**2020-2021第二学期**

**网络工程专业2018级**

**《算法数据与分析》课程设计**

**班级：网络工程**

**学号：181124039**

**姓名：张怀周**

**排序算法问题**

1. **问题描述**

排序算法：执行一系列基准测试来比较归并排序和快速排序哪个执行得更快。基准测试里面应该包含“随机”序列和“几乎”已排序的序列。

要求：

1. 运用所学设计至少一种算法，并给出运行结果；
2. 对所设计算法的时空效率进行分析；
3. 写出你的心得与体会；
4. **算法分析**

快速排序：快速排序的交换时跳跃式的，每次排序的时候设置一个基准点，将小于等于基准点的数全部放到基准点的左边，将大于等于基准点的数全部放到基准点的右边。

归并排序：先把数组分为两个子数组，一直递归把数组划分为两个子数组，直到数组里只有一个元素，这时候才开始排序，让两个数组间排好序，依次按照递归的返回来把两个数组进行排好序，到最后就可以把整个数组排好序。

1. **算法描述**

快速排序：

1、设置两个变量 low、high，排序开始时：low=0，high=size-1。

2、整个数组找基准正确位置，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面

默认数组的第一个数为基准数据，赋值给temp，即temp=[begin]。

因为默认数组的第一个数为基准，所以从后面开始向前搜索（end–），找到第一个小于key的arr[end]，就将 arr[end] 赋给 arr[begin]，即 arr[begin] = arr[end]。（循环条件是 arr[end] >= temp；结束时 arr[end] < temp）

此时从前面开始向后搜索（begin ++），找到第一个大于temp的arr[begin]，就将 arr[begin] 赋给 arr[end]，即 arr[end] = arr[begin]。（循环条件是 array[low] <= key；结束时 arr[begin] > temp）

循环 2-3 步骤，直到 begin =high，该位置就是基准位置。

把基准数据赋给当前位置。

3、第一趟找到的基准位置，作为下一趟的分界点。

4、递归调用重复2、3、4的步骤。

5、最终就会得到排序好的数组。

归并排序：

归并排序的基本思路就是将数组分成二组A，B，如果这二组组内的数据都是有序的，那么就可以很方便的将这二组数据进行排序。可以将A，B组各自再分成二组。依次类推，当分出来的小组只有一个数据时，可以认为这个小组组内已经达到了有序，然后再合并相邻的二个小组就可以了。这样通过先递归的分解数列，再合并数列就完成了归并排序。

4. **算法实现**

#include<cstdio>

#include<Windows.h>

#include<ctime>

#include<cstdlib>

#include<cstring>

#include <math.h>

#include<iostream>

using namespace std;

const int maxn = 1000000;

int n;

int a[maxn]; //a[]为随机产生的数组

int b[maxn], t[maxn]; //b[]复制随机数组,t[]用于归并排序,暂时保存a[]

//快速排序

void quickSort(int \*arr,int begin,int end)

{

if(begin < end)//如果区间不只一个数

{

int temp = arr[begin];

int i = begin;

int j = end;

while(i < j)//不重复遍历

{

while(i<j && arr[j] > temp)

j--;

arr[i] = arr[j];

while(i<j && arr[i] <= temp)

i++;

arr[j] = arr[i];

}

arr[i] = temp;

quickSort(arr,begin,i-1);//对基准元素的左边子区间进行相似的快速排序

quickSort(arr,i+1,end);//对基准元素的右边子区间进行相似的快速排序

}

//如果区间只有一个数，则返回

else

return;

}

//归并排序

void merge\_sort(int \*a, int x, int y, int \*t)

{

if(y-x > 1)

{

int m = (y-x)/2 + x; //取中间

int p = x, q = m, i = x;

merge\_sort(b, p, m, t); //两边分别排序

merge\_sort(b, m, y, t);

while(p < m && q < y)

{

if(b[p] <= b[q]) t[i++] = b[p++];

else t[i++] = b[q++];

}

while(p < m) t[i++] = b[p++];

while(q < y) t[i++] = b[q++];

for(i = x; i < y; i++) //将排序好的部分重新存入数组 a[]

b[i] = t[i];

}

}

int main()

{

printf("随机产生n个数： ");

while(scanf("%d", &n) != EOF)

{

LARGE\_INTEGER begin, end;

long long MergeSortCost, QuickSortCost;

memset(a, 0, sizeof(a));

for(int i = 0; i < n; i++) //生成随机数

{

a[i]= rand();

b[i] = a[i];

printf("%d\n",a[i]);

}

QueryPerformanceCounter(&begin); //计算归并排序时间

merge\_sort(b, 0, n, t);

QueryPerformanceCounter(&end);

MergeSortCost = end.QuadPart - begin.QuadPart;

QueryPerformanceCounter(&begin); // 计算快排时间

quickSort(a,0, n-1);

QueryPerformanceCounter(&end);

QuickSortCost = end.QuadPart - begin.QuadPart;

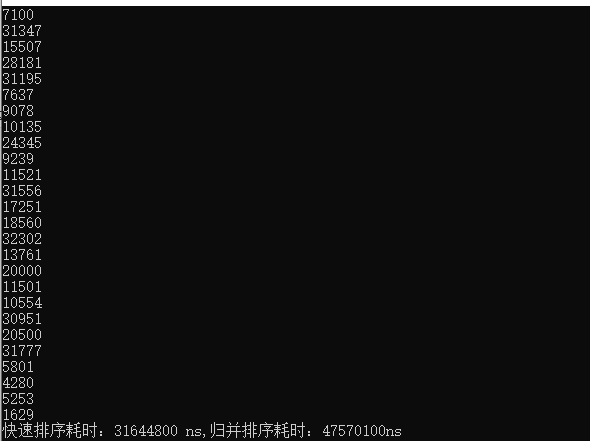
printf("快速排序耗时：%lld ns,归并排序耗时：%lldns\n\n\n", QuickSortCost\*100, MergeSortCost\*100);

}

return 0;

}5.**测试样例**

输入100000个数时运行结果为：



**6.时空分析**

归并排序的时间复杂度为O(nlogn)，

快速排序的时间复杂度为O(nlog2n)

7.**个人心得体会**

在算法设计中，首先需要生成一个随机的序列S，用数组a[n]来表示，然后分别构造出归并排序和快速排序的方法，调用生成的数组a[n]，实现对随机数组中的元素按照从小到大的顺序排列。归并和快速排序的平均时间复杂度都是一样的为O（nlogn）。