**算法分析与设计实验报告**

**第 1 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 杨杰 | 学号 | 201908010705 | | 班级 | 计科1907 |
| 时间 | 3.18 | 地点 | 软件大楼 | | | |
| 实验名称 | 分治法实现合并排序 | | | | | |
| 实验目的 | 通过上机实验，要求掌握分治法实现合并排序算法的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 利用分治法实现合并排序，根据不同数据规模的测试用例，能准确输出测试用例的排序结果，并计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | 分而治之(divide - conquer)，每个递归过程涉及三个步骤： 第一，分解：把待排序的 n 个元素的序列分解成两个子序列, 每个子序列包括 n/2 个元素； 第二，治理：对每个子序列分别调用归并排序MergeSort, 进行递归操作； 第三，合并：合并两个排好序的子序列,生成排序结果。 | | | | | |
| 关键代码 |  | | | | | |
| 算法复杂度分析 | 设需排序元素的数目为n，递归的深度为logn（简单起见，设n是2的幂），第一层递归是对一个长度为n的数组排序，下一层是对两个长度为 n/2的子数组排序，……，最后一层对n个长度为1 的子数组排序。 时间复杂度：T(n)=O(nlogn) 空间复杂度：S(n)=O(n) | | | | | |
| 测试结果  （含运行时间） |  | | | | | |
| 实验心得 | 最初调用rand()函数生成随机数时，并没有指定随机数的种子，导致每次生成的随机数都是相同的，后来调用srand((int)time(NULL))函数，将随机数种子设置为系统时间，解决了这个问题。  clock()函数、GetTickCount()函数、timeGetTime()函数、Boost库中的timer、高精度时控函数QueryPerformanceFrequency()和QueryPerformanceCounter()，以上函数都可以记录程序运行时间，但由于数据量较小时，程序运行时间很短，前三个函数的精度不足以计时，这时应该选用高精度时控函数QueryPerformanceFrequency()和QueryPerformanceCounter()。  经过这次实验，我对于归并排序的相关代码已基本熟悉，算法知识得到了复习与巩固。在写代码与调试的过程中，在解决问题过程中，丰富了个人编程的经历和经验，提高了个人解决问题的能力。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

*//归并排序*

#include <iostream>

#include <fstream>

#include <windows.h>

#include <cmath>

using namespace std;

int A[1000001],temp[1000001];

int count=0;

void Init(fstream &infile)

{

    int x=0;

    count=0;

    while(infile>>x)

        A[count++]=x;

}

void print(fstream &outfile)

{

    for(int i=0;i<count;i++)

        outfile<<A[i]<<endl;

}

bool cmp(int x,int y)

{

    return x<y;

}

void mergesort(int A[],int temp[],int left,int right)*//归并排序*

{

    if(left==right)

        return;

    int mid=(left+right)/2;

    mergesort(A,temp,left,mid);

    mergesort(A,temp,mid+1,right);

    for(int i=left;i<=right;i++)

        temp[i]=A[i];

    int i1=left;

    int i2=mid+1;

    for(int curr=left;curr<=right;curr++)

    {

        if(i1==mid+1)

            A[curr]=temp[i2++];

        else if(i2>right)

            A[curr]=temp[i1++];

        else if(cmp(temp[i1],temp[i2]))

            A[curr]=temp[i1++];

        else

            A[curr]=temp[i2++];

    }

}

int main()

{

    string files[6]={"100.txt","10k.txt","1M.txt",

                    "out\_100.txt","out\_10k.txt","out\_1M.txt"};

    fstream file;

    double time=0;

    LARGE\_INTEGER nFreq,nBeginTime,nEndTime;

    int k=0;

    for(int i=0;i<3;i++)*//对3个测试文件进行归并排序*

    {

        file.open(files[i],ios::in);*//打开测试文件*

        Init(file);*//初始化文件将数据填入序列A中*

        file.close();

        QueryPerformanceFrequency(&nFreq);

        QueryPerformanceCounter(&nBeginTime);*//开始计时*

        mergesort(A,temp,0,100\*pow(10,k)-1);*//归并排序*

        QueryPerformanceCounter(&nEndTime);*//停止计时*

        time=(double)(nEndTime.QuadPart-nBeginTime.QuadPart)/(double)nFreq.QuadPart;*//计算程序执行时间单位为s*

        file.open(files[i+3],ios::out);*//打开测试文件*

        file<<"数据规模为"<<100\*pow(10,k)<<"的数据进行归并排序,结果为"<<time\*1000<<"ms"<<endl;

        print(file);

        file.close();

        k+=2;

    }

    return 0;

}

*//测试数据生成器.cpp*

#include <iostream>

#include <time.h>

#include <fstream>

#include <cmath>

#define random(x) (int)(((double)rand()/RAND\_MAX)\*pow(10,rand()%x))

using namespace std;

const int maxn=1e6;*//1M*

int array[maxn+5]={0};*//存储产生的随机数*

void print(fstream &outfile,int n,int A[])*//输出*

{

    for(int i=0;i<n;i++)

        outfile<<A[i]<<endl;

}

int main()

{

    srand((int)time(NULL));*//time seed*

    string files[5]={"100.txt","1k.txt","10k.txt","100k.txt","1M.txt"};

    fstream file;

    for(int i=0;i<maxn;i++)*//随机产生1M的数据*

        array[i]=random(7);

    int k=100;

    for(int i=0;i<5;i++)*//装载5个test文件*

    {

        file.open(files[i],ios::out);*//打开文件*

        print(file,k,array);

        file.close();*//关闭文件*

        k\*=10;

    }

    cout<<"随机数生成成功！"<<endl;

    return 0;

}