实验2 分治算法实现

姓名：叶倩琳

班级：软工1706

完成时间：2019/03/25

**问题一**

1. **问题描述**

实现插入排序、合并排序和快速排序，比较算法的性能。

输入数据为：3 2 5 7 8 9 5 0 1 3 4

尝试改变数列规模或初始输入使性能差距更明显。

1. **代码实现**

#include "stdafx.h"

#include<iostream>

#include<algorithm>

using namespace std;

//打印数组

void print(int \*a,int left,int right){

int i;

for(i=left;i<=right;i++)

cout<<a[i]<<" ";

cout<<endl<<endl;

}

//插入排序

void InsertSort(int \*a,int left,int right){

int i,j;

for(i=left+1;i<=right;i++){

for(j=left;j<i;j++){ //第二层循环对已排好序的序列进行扫描，和要插入的数进行比较，决定插

入到哪里

if(a[j]>a[i])

swap(a[i],a[j]);

}

}

}

//合并排序

void Merge(int \*a,int left,int mid,int right);

void MergeSort(int \*a,int left,int right){

if(left<right){

int mid=(left+right)/2;

MergeSort(a,left,mid);

MergeSort(a,mid+1,right);

Merge(a,left,mid,right);

}

}

void Merge(int \*a,int left,int mid,int right){ //合并

int\*b=new int[right-left+1];

int i=left,j=mid+1,k=left;

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

if(a[i]<=a[j]) b[k++]=a[i++];

else b[k++]=a[j++];

}

if(i>mid){

while(j<=right)

b[k++]=a[j++];

}

else{

while(i<=mid)

b[k++]=a[i++];

}

for(i=left;i<=right;i++) //复制回数组a

a[i]=b[i];

}

//快速排序

int Partition(int \*a,int left,int right);

int RandomPartition(int \*a,int left,int right);

void RandomQuickSort(int \*a,int left,int right){

if(left<right){

int q=RandomPartition(a,left,right);

RandomQuickSort(a,left,q-1); //对左半段排序

RandomQuickSort(a,q+1,right); //对右半段排序

}

}

int RandomPartition(int \*a,int left,int right){ //产生随机划分

int q=left+rand()%(right-left);

swap(a[q],a[left]);

return Partition(a,left,right);

}

int Partition(int \*a,int left,int right){

int i=left,j=right+1;

int x=a[left]; //基准

while(true){

while(a[++i]<x && i<right);

while(a[--j]>x); //两个都为false跳出循环

if(i>=j) break;

swap(a[i],a[j]);

}

a[left]=a[j];

a[j]=x;

return j;

}

int main(int argc, char\* argv[])

{

int a[]={3,2,5,7,8,9,5,0,1,3,4};

cout<<"原数组为：";

print(a,0,10);

cout<<"插入排序后的数组为：";

InsertSort(a,0,10);print(a,0,10);

cout<<"合并排序后的数组为：";

//MergeSort(a,0,10);print(a,0,10);

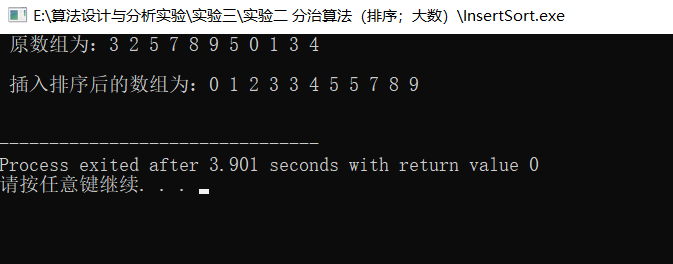
cout<<"快速排序后的数组为：";

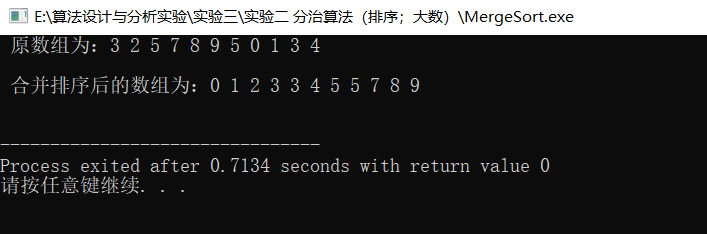
//RandomQuickSort(a,0,10);print(a,0,10);

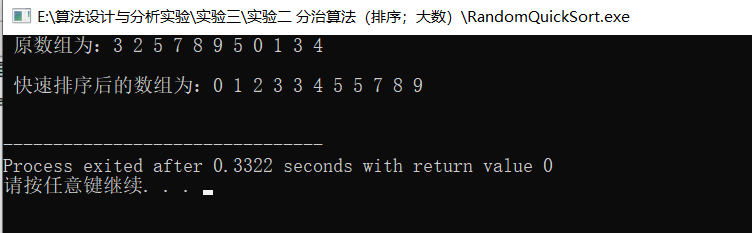
return 0;

