

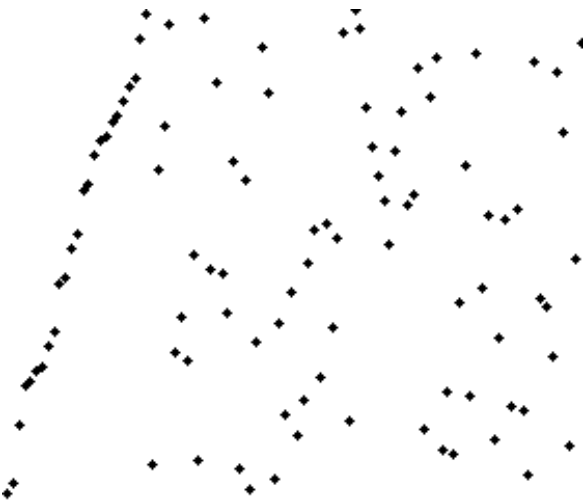
浅谈算法和数据结构: 三 合并排序

合并排序，顾名思义，就是通过将两个有序的序列合并为一个大的有序的序列的方式来实现排序。合并排序是一种典型的分治算法：首先将序列分为两部分，然后对每一部分进行循环递归的排序，然后逐个将结果进行合并。

input	M	E	R	G	E	S	O	R	T	E
sort left half	E	E	G	M	O	R	R	S	T	E
sort right half	E	E	G	M	O	R	R	S	A	E
merge results	A	E	E	E	E	G	L	M	M	O

合并排序最大的优点是它的时间复杂度为 $O(n \lg n)$ ，这个是我们之前的选择排序和插入排序所达不到的。他还是一种稳定性排序，也就是相等的元素在序列中的相对位置在排序前后不会发生变化。他的唯一缺点是，需要利用额外的 N 的空间来进行排序。

一 原理



合并排序依赖于合并操作，即将两个已经排序的序列合并成一个序列，具体的过程如下：

公告

访问统计:

81446198

昵称: yangecn
u

园龄: 7年5
个月

荣誉: 推荐博
客

粉丝: 2033

关注: 33

+加关注

< 2017年3月 >

日 一 二 三 四 五 六

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 7 8

随笔分类

Data
Structures &
Algorithms(
12)
Database(5)
Design
Patterns(8)
DotNet
Framework(

- 1. 申请空间，使其大小为两个已经排序序列之和，然后将待排序数组复制到该数组中。
- 2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置
- 3. 比较复制数组中两个指针所指向的元素，选择相对小的元素放入到原始待排序数组中，并移动指针到下一位置
- 4. 重复步骤3直到某一指针达到序列尾
- 5. 将另一序列剩下的所有元素直接复制到原始数组末尾

该过程实现如下，注释比较清楚：

```
private static void Merge(T[] array, int lo, int mid,
int hi)
{
    int i = lo, j = mid + 1;
    //把元素拷贝到辅助数组中
    for (int k = lo; k <= hi; k++)
    {
        aux[k] = array[k];
    }
    //然后按照规则将数据从辅助数组中拷贝回原始的array中
    for (int k = lo; k <= hi; k++)
    {
        //如果左边元素没了， 直接将右边的剩余元素都合并到到原
        数组中
        if (i > mid)
        {
            array[k] = aux[j++];
        }//如果右边元素没有了， 直接将所有左边剩余元素都合并到
        原数组中
        else if (j > hi)
        {
            array[k] = aux[i++];
        }//如果左边右边小， 则将左边的元素拷贝到原数组中
        else if (aux[i].CompareTo(aux[j]) < 0)
        {
            array[k] = aux[i++];
        }
        else
        {
            array[k] = aux[j++];
        }
    }
}
```

下图是使用以上方法将E E G M R和A C E R T这两个有序序列合并为一个大的序列的过程演示：

26)
Excel
Developmen
t(11)
Kinect for
Windows
SDK(29)
Reactive
Extensions(
7)
Some
Reading
Books(1)

