  - ⑤、间隔为2h的希尔排序
- 2、快速排序
  - ①、快速排序的基本思路
  - ②、快速排序的算法实现
  - ③、快速排序图示
  - ④、快速排序完整代码
  - ⑤、优化分析

春晚好看吗？不存在的！！！

在Java数据结构和算法（三）——冒泡、选择、插入排序算法中我们介绍了三种简单的排序算法，它们的时间复杂度大O表示法都是O(N^2)，如果数据量少，我们还能忍受，但是数据量大，那么这三种简单的排序所需要的时间则是我们所不能接受的。接着我们在讲解递归 的时候，介绍了归并排序，归并排序需要O(NlogN)，这比简单排序要快了很多，但是归并排序有个缺点，它需要的空间是原始数组空间的两倍，当我们需要排序的数据占据了整个内存的一半以上的空间，那么是不能使用归并排序的。

本篇博客将介绍几种高级的排序算法：希尔排序和快速排序。

回到顶部

1、希尔排序

希尔排序是基于直接插入排序的，它在直接插入排序中增加了一个新特性，大大的提高了插入排序的执行效率。所以在讲解希尔排序之前，我们先回顾一下直接插入排序。

①、直接插入排序

直接插入排序基本思想是每一步将一个待排序的记录，插入到前面已经排好序的有序序列中去，直到插完所有元素为止。

昵称：YSOcean
园龄：2年
粉丝：2067
关注：13
+加关注

< 2019年3月 >

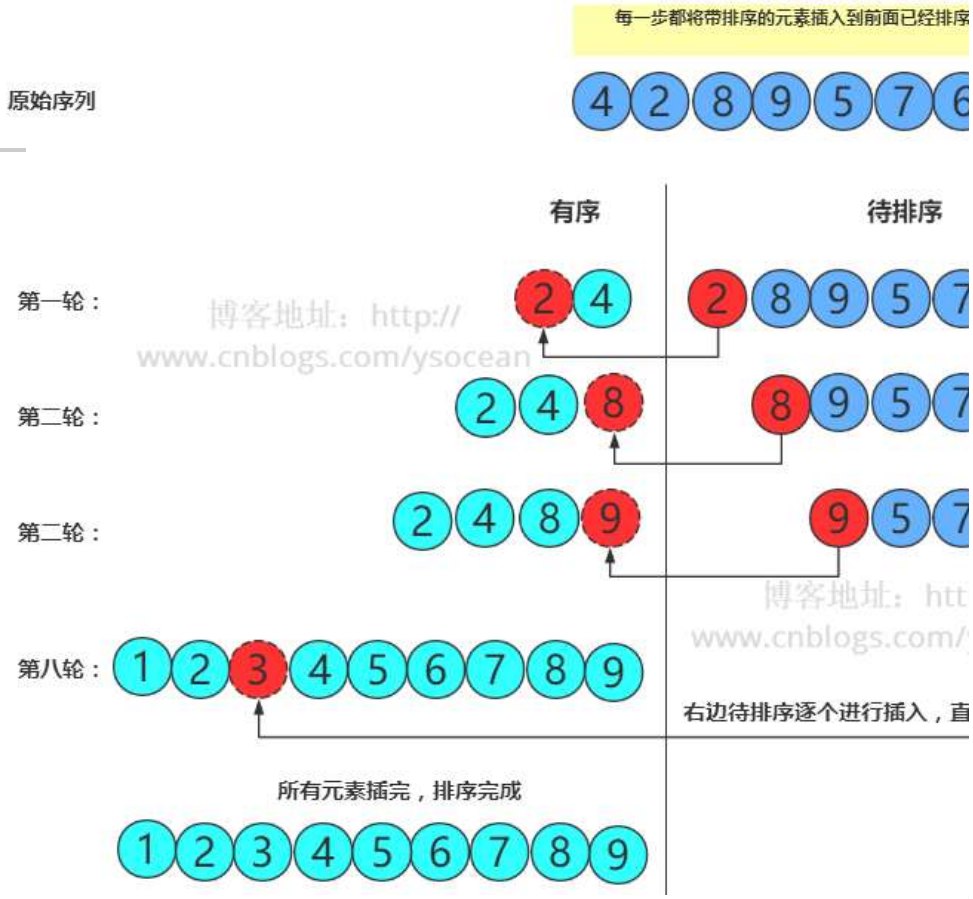
日	一	二	三	四	五	六
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

