归并排序利用插入排序优化

对于归并排序的优化，除了采用一次性内存分配策略外，还可以对小规模数组采用插入排序以提高效率。
相比较而言，插入排序的原地、迭代实现的性质使得其对于小规模数组的排序更具优势。那么，一个值得思考的问题是，当子问题规模为多大时，适合采用插入排序?
考虑一个理想化的模型：有n/k个具有k个元素的列表，我们需要对每个列表采用插入排序，再利用标准合并过程完成整个排序。那么我们可以得到如下的分析：
(0) 对每个列表排序的最坏时间是Θ(k2),则n/k个列表需要Θ(nk)的渐进时间。
(1) 合并这些列表需要Θ(nlog(n/k))的时间：最初合并n/k个规模为k的列表需要cn/k * k = Θ(n)，再利用数学归纳法可证每次合并都需要Θ(n)，并且需要log(n/k)次合并。或者也可以通过递归树尽心分析。
(2) 总时间为Θ(nk+nlog(n/k))，我们可以利用这个总渐进时间，导出k取值的界

$$\Theta\left(nk+nlog_k^n\right)=\Theta\left(n\left(k+logn-logk\right)\right)$$
$$=\Theta\left(nlogn\right)\quad\Rightarrow k_{\max}=\Theta\left(logn\right)$$

由反证法可以得到，k的阶取值不能大于Θ(logn)，并且这个界可以保证插排优化的渐进时间不会慢于原始归并排序。
由于对数函数的增长特点，结合实际排序规模，k得实际取值一般在10~20间。
在归并中利用插入排序不仅可以减少递归次数，还可以减少内存分配次数（针对于原始版本）。
为了比较实际效果，我分别写了四个版本的代码，分别对应：原始版本， 插排优化， 内存分配策略优化， 内存分配策略+插排优化。并且对1000W和1亿个随机数进行了测试，得到了如下结果

	A	B	C	D	E	F
		次数	原始归并	插排优化	一次性内存分配策略	一次性策略+插排优化
	数据规模1000W					
		1	41	4	3	2
		2	42	4	3	2
		3	42	5	2	2
		4	41	4	3	2
		5	41	4	3	2
		AVG	41.4	4.2	2.8	2
	数据规模1Y					
		1	N/A	61	28	23
		2	N/A	61	28	23
		3	N/A	63	28	24
		4	N/A	61	28	23
		5	N/A	61	28	23
		AVG	N/A	61.4	28	23.2

考虑到数据规模，插排优化的k的取值为20。N/A表示未进行测试。

对于1000W的规模，优化版本的时间均可以控制在5S内，而原始版本需要40S+。并且优化内存分配策略的版本效率比起插排优化有微弱的优势。
对于1亿的数据规模（我放弃了测试原版性能，因为时间真的太长了...），内存分配的优化比起插入排序要更加明显，而结合二者的优化也只比前者快了几秒。
另外，在1亿的数据规模测试中，我试探性地把k从20调到了25，发现对于同时采用插排优化和内存优化测了的版本基本上只快了1S。
所以可以预见的是，对于更大规模的数据，动态内存分配是一个很大的瓶颈，我们可以稍稍计算下n个元素的合并需要多少次内存分配。
利用归纳可以很容易的算出：

$$
\begin{aligned}
c &= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 2 + 1 \\
&= n\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + n\right) \\
&= n - 1
\end{aligned}
$$

也就是说，原始的归并排序对于n个元素需要Θ(n)的分配。这个瓶颈是很明显的。
上面的测试比起科学严谨还差一些，不过还是能够说明一些问题的。

```
package xwq.sort;
import xwq.util.StdIn;
import xwq.util.StdOut;

/**
 * 优化后的归并排序
 * 使用插入排序优化
 * copy操作由原来的循环复制改为使用
 * java API:System.arraycopy(Object src,int srcPos,Object dest,int destPos,int length)
 */
public class MergeSortX {

    private static int CUTOFF = 10;//插入排序临界值
```

```
    public static void sort(Comparable[] a) {
        Comparable[] acopy = new Comparable[a.length];
        sort(a,acopy,0,a.length-1);
    }

    private static void sort(Comparable[] a,Comparable[] acopy,int low,int high) {
        if(low >= high) return;

        //插入排序优化
        if((high-low+1) <= CUTOFF) {
            insertSort(a,low,high);
            return;
        }

        int mid = (low+high)/2;
        sort(a,acopy,low,mid);
        sort(a,acopy,mid+1,high);
        merge(a,acopy,low,mid,high);
    }

    //合并两个已经递增有序的数组a[low,mid]，a[low+1,high]
    private static void merge(Comparable[] a,Comparable[] acopy,int low,int mid,int
high) {

        //for(int i = low;i<=high;i++)
        //  acopy[i] = a[i];
        //copy待排序部分数组到辅助数组
        //System.arraycopy(Object src,int srcPos,Object dest,int destPos,int length)
        //使用System.arraycopy快于使用循环复制
        System.arraycopy(a, low, acopy, low, high-low+1);

        //左半待合并数组a[low,mid],右半待合并数组a[low+1,high]
        int i = low; //待合并左半部分数组起点位置
        int j = mid+1;//待合并右半部分数组起点位置
        int k = low; //合并数组起点位置
```

```
    while(k<=high) {
        //左半数组已全部插入
        if(i>mid) a[k++] = acopy[j++];
        //右半数组已全部插入
        else if(j>high)  a[k++] = acopy[i++];
        //左半数组目前所指向的元素值<右半数组所指向的元素值
        else if(less(acopy[i],acopy[j])) a[k++] = acopy[i++];
        //左半数组目前所指向的元素值>=右半数组所指向的元素值
        else a[k++] = acopy[j++];
    }
}

//插入排序
private static void insertSort(Comparable[] a,int low, int high) {
    for(int i = low+1;i<=high;i++) {
        Comparable insert = a[i];
        int pos = i-1;
        while(pos>=low && less(insert,a[pos])) {
            a[pos+1] = a[pos];
            pos--;
        }
        a[pos+1] = insert;
    }
}

//v<w
private static boolean less(Comparable v,Comparable w) {
    return v.compareTo(w) < 0;
}

//输出排序数组
public static void print(Comparable a[]) {
    for(int i = 0;i<a.length;i++)
        StdOut.print(a[i]+" ");
```

```
    }

    //测试函数
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        sort(a);
        print(a);
    }

}
```

作者：清文
来源：CSDN
原文：https://blog.csdn.net/qing0706/article/details/50560679
版权声明：本文为博主原创文章，转载请附上博文链接！