随笔档案

- 2016年5月
(2)
- 2015年11月
(1)
- 2015年5月
(1)
- 2015年4月
(1)
- 2015年3月
(1)
- 2015年2月
(1)
- 2015年1月
(1)
- 2014年12月
(1)
- 2014年11月
(3)
- 2014年10月
(1)
- 2014年9月
(1)
- 2014年8月
(3)

		a[]														
	k	0	1	2	3	4	5	6	7	8	9	i	j	0	1	2
Input copy		E	E	G	M	R	A	C	E	R	T			-	-	-
		E	E	G	M	R	A	C	E	R	T			E	E	G
												0	5			
	0	A										0	6	E	E	G
	1	A	C									0	7	E	E	G
	2	A	C	E								1	7	E	E	G
	3	A	C	E	E							2	7		E	G
	4	A	C	E	E	E						2	8			G
	5	A	C	E	E	E	G					3	8			G
	6	A	C	E	E	E	G	M				4	8			
	7	A	C	E	E	E	G	M	R			5	8			
	8	A	C	E	E	E	G	M	R	R		5	9			
	9	A	C	E	E	E	G	M	R	R	T	6	10			
merged result		A	C	E	E	E	G	M	R	R	T					

二 实现

合并排序有两种实现，一种是至上而下(Top-Down)合并，一种是至下而上 (Bottom-Up)合并，两者算法思想差不多，这里仅介绍至上而下的合并排序。

至上而下的合并是一种典型的分治算法(Divide-and-Conquer)，如果两个序列已经排好序了，那么采用合并算法，将这两个序列合并为一个大的序列也就是对大的序列进行了排序。

首先我们将待排序的元素均分为左右两个序列，然后分别对其进去排序，然后对这个排好序的序列进行合并，代码如下：

```
public class MergeSort<T> where T : IComparable<T>
{
    private static T[] aux; // 用于排序的辅助数组
    public static void Sort(T[] array)
    {
        aux = new T[array.Length]; // 仅分配一次
        Sort(array, 0, array.Length - 1);
    }
    private static void Sort(T[] array, int lo, int
hi)
    {
        if (lo >= hi) return; //如果下标大于上标，则返回
        int mid = lo + (hi - lo) / 2; //平分数组
        Sort(array, lo, mid); //循环对左侧元素排序
        Sort(array, mid + 1, hi); //循环对右侧元素排序
        Merge(array, lo, mid, hi); //对左右排好的序列进行
合并
    }
    ...
}
```

以排序一个具有15个元素的数组为例，其调用堆栈为：

- 2014年7月
- (2)
- 2014年6月
- (1)
- 2014年5月
- (7)
- 2014年4月
- (2)
- 2014年3月
- (6)
- 2014年2月
- (1)
- 2014年1月
- (6)
- 2013年12月
- (3)
- 2013年11月
- (1)
- 2013年10月
- (2)
- 2013年9月
- (2)
- 2013年8月
- (4)
- 2013年7月
- (4)
- 2013年6月
- (2)
- 2013年5月
- (1)
- 2013年4月
- (2)
- 2013年3月
- (1)
- 2013年2月
- (2)
- 2013年1月
- (1)
- 2012年12月
- (2)

```

sort(a, 0, 15)
  sort left half sort(a, 0, 7)
    sort(a, 0, 3)
      sort(a, 0, 1)
        merge(a, 0, 0, 1)
      sort(a, 2, 3)
        merge(a, 2, 2, 3)
      merge(a, 0, 1, 3)
    sort(a, 4, 7)
      sort(a, 4, 5)
        merge(a, 4, 4, 5)
      sort(a, 6, 7)
        merge(a, 6, 6, 7)
      merge(a, 4, 5, 7)
    merge(a, 0, 3, 7)
  sort right half sort(a, 8, 15)
    sort(a, 8, 9)
      merge(a, 8, 8, 9)
    sort(a, 10, 11)
      merge(a, 10, 10, 11)
    merge(a, 8, 9, 11)
  sort(a, 12, 15)
    sort(a, 12, 13)
      merge(a, 12, 12, 13)
    sort(a, 14, 15)
      merge(a, 14, 14, 15)
    merge(a, 12, 13, 15)
  merge(a, 8, 11, 15)
merge results merge(a, 0, 7, 15)