我的标签

Linux系列教程(25)
深入理解计算机系统(24)
Java数据结构和算法(16)
MyBatis详解系列(11)
JDK源码解析(11)
Maven系列教程(8)
Redis详解(8)
Spring入门系列(8)
Java IO详解系列(7)
Java高并发设计(7)
更多

随笔分类

Java SE(22)
JavaWeb(34)
Java高并发
Java关键字(6)
Java数据结构和算法(15)



实现代码为：

```
package com.ys.sort;

public class InsertSort {
    public static int[] sort(int[] array){
        int j;
        //从下标为1的元素开始选择合适的位置插入，因为下标为0的只有一个元素，默认是有序的
        for(int i = 1 ; i < array.length ; i++){
            int tmp = array[i];//记录要插入的数据
            j = i;
            while(j > 0 && tmp < array[j-1]){//从已经排序的序列最右边的开始比较，找到比其小的数
                array[j] = array[j-1];//向后挪动
                j--;
            }
            array[j] = tmp;//存在比其小的数，插入
        }
        return array;
    }

}
```

我们可以分析一下这个直接插入排序，首先我们将需要插入的数放在一个临时变量中，这也是一个标记符，标记符左边的数是已经排好序的，标记符右边的数是需要排序的。接着将标记的数和左边排好序的数进行比较，假如比目标数大则将左边排好序的数向右边移动一位，直到找到比其小的位置进行插入。

这里就存在一个效率问题了，如果一个很小的数在很靠近右边的位置，比如上图右边待排序的数据 1 ，那么想让这个很小的数 1 插入到左边排好序的位置，那么左边排好序的数据项都必须向右移动一位，这个步骤就是将近执行了N次复制，虽然不是每个数据项都必须移动N个位置，但是每个数据项平均移动了N/2次，总共就是$N^2/2$，因此插入排序的效率是$O(N^2)$。

那么如果以某种方式不必一个一个移动中间所有的数据项，就能把较小的数据项移动到左边，那么这个算法的执行效率会有很大的改进。

②、希尔排序图解

Java虚拟机
JDK源码解析(11)
Linux(9)
Linux详解(24)
Nginx详解(4)
Redis详解(8)
TCP/IP协议
编程小技巧(1)
查找算法(1)
大数据(3)
工具使用(15)
计算机系统与结构(24)
计算机组成与系统结构
浪潮之巅(1)
排序算法
前端(5)
日常工作问题(7)
设计模式(1)
算法分析(1)
消息中间件(6)
邮件服务(4)

