哈尔滨工业大学

数据科学与大数据技术专业

实验报告

课程名称：计算机算法基础

实验项目：分治与排序

实验题目：归并排序与快速排序算法实现比较

实验日期：2019.11.08

班级：1534701

学号姓名：1153470114 薛博洋

1153470104 沈孙乐

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

**一、实验目的**

排序方法的实验比较。排序方法是数据处理的最基本和最重要的操作。其目的是将一组“无序”的记录序列调整为“有序”的记录序列。

**二、实验要求及实验环境**

**（1）实验要求**

1. 实现归并排序的递归和非递归算法。

2. 实现快速排序算法。划分方法要求：首元素划、三数取中划分法、随机划分

法、“不同取大”划分法。

3. 实验方法

1、生成不同规模的数据，分别对正序、反序和随机的初始序列进行排序，

考察算法对初始序列的敏感程度。

2、采用不同的划分基准的选取方法，分别对对正序、反序和随机的初始

序列进行排序，考察划分基准的选取对算法时间性能的影响。

**（2）实验环境**

Windows 10系统 C语言 codebooks

**三、设计的基本思想和算法的原理**

（1）归并排序递归版本

1、对于一个长度为n的数组，先进行分解，取中间数mid = （high+low）/2，将一个较长数组分解为两个大致相同长度的小数组。

2、再用递归的方式进行归并。递归的终止返回为n = 2 的情况，此时对于数组的这个部分来说，两个分部只有一个元素，自然有序，因此对二者进行比较合并为一个有序数组。

对于递归的一般情况：在归并的时候，从每一个元素的起始处取一个元素，选择其中较小的取出进行排列，一直进行直到有一方的数组元素取尽，则将剩余元素并上。

最终成为一个完整的有序数列。

（2）归并排序非递归版本

归并排序的思想就是数组拆分成小组，然后将它进行合并排序，因此我们可以人为地定义拆分的方式，直接用循环的方式归并排序。

1. 对数组进行拆分，拆分分为若干层，层数等于log2(length)+1。从最底层的两个一对进行拆分，直到最上层进行2^ (log2(length)+1)为最大长度的拆分。
2. 每一层对拆分出来的组内进行排序，因为都是2 的倍数所以一定可以排序，而每一层因为上一层中已经对下一层的那组的前半部分进行了排序，所以在组内元素大于2的时候可以应用归并的方法，顺次对前半后半的首元素进行比较归并。

每一层都可能会有一些多余的元素，将它与剩余的半组进行归并排序，如果剩余的不足半组，则不需要进行操作，下一层会进行排序。

在递归的最后一层，则直接将剩余的两个部分进行归并排序，得到最后的完整序列。

**四、实验结果与分析**

（1）所有实验结果见“实验结果汇总表”。

（2）归并排序分析

归并排序的时间复杂度为n\*log(n)。所以当数据量不同时，设一个数据为m，另一个数为km,其运算时间的比较为：

kmlog(km) / mlog(m) = k \* (1 + log(k)/log(m))

所以当数据量增加时，运算时间之比应该趋近于数据量的比值，所以有：

（2.1）算法实际增长速度，及其原因分析：

因为数据量比较大，所以所有数据都逐渐趋向于它们之间的比值，但是又不等于比值。以归并排序非递归算法为例：

它们之间的比值都趋近于1.116，虽然没能接近于它们应有的比值：1.25-1.167之间，但是却绝不是复杂度为O(n)级别的情况。因为O(n)情况下由于额外的运算需求，它们之间的比值将会小于n。

（2.3）复杂性相当的排序快慢比较：

在相同数量级别的数据下，递归排序的速度比非递归排序的速度慢。首先是非递归操作不需要进行分类的计算，所以节约了许多时间，另外递归操作需要更多的内存资源，可能对性能也有一定的影响。

（2.4）原始数据对于排序的影响：

随机数列的排序速度总是小于有序数组的排序速度，因为它的交换操作将会多于后两种数组的情况。

（2.5）结论：1、递归的归并操作效率更低，是由于相对非递归操作有更多的分解计算操作。2、原始数据的情况好坏对排序效率具有一定的影响，影响了它的操作数目。3、二者的计算效率都是nlog(n)

（3）快速排序分析

**五、经验体会与不足**

1、对于迭代和递归的理解不够深刻，在将递归转换为迭代的时候出现问题，在归并排序的非递归算法时在分类条件和运算方面没有系统地想清楚，造成了一些问题。

2、对于c语言不够熟悉，在创造数据的时候遇到了困难，尤其是当数据增加到百万级别的时候，堆栈的溢出成为了创建数据的一个难点。虽然不是算法的内容，但是在实验中还是造成了不小的麻烦。

3、

**六、附录：源代码（带注释）**

归并排序：

//内部函数merge

void merge(int a[], int b[], int low, int mid, int high){

int i = low;

int k = low;

int j = mid + 1;

//一般情况下，从中点和起点开始对数组进行归并

while((i <= mid) && (j <= high)){

if(a[i] < a[j]){

b[k] = a[i];

k++;

i++;

}else{

b[k] = a[j];

k++;

j++;

}

}

//归并余下的剩余情况

while(j <= high){

b[k] = a[j];

k++;

j++;

}

while(i <= mid){

b[k] = a[i];

k++;

i++;

}

//将排好序列的数组放入原来的数组中

for(int i = low; i < high + 1; i++){

a[i] = b[i];

}

}

//递归的归并排序

void mergeSortRe(int a1[], int b1[], int low, int high){

if(high - low > 0){

mergeSortRe(a1, b1, low, (low+high)/2);

mergeSortRe(a1, b1, (low+high)/2 + 1, high);

merge(a1, b1, low, (low+high)/2, high);

}

}

//一个中间需要的函数

int middle(int i, int j){

return (i + j) / 2;

}

//非递归的归并排序

void mergeSort(int a[], int b[], int length){

//计算出需要归并的有多少层

int hei = log2(length) + 1;

for(int i = 1; i < hei ; i++){

//每层按照buf的大小来进行归并

int buf = (int)pow(2, i);

for(int j = 0; j < length / buf; j++){

merge(a, b, j \* buf, middle(j \* buf, (j+1) \* buf - 1), (j+1) \* buf - 1);

//每一层的末尾剩余处理。

if(((length % buf) > (buf / 2)) && (j == length / buf - 1)){

merge(a, b, buf \* (j + 1), (j + 1) \* buf + buf / 2 - 1, length - 1);

}

}

//整个循环末尾的处理

if(i == hei - 1){

merge(a, b , 0, buf-1, length - 1);

}

}

}