```

Top-down mergesort call trace

我们单独将Merge步骤拿出来，可以看到合并的过程如下：

	lo	hi	0	1	2	3	4	5	6	7	8
			M	E	R	G	E	S	O	R	T
merge(a, 0, 0, 1)	0	1	E	M	R	G	E	S	O	R	T
merge(a, 2, 2, 3)	2	3	E	M	G	R	E	S	O	R	T
merge(a, 0, 1, 3)	0	3	E	G	M	R	E	S	O	R	T
merge(a, 4, 4, 5)	4	5	E	G	M	R	E	S	O	R	T
merge(a, 6, 6, 7)	6	7	E	G	M	R	E	S	O	R	T
merge(a, 4, 5, 7)	4	7	E	G	M	R	E	O	R	S	T
merge(a, 0, 3, 7)	0	7	E	E	G	M	O	R	R	S	T
merge(a, 8, 8, 9)	8	9	E	E	G	M	O	R	R	S	E
merge(a, 10, 10, 11)	10	11	E	E	G	M	O	R	R	S	E
merge(a, 8, 9, 11)	8	11	E	E	G	M	O	R	R	S	A
merge(a, 12, 12, 13)	12	13	E	E	G	M	O	R	R	S	A
merge(a, 14, 14, 15)	14	15	E	E	G	M	O	R	R	S	A
merge(a, 12, 13, 15)	12	15	E	E	G	M	O	R	R	S	A
merge(a, 8, 11, 15)	8	15	E	E	G	M	O	R	R	S	A
merge(a, 0, 7, 15)	0	15	A	E	E	E	E	G	L	M	M

Trace of merge results for top-down mergesort

2012年11月
(3)
2012年10月
(2)
2012年9月
(1)
2012年8月
(1)
2012年7月
(2)
2012年6月
(2)
2012年5月
(7)
2012年4月
(11)
2012年3月
(4)
2011年7月
(3)

积分与排名

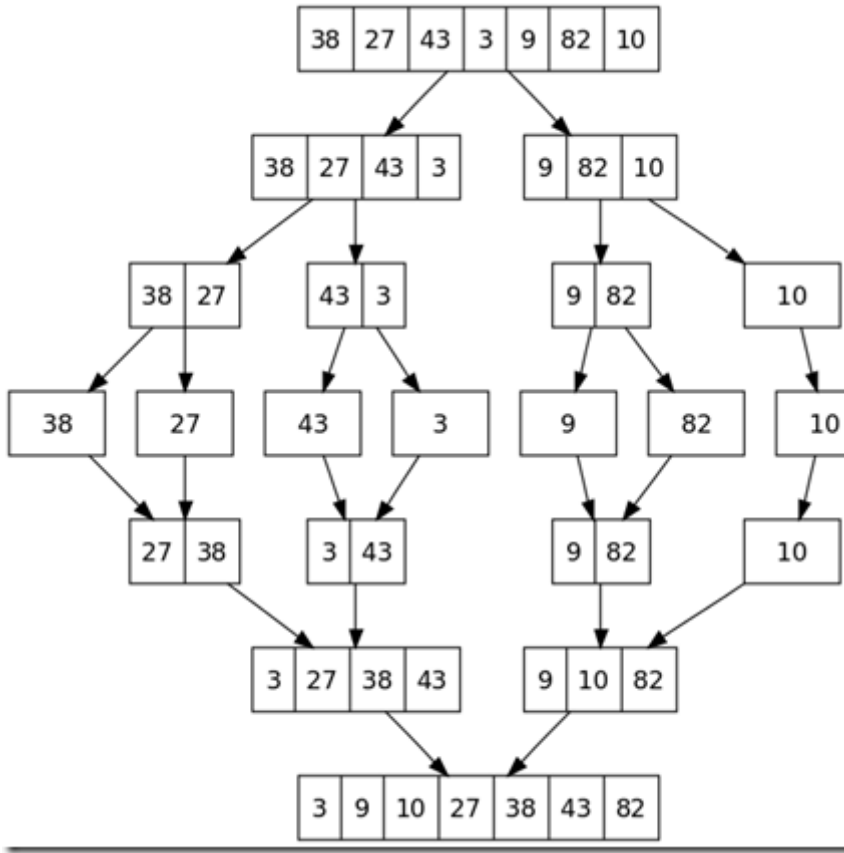
积分 -
313940
排名 - 448

阅读排行榜

1. 浅谈依赖注入(54510)
2. [译]Kinect for Windows SDK开发入门(一)：开发环境配置(53981)