积分与排名

积分 - 441370

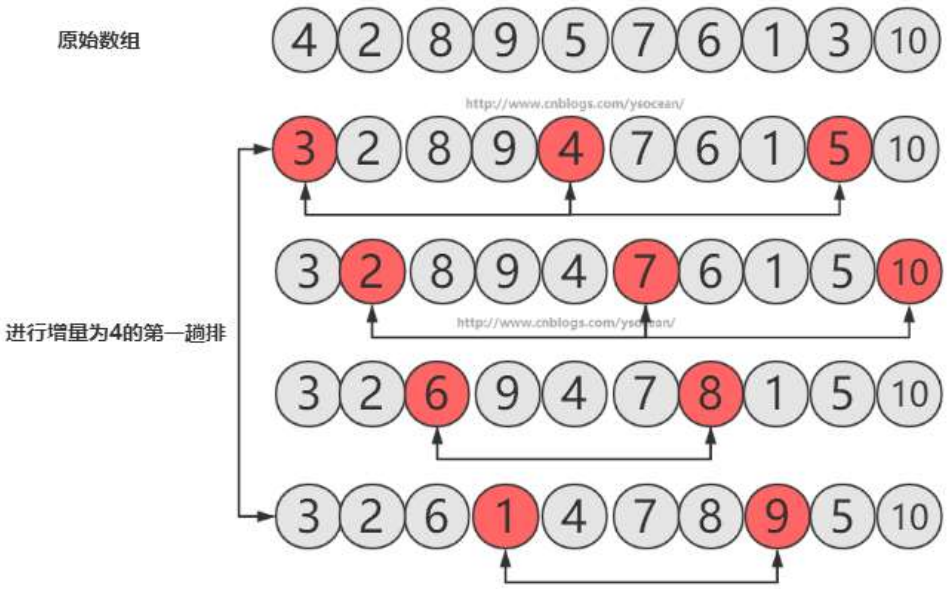
排名 - 415

阅读排行榜

1. Tomcat 部署项目的三种方法(152987)
2. Java 集合详解(74896)
3. Java数据结构和算法（一）——简介(62792)
4. MyBatis 详解（一对一，一对多，多对多）(62095)

希尔排序应运而生了，希尔排序通过加大插入排序中元素的间隔，并在这些有间隔的元素中进行插入排序，从而使数据项能够大跨度的移动。当这些数据项排过一趟序后，希尔排序算法减小数据项的间隔再进行排序，依次进行下去，最后间隔为1时，就是我们上面说的简单的直接插入排序。

下图显示了增量为4时对包含10个数组元素进行排序的第一个步骤，首先对下标为 0,4,8 的元素进行排序，完成排序之后，算法右移一步，对 1,5,9 号元素进行排序，依次类推，直到所有的元素完成一趟排序，也就是说间隔为4的元素都已经排列有序。



当我们完成4-增量排序之后，在进行普通的插入排序，即1-增量排序，会比前面直接执行简单插入排序要快很多。

③、排序间隔选取

对于10个元素，我们选取4的间隔，那么100个数据，1000个数据，甚至更多的数据，我们应该怎么选取间隔呢？

希尔的原稿中，他建议间隔选为N/2，也就是每一趟都将排序分为两半，因此对于N=100的数组，逐渐减小的间隔序列为：50,25,12，6,3,1。这个方法的好处是不需要在开始排序前为找到初始序列的间隔而计算序列，只需要用2整除N。但是这已经被证明并不是最好的序列。

间隔序列中的数字互质是很重要的指标，也就是说，除了1，他们没有公约数。这个约束条件使得每一趟排序更有可能保持前一趟排序已经排好的结果，而希尔最初以N/2的间隔的低效性就是没有遵守这个准则。

所以一种希尔的变形方法是用2.2来整除每一个间隔，对于n=100的数组，会产生序列45，20，9,4,1。这比用2会整除会显著的改善排序效果。

还有一种很常用的间隔序列：**knuth 间隔序列 3h+1**

h	3*h+1	(h-1)/3
1	4	
4	13	1
13	40	4
40	121	13
121	364	40
364	1093	121
1093	3280	364

但是无论是什么间隔序列，最后必须满足一个条件，就是逐渐减小的间隔最后一定要等于1，因此最后一趟排序一定是简单的插入排序。

下面我们通过knuth间隔序列来实现希尔排序：

④、knuth间隔序列的希尔排序算法实现

5. Java数据结构和算法（七）——链表(46203)

评论排行榜

1. 深入理解计算机系统（1.1）------Hello World 是如何运行的(27)

2. SpringMVC详解（四）-----SSM三大框架整合之登录功能实现(23)

3. 深入理解计算机系统（序章）------谈程序员为什么要懂底层计算机结构(20)

4. Tomcat 部署项目的三种方法(18)

5. Java数据结构和算法（十）——二叉树(18)

```java
//希尔排序 knuth 间隔序列 3h+1
public static void shellKnuthSort(int[] array){
    System.out.println("原数组为"+Arrays.toString(array));
    int step = 1 ;
    int len = array.length;
    while(step <= len/3){
        step = step*3 + 1;//1,4,13,40.....
    }
    while(step > 0){
        //分别对每个增量间隔进行排序
        for(int i = step ; i < len ; i++){
            int temp = array[i];
            int j = i;
            while(j > step-1 && temp <= array[j-step]){
                array[j] = array[j-step];
                j -= step;
            }
            array[j] = temp;
        }//end for
        System.out.println("间隔为"+step+"的排序结果为"+Arrays.toString(array));
        step = (step-1)/3;
    }//end while(step>0)

    System.out.println("最终排序: "+Arrays.toString(array));
}
```

测试结果：

```java
public static void main(String[] args) {
    int[] array = {4,2,8,9,5,7,6,1,3,10};
    shellKnuthSort(array);
}
```

```
原数组为[4, 2, 8, 9, 5, 7, 6, 1, 3, 10]
间隔为4的排序结果为[3, 2, 6, 1, 4, 7, 8, 9, 5, 10]
间隔为1的排序结果为[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

⑤、间隔为2h的希尔排序

```java
//希尔排序 间隔序列2h
public static void shellSort(int[] array){
    System.out.println("原数组为"+Arrays.toString(array));
    int step;
    int len = array.length;
    for(step = len/2 ;step > 0 ; step /= 2){
        //分别对每个增量间隔进行排序
        for(int i = step ; i < array.length ; i++){
            int j = i;
            int temp = array[j];
            if(array[j] < array[j-step]){
                while(j-step >=0 && temp < array[j-step]){
                    array[j] = array[j-step];
                    j -= step;
                }
                array[j] = temp;
            }
        }
        System.out.println("间隔为"+step+"的排序结果为"+Arrays.toString(array));
    }
}
```

测试结果：

```
原数组为[4, 2, 8, 9, 5, 7, 6, 1, 3, 10]
间隔为5的排序结果为[4, 2, 1, 3, 5, 7, 6, 8, 9, 10]
间隔为2的排序结果为[1, 2, 4, 3, 5, 7, 6, 8, 9, 10]
间隔为1的排序结果为[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

回到顶部

2、快速排序

快速排序是对冒泡排序的一种改进，由C. A. R. Hoare在1962年提出的一种划分交换排序，采用的是分治策略（一般与递归结合使用），以减少排序过程中的比较次数。

①、快速排序的基本思路

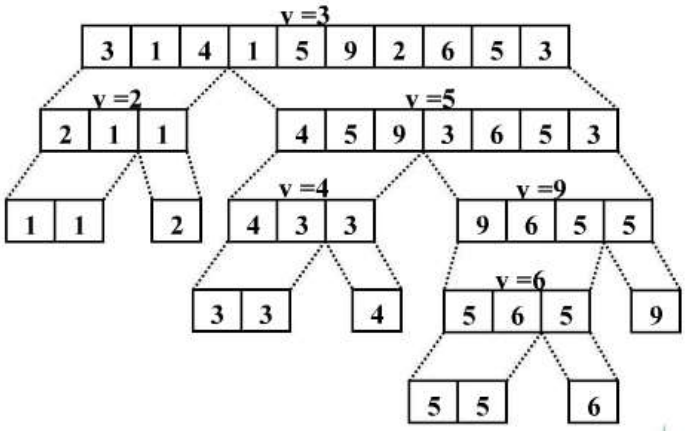
一、先通过第一趟排序，将数组原地划分为两部分，其中一部分的所有数据都小于另一部分的所有数据。原数组被划分为2份

二、通过递归的处理， 再对原数组分割的两部分分别划分为两部分，同样是使得其中一部分的所有数据都小于另一部分的所有数据。 这个时候原数组被划分为了4份

三、就1,2被划分后的最小单元子数组来看，它们仍然是无序的，但是！ 它们所组成的原数组却逐渐向有序的方向前进。

四、这样不断划分到最后，数组就被划分为多个由一个元素或多个相同元素组成的单元，这样数组就有序了。

具体实例：



对于上图的数组[3,1,4,1,5,9,2,6,5,3]，通过第一趟排序将数组分成了[2,1,1]或[4,5,9,3,6,5,3]两个子数组，且对于任意元素，左边子数组总是小于右边子数组。通过不断的递归处理，最终得到有序数组[1 1 2 3 3 4 5 5 6]

②、快速排序的算法实现

假设被排序的无序区间为[A[i],.....,A[j]]

一、基准元素选取：选择其中的一个记录的关键字 v 作为基准元素（控制关键字）;**怎么选取关键字?**

二、划分：通过基准元素 v 把无序区间 A[I].....A[j] 划分为左右两部分，使得左边的各记录的关键字都**小于** v；右边的各记录的关键字都**大于等于** v； **（如何划分？）**

三、递归求解：重复上面的一、二步骤，分别对左边和右边两部分递归进行快速排序。

四、组合：左、右两部分均有序，那么整个序列都有序。

上面的第 三、四步不用多说，主要是第一步怎么选取关键字，从而实现第二步的划分?