三 图示及动画

如果以排序38,27,43,3,9,82,10为例，将合并排序画出来的话，可以看到如下图：

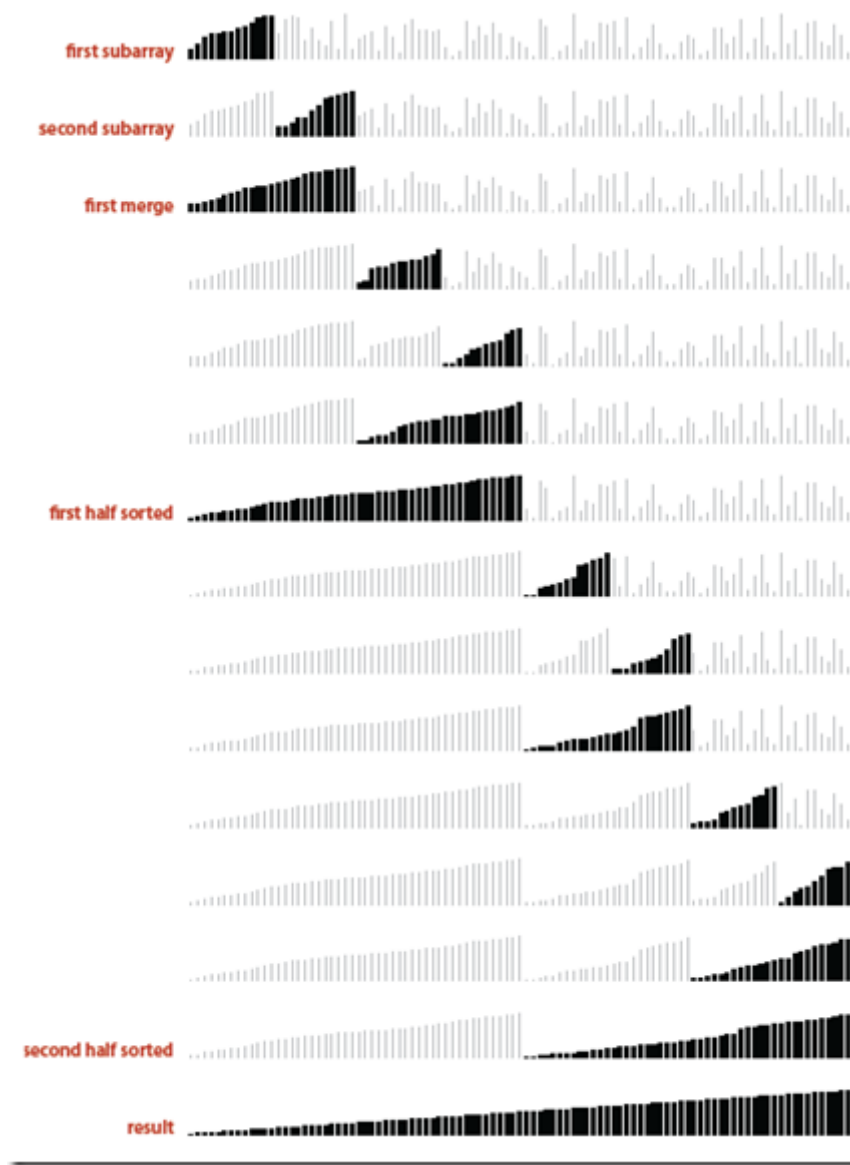


下图是合并排序的可视化效果图：

- 3. [\[译\]Kinect for Windows SDK开发入门 \(二\)：基础知识上 \(34012\)](#)
- 4. [浅谈Excel 开发：— Excel 开发概述\(33470\)](#)
- 5. [浅谈命令 查询职责分离 \(CQRS\)模式 \(30839\)](#)