划分的过程涉及到三个关键字： **“基准元素”、“左游标”、“右游标”**

基准元素：它是将数组划分为两个子数组的过程中，用于界定大小的值，以它为判断标准，将小于它的数组元素“划分”到一个“小数值的数组”中，而将大于它的数组元素“划分”到一个“大数值的数组”中，这样，我们就将数组分割为两个子数组，而其中一个子数组的元素恒小于另一个子数组里的元素。

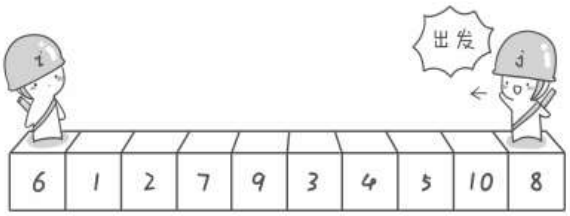
左游标：它一开始指向待分割数组最左侧的数组元素，在排序的过程中，它将向右移动。

右游标：它一开始指向待分割数组最右侧的数组元素，在排序的过程中，它将向左移动。

注意：上面描述**的基准元素/右游标/左游标**都是**针对单趟排序过程**的， 也就是说，在整体排序过程的**多趟排序中**，各趟排序取得的基准元素/右游标/左游标**一般都是不同的。**

对于基准元素的选取，原则上是任意的。但是一般我们选取数组中第一个元素为基准元素（假设数组是随机分布的）

③、快速排序图示

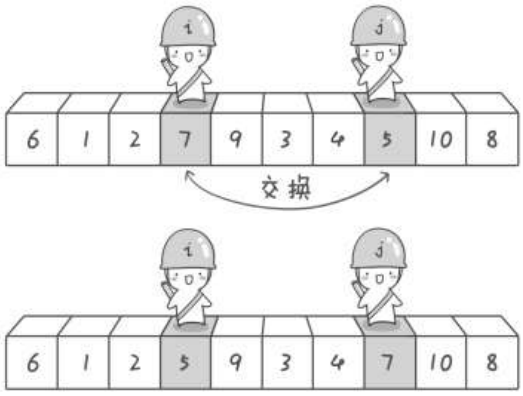


上面表示的是一个无序数组，选取第一个元素 6 作为基准元素。左游标是 i 哨兵，右游标是 j 哨兵。然后左游标向左移动，右游标向右移动，它们遵循的规则如下：

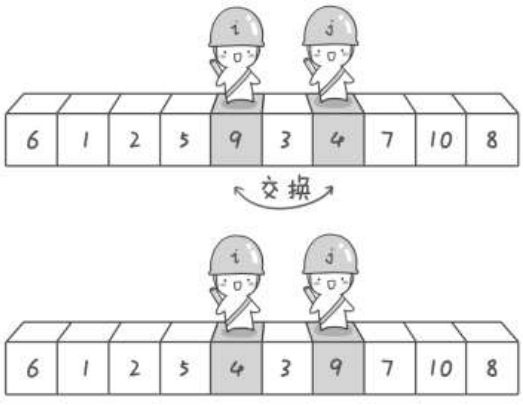
一、**左游标**向**右**扫描， **跨过所有小于基准元素的数组元素**，直到遇到一个**大于或等于基准元素**的数组元素， 在那个位置**停下。**

二、**右游标**向**左**扫描， **跨过所有大于基准元素的数组元素，**直到遇到一个**小于或等于基准元素**的数组元素，在那个位置**停下。**

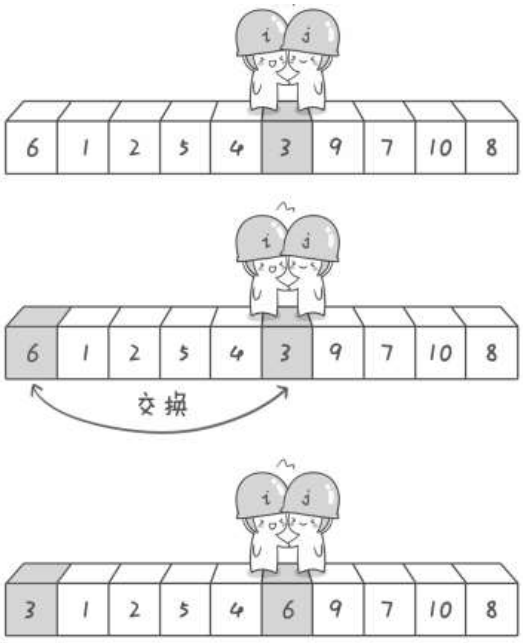
第一步：哨兵 j 先开始出动。因为此处设置的基准数是最左边的数，所以需要让哨兵 j 先开始出动，哨兵 j 一步一步的向左挪动，直到找到一个小于 6 的元素停下来。接下来，哨兵 i 再一步一步的向右挪动，直到找到一个大于 6 的元素停下来。最后哨兵 i 停在了数字 7 面前，哨兵 j 停在了数字 5 面前。