推荐排行榜

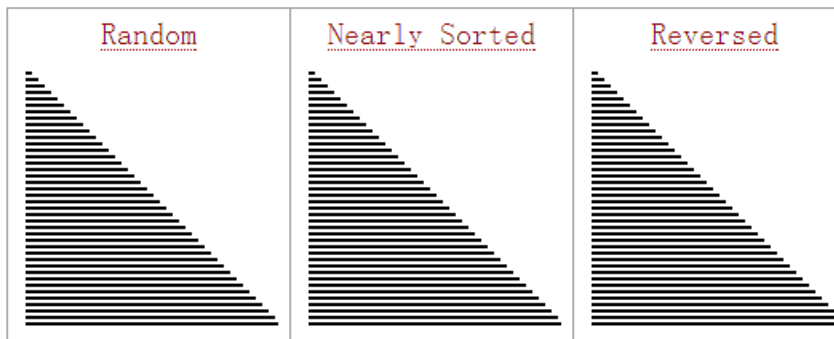
- 1. [C# 中参数 验证方式的演变\(118\)](#)
- 2. [.NET 环境 中使用 RabbitMQ\(78\)](#)
- 3. [减少.NET 应用程序内存 占用的一则实 践\(75\)](#)
- 4. [浅谈并发 与并行\(一\) \(72\)](#)
- 5. [浅谈依赖 注入\(68\)](#)



对6 5 3 1 8 7 2 4 进行合并排序的动画效果如下：

6 5 3 1 8 7 2 4

下图演示了合并排序在不同的情况下的效率：



四 分析

1. 合并排序的平均时间复杂度为 $O(n \lg n)$

证明：合并排序是目前我们遇到的第一个时间复杂度不为 n^2 的时间复杂度为 $n \lg n$ (这里 $\lg n$ 代表 $\log_2 n$)的排序算法，下面给出对合并排序的时间复杂度分析的证明：

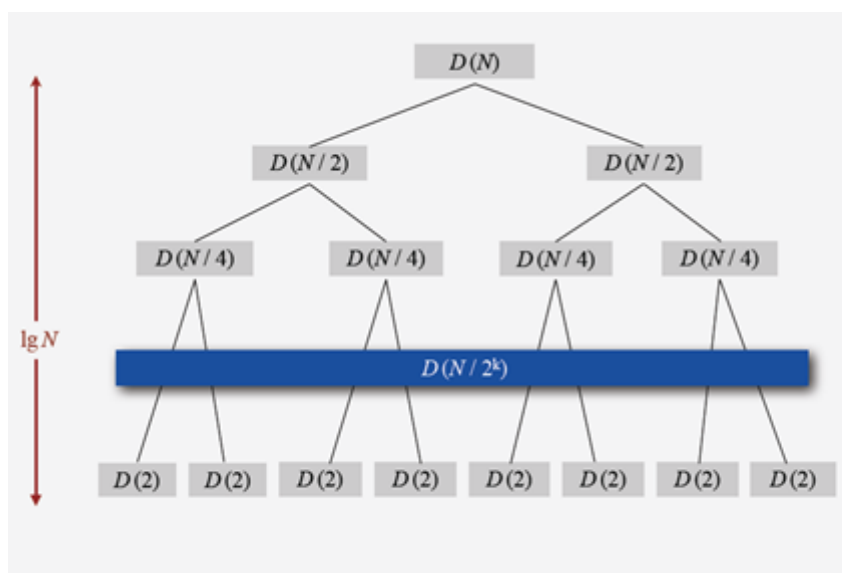
假设 $D(N)$ 为对整个序列进行合并排序所用的时间，那么一个合并排序又可以二分为两个 $D(N/2)$ 进行排序，再加上与 N 相关的比较和计算中间数所用的时间。整个合并排序可以用如下递归式表示：

$$D(N) = 2D(N/2) + N, N > 1;$$

$$D(N) = 0, N = 1; \text{ (当 } N=1 \text{ 时，数组只有1个元素，已排好序，时间为0)}$$

因为在分治算法中经常会用到递归式，所以在CLRS中有一章专门讲解递归式的求解和证明，使用主定理(master theorem)可以直接求出该递归式的值，后面我会简单介绍。这里简单的列举两种证明该递归式时间复杂度为 $O(n \lg n)$ 的方法：

Prof1：处于方便性考虑，我们假设数组 N 为2的整数幂，这样根据递归式我们可以画出一棵树：



可以看到我们对数组N进行MergeSort的时候，是逐级划分的，这样就形成了一个**满二叉树**，树的每一及子节点都为N，树的深度即为层数 $\lg N + 1$ ，满二叉树的深度的计算可以查阅相关资料，上图中最后一层子节点没有画出来。这样，这棵树有 $\lg N + 1$ 层，每一层有N个节点，所以

$$D(N) = (\lg N + 1)N = N \lg N + N = N \lg N$$

Prof2: 我们在为递归表达式求解的时候，还有一种常用的方法就是数学归纳法，

首先根据我们的递归表达式的初始值以及观察，我们猜想 $D(N) = N \lg N$ 。

1. 当 $N=1$ 时， $D(1)=0$,满足初始条件。
2. 为便于推导，假设N是2的整数次幂 $N=2^k$ ，即 $D(2^k)=2^k \lg 2^k = k \cdot 2^k$
3. 在 $N+1$ 的情况下 $D(N+1)=D(2^{k+1})=2^{k+1} \lg 2^{k+1}=(k+1) \cdot 2^{k+1}$,所以假设成立， $D(N)=N \lg N$ 。

2. 合并排序需要额外的长度为N的辅助空间来完成排序

如果对长度为N的序列进行排序需要 $\leq c \log N$ 的额外空间，认为就是**就地排序**(in place排序)也就是完成该排序操作需要较小的，固定数量的额外辅助内存空间。之前学习过的选择排序，插入排序，希尔排序都是原地排序。

但是在合并排序中，我们要创建一个大小为N的辅助排序数组来存放初始的数组或者存放合并好的数组，所以需要长度为N的额外辅助空间。当然也有前人已经将合并排序改造为了**就地合并排序**，但是算法的实现变得比较复杂。

需要额外N的空间来辅助排序是合并排序的最大缺点，如果在内存比较关心的环境中可能需要采用其他算法。

五 几点改进

对合并排序进行一些改进可以提高合并排序的效率。

1. 当划分到较小的子序列时，通常可以使用插入排序替代合并排序

对于较小的子序列（通常序列元素个数为7个左右），我们就可以采用插入排序直接进行排序而不用继续递归了），算法改造如下：

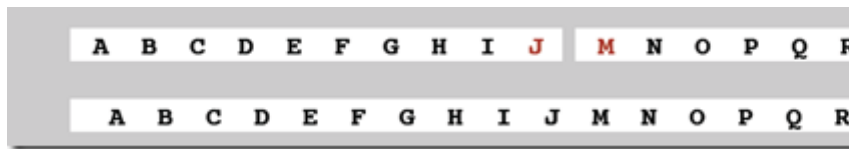
```
private const int CUTOFF = 7; //采用插入排序的阈值
private static void Sort(T[] array, int lo, int hi)
{
    if (lo >= hi) return; //如果下标大于上标，则返回
    if (hi - lo <= CUTOFF - 1)
        Sort<T>.SelectionSort(array, lo, hi);
    int mid = lo + (hi - lo) / 2; //平分数组
```



```
Sort(array, lo, mid); //循环对左侧元素排序
Sort(array, mid + 1, hi); //循环对右侧元素排序
Merge(array, lo, mid, hi); //对左右排好的序列进行合并
}
```

2. 如果已经排好序了就不用合并了

当已排好序的左侧的序列的最大值 \leq 右侧序列的最小值的时候，表示整个序列已经排好序了。



算法改动如下：

```
private static void Sort(T[] array, int lo, int hi)
{
    if (lo >= hi) return; //如果下标大于上标, 则返回
    if (hi - lo <= CUTOFF - 1)
        Sort<T>.SelectionSort(array, lo, hi);
    int mid = lo + (hi - lo) / 2; //平分数组
    Sort(array, lo, mid); //循环对左侧元素排序
    Sort(array, mid + 1, hi); //循环对右侧元素排序
    if (array[mid].CompareTo(array[mid + 1]) <= 0)
        return;
    Merge(array, lo, mid, hi); //对左右排好的序列进行合并
}
```

3. 并行化

分治算法通常比较容易进行并行化，在[浅谈并发与并行](#)这篇文章中已经展示了如何对快速排序进行并行化（快速排序在下一篇文章中讲解），合并排序一样，因为我们均分的左右两侧的序列是独立的，所以可以进行并行，值得注意的是，并行化也有一个阈值，当序列长度小于某个阈值的时候，停止并行化能够提高效率，这些详细的讨论在[浅谈并发与并行](#)这篇文章中有详细的介绍了，这里不再赘述。