到此，第一次交换结束，接着哨兵 j 继续向左移动，它发现 4 比基准数 6 要小，那么在数字4面前停下来。哨兵 i 也接着向右移动，然后在数字 9 面前停下来，然后哨兵 i 和 哨兵 j 再次进行交换。



第二次交换结束，哨兵 j 继续向左移动，然后在数字 3 面前停下来；哨兵 i 继续向右移动，但是它发现和哨兵 j 相遇了。那么此时说明探测结束，将数字 3 和基准数字 6 进行交换，如下：



到此，第一次探测真正结束，此时已基准点 6 为分界线，6 左边的数组元素都小于等于6,6 右边的数组元素都大于等于6。

左边序列为【3,1,2,5,4】,右边序列为【9,7,10,8】。接着对于左边序列而言，以数字 3 为基准元素，重复上面的探测操作，探测完毕之后的序列为【2,1,3,5,4】；对于右边序列而言，以数字 9 位基准元素，也重复上面的探测操作。然后一步一步的划分，最后排序完全结束。

通过这一步一步的分解，我们发现快速排序的每一轮操作就是将基准数字归位，知道所有的数都归位完成，排序就结束了。

④、快速排序完整代码

```
package com.ys.high.sort;

public class QuickSort {

    //数组array中下标为i和j位置的元素进行交换
    private static void swap(int[] array , int i , int j){
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
```

```
11
12      private static void recQuickSort(int[] array,int left,int right){
13          if(right <= left){
14              return;//终止递归
15          }else{
16
17              int partition = partitionIt(array,left,right);
18              recQuickSort(array,left,partition-1);// 对上一轮排序(切分)时，基准元素左边的子数组
19              recQuickSort(array,partition+1,right);// 对上一轮排序(切分)时，基准元素右边的子数组
20          }
21      }
22
23      private static int partitionIt(int[] array,int left,int right){
24          //为什么 j加一个1，而i没有加1,是因为下面的循环判断是从--j和++i开始的.
25          //而基准元素选的array[left],即第一个元素，所以左游标从第二个元素开始比较
26          int i = left;
27          int j = right+1;
28          int pivot = array[left];// pivot 为选取的基准元素（头元素）
29          while(true){
30              while(i<right && array[++i] < pivot){}
31
32              while(j > 0 && array[--j] > pivot){}
33
34              if(i >= j){// 左右游标相遇时候停止，  所以跳出外部while循环
35                  break;
36              }else{
37                  swap(array, i, j);// 左右游标未相遇时停止，交换各自所指元素，循环继续
38              }
39          }
40          swap(array, left, j);//基准元素和游标相遇时所指元素交换，为最后一次交换
41          return j;// 一趟排序完成，  返回基准元素位置(注意这里基准元素已经交换位置了)
42      }
43
44      public static void sort(int[] array){
45          recQuickSort(array, 0, array.length-1);
46      }
47
48      //测试
49      public static void main(String[] args) {
50          //int[] array = {7,3,5,2,9,8,6,1,4,7};
51          int[] array = {9,9,8,7,6,5,4,3,2,1};
52          sort(array);
53          for(int i : array){
54              System.out.print(i+" ");
55          }
56          //打印结果为：1 2 3 4 5 6 7 7 8 9
57      }
58  }
```

⑤、优化分析

假设我们是对一个逆序数组进行排序，选取第一个元素作为基准点，即最大的元素是基准点，那么第一次循环，左游标要执行到最右边，而右游标执行一次，然后两者进行交换。这也会划分成很多的子数组。

那么怎么解决呢？理想状态下，应该选择被排序数组的中值数据作为基准，也就是说一半的数大于基准数，一般的数小于基准数，这样会使得数组被划分为两个大小相等的子数组，对快速排序来说，**拥有两个大小相等的子数组是最优的情况。**

三项取中划分

为了找到一个数组中的中值数据，一般是取数组中第一个、中间的、最后一个，选择这三个数中位于中间的数。

```
1  //取数组下标第一个数、中间的数、最后一个数的中间值
2  private static int medianOf3(int[] array,int left,int right){
3      int center = (right-left)/2+left;
```

```
    if(array[left] > array[right]){ //得到 array[left] < array[right]
        swap(array, left, right);
    }
    if(array[center] > array[right]){ //得到 array[left] array[center] < array[right]
        swap(array, center, right);
    }
    if(array[center] > array[left]){ //得到 array[center] <  array[left] < array[right]
        swap(array, center, left);
    }

    return array[left]; //array[left]的值已经被换成三数中的中位数， 将其返回
}
```

```
private static int partitionIt(int[] array,int left,int right){
    //为什么 j加一个1，而i没有加1,是因为下面的循环判断是从--j和++i开始的.
    //而基准元素选的array[left],即第一个元素，所以左游标从第二个元素开始比较
    int i = left;
    int j = right+1;
    int pivot = array[left];// pivot 为选取的基准元素（头元素）

    int size = right - left + 1;
    if(size >= 3){
        pivot = medianOf3(array, left, right); //数组范围大于3，基准元素选择中间值。
    }
    while(true){
        while(i<right && array[++i] < pivot){}

        while(j > 0 && array[--j] > pivot){}

        if(i >= j){// 左右游标相遇时候停止， 所以跳出外部while循环
            break;
        }else{
            swap(array, i, j);// 左右游标未相遇时停止，交换各自所指元素，循环继续
        }
    }
    swap(array, left, j);//基准元素和游标相遇时所指元素交换，为最后一次交换
    return j;// 一趟排序完成， 返回基准元素位置(注意这里基准元素已经交换位置了)
}
```

处理小划分

如果使用三数据取中划分方法，则必须遵循快速排序算法不能执行三个或者少于三个的数据，如果大量的子数组都小于3个，那么使用快速排序是比较耗时的。联想到前面我们讲过简单的排序（冒泡、选择、插入）。

当数组长度小于M的时候（high-low <= M）， 不进行快排，而进行插入排序。**转换参数M的最佳值和系统是相关的，一般来说， 5到15间的任意值在多数情况下都能令人满意。**

```
//插入排序
private static void insertSort(int[] array){
    for(int i = 1 ; i < array.length ; i++){
        int temp = array[i];
        int j = i;
        while(j > 0 && array[j-1] > temp){
            array[j] = array[j-1];
            j--;
        }
        array[j] = temp;
    }
}
```

分类: Java数据结构和算法

标签: Java数据结构和算法

好文要顶 关注我 收藏该文

YSOcean
关注 - 13
粉丝 - 2067
+加关注

7 0

« 上一篇：Java数据结构和算法（十五）——无权无向图
» 下一篇：Java关键字(一)——instanceof

posted @ 2018-02-15 23:04 YSOcean 阅读(5049) 评论(7) 编辑 收藏

评论列表

#1楼 2018-02-15 23:34 WilliamChang

支持一下！

支持(0) 反对(0)

#2楼 2018-02-16 00:44 小康222

有没有人和我一样读初二或者小学一起交流啊

支持(0) 反对(0)

#3楼 2018-02-18 18:24 鲸落、

厉害，收藏了

支持(0) 反对(0)

#4楼 2018-02-23 16:39 萌新啊萌新是我

老哥我都上班两天了。你啥时候回来啊

支持(0) 反对(0)

#5楼[楼主] 2018-02-24 19:27 YSOcean

@ 萌新啊萌新是我
大兄弟，早回来了

支持(0) 反对(0)

#6楼 2018-10-06 16:08 正在修炼的标准程序猴

看到了啊哈算法的图 哈哈哈

支持(0) 反对(0)

#7楼 2019-02-16 00:43 FAB4

刷完了，脍炙人口，还需要自己花时间，整理消化一下

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

相关博文：
- 算法 - 排序 - 快速排序
- 算法——高级排序——快速排序，归并排序，希尔排序
- 排序算法——选择排序
- 【排序算法】排序算法之冒泡排序
- 排序算法--快速排序

最新新闻：
- 采访Facebook产品设计师：我是如何从零开始转行成功的？
- 游戏陪练，电竞行业造血者？
- 露露事件背后是腾讯资产的流失
- 电池和续航，可能是苹果AirPods便利性的「最大受害者」
- 收购爱康国宾，阿里巴巴图什么？

» 更多新闻...