六 用途

合并排序和快速排序一样都是时间复杂度为 $n\lg n$ 的算法，但是和快速排序相比，合并排序是一种稳定性排序，也就是说排序关键字相等的两个元素在整个序列排序的前后，相对位置不会发生变化，这一特性使得合并排序是稳定性排序中效率最高的一个。在Java中对引用对象进行排序，Perl、C++、Python的稳定性排序的内部实现中，都是使用的合并排序。

七 结语

本文介绍了分治算法中比较典型的一个合并排序算法，这也是我们遇到的第一个时间复杂度为 $n\lg n$ 的排序算法，并简要对算法的复杂度

进行的分析，希望本文对您理解合并排序有所帮助，下文将介绍快速排序算法。



作者: [yangecnu](#) ([yangecnu's Blog on 博客园](#))
出处: <http://www.cnblogs.com/yangecnu/>
本作品由yangecnu 创作，采用[知识共享署名-非商业性使用-禁止演绎 2.5 中国大陆许可协议](#)进行许可。欢迎转载，但任何转载必须保留完整文章，在显要地方显示署名以及原文链接。如您有任何疑问或者授权方面的协商，请 [给我留言](#)。

分类: [Data Structures & Algorithms](#)

标签: [Merge Sort](#), [合并排序](#)

好文要顶

关注我

收藏该文



[yangecnu](#)
关注 - 33
粉丝 - 2033

荣誉: [推荐博客](#)
[+加关注](#)

16

推荐

0

反对

« 上一篇: [浅谈算法和数据结构: 二 基本排序算法](#)

» 下一篇: [浅谈Excel开发: 十一 针对64位Excel的插件的开发和部署](#)

posted @ 2014-01-27 06:40 [yangecnu](#) 阅读(15103) 评论(7) [编辑](#) [收藏](#)

评论列表

#1楼 2014-01-27 08:36 [LionJoy](#)

膜拜

[支持\(1\)](#) [反对\(0\)](#)

#2楼 2014-01-27 14:06 [牲口TT](#)

最近要过年了，比较闲，也打算写几篇关于排序的文章，刚刚着手，但是当我看到你的文章的时候，自己感觉好羞愧啊，都没有写下去的勇气了。太膜拜你了。能请教一下你的动态图都是用什么画的？

[支持\(2\)](#) [反对\(0\)](#)

#3楼 2014-01-27 16:03 [badnewfish](#) 

@ 牲口TT

一般判断好文章的标准就是：有没有求工具的，求图的，球背景音乐的！

[支持\(1\)](#) [反对\(0\)](#)

#4楼 2014-01-27 16:19 [牲口TT](#) 


@ badnewfish

引用

@牲口TT一般判断好文章的标准就是：有没有求工具的，求图的，球背景音乐的！

作者的文章布局，插图等都太漂亮了。

[支持\(1\)](#) [反对\(0\)](#)


#5楼[楼主] 2014-01-27 16:38 [yangeclu](#) 

@ badnewfish

@牲口TT

一部分动画是来自[wikipedia](#)，一部分来自[sorting-algorithms](#)这个网站的屏幕截图。其他图片来自[Algorithms, 4th Edition](#)这本书。

[支持\(1\)](#) [反对\(0\)](#)

#6楼 2015-07-20 09:11 [sourcecode_2345](#) 

谢谢楼主分享，讲得非常详细，数据结构源

码：<http://algorithm.eyesourcecode.com/>

[支持\(1\)](#) [反对\(0\)](#)

#7楼 2015-12-23 22:38 [91深蓝](#) 

感谢有你这个大神在！

[支持\(1\)](#) [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。

最新IT新闻：

- 乐视电视最大代工厂：库存一直较少 目前没拖欠付款
 - 联通千亿帝国危机：净利暴跌96% 取消漫游费损失63亿
 - 三星语音助手“Bixby”被指命名糟糕 全世界都很难发音
 - 数据：Windows XP份额加起来比8和Vista总和还多
 - 苹果故意不让Workflow支持谷歌？这是造谣
- » [更多新闻...](#)

最新知识库文章：

- [程序员学习能力提升三要素](#)
- [为什么我要写自己的框架?](#)
- [垃圾回收原来是这么回事](#)
- [「代码家」的学习过程和学习经验分享](#)
- [写给未来的程序媛](#)
- » [更多知识库文章...](#)