}

1. **输出结果**







1. **性能分析**

**1)插入排序**

插入排序就是从待排序列中选出一个元素，插入到已经有序的元素之中。首先，将第一个元素看做是有序的元素（即待排序列的第一个元素看做是有序序列），然后将第二个元素和有序序列作比较，按正确的序列插入到序列中去。以此类推，直到所有的元素都有序。

时间复杂度 :

最坏情况：当序列为逆序状态，首先遍历整个序列，将待插入元素放在已排序的序列最前面，之后的所有元素都需要向后移动一位，比较和移动的时间复杂度都是O(n)，再加上遍历整个序列的复杂度，总复杂度为O(n^2)。

最好情况：当序列为正序，当插入元素时，只比较一次，时间复杂度为O(n)。

平均情况：当被插入的元素放在已排序的序列中间位置时，比较和移动的时间复杂度为O(n/2)，总的时间复杂度仍为O(n^2)。

插入排序是稳定的排序算法。

**2)合并排序**

合并排序采用分治算法，先递归的把数组划分为两个子数组，一直递归到数组中只有一个元素，然后再调用Merge函数把两个子数组合并。Merge排序函数的思路是，让两个数组的元素逐一进行比较，开辟一个临时数组int \*b=new int[right-left]，把大的/小的元素存放到临时数组b中，如果有一个数组的元素被取光了，那就直接把另一数组的元素放到临时数组中，然后把临时数组中的元素都复制到实际的数组（a）中。所以，合并排序的空间复杂度就是临时的数组和递归时压入栈的数据占用的空间n + logn，即空间复杂度为O(n)。

时间复杂度 :

合并排序的主要是分解和合并，分解序列时，每次都把序列分为大致相等的两部分T(n)=2T(n/2)+O(n)＝O(nlogn)。合并时要遍历一遍数组，时间复杂度为O(n)。合并排序与快速排序类似，可以用二叉树表示，树的每层元元素的个数最多是n，即每层最多进行n次比较，且递归树高最多为log2n，时间复杂度为O( nlogn )。

合并排序也是稳定的排序算法。

**3)快速排序**

快速排序也是基于分治策略的一种排序算法。它的原理是选择一个基准元素，通过一趟扫描，将待排序列分成两部分，比基准元素小和大于等于基准元素，再用递归地排序划分的两部分。其中，Partition函数用来确定基准元素以对子数组进行划分，该函数要遍历一遍数组，Partition的计算时间为O(n)。

时间复杂度

最坏情况：函数Partition的每一步都出现不对称划分，T(n)=T(n-1)+O(n)，复杂度为O(n^2) 。

最好情况：每次选取的基准平分整个序列，T(n)=2T(n/2)+O(n),时间复杂度为O(nlog2n)。

平均情况：时间复杂度为O(nlog2n)。

快速排序每次排序都可能会打乱之前排好序的元素，所以是不稳定的算法。

1. **实验体会**

总的来说，每种排序算法都存在优缺点，时间复杂度也不同，我们要根据实际选择合适的时间复杂度尽可能小的排序算法。就本题的计算而言，合并算法最优。

**问题二**

1. **问题描述**

大数乘法问题。分别尝试计算9\*9， 9999\*9999, 9999999999\*8888888888的结果。分析算法性能。

1. **代码实现**

**法一：**

#include <iostream>

#include <cstdlib>

#include <algorithm>

#include <cmath>

using namespace std;

string product(string a,string b){ //乘积

int len1 = a.length();

int len2 = b.length();

string ans;

int num = 0;

int carry = 0;

if(len1 == 1){

for(int i=len2-1;i>=0;--i){

num = carry + (a[0]-'0')\*(b[i]-'0');

carry = num/10;

num %= 10;

ans += (char)(num+'0');

}

if(carry) ans += (char)(carry+'0');

}

else if(len2 == 1){

for(int i=len1-1;i>=0;--i){

num = carry + (b[0]-'0')\*(a[i]-'0');

carry = num/10;

num %= 10;

ans += (char)(num+'0');

}

if(carry) ans += (char)(carry+'0');

}

reverse(ans.begin(),ans.end());

return ans;

}

string add(string a,string b,string c,string d){ //求和

string ans;

int len1 = a.length()-1;

int len2 = b.length()-1;

int len3 = c.length()-1;

int len4 = d.length()-1;

int num = 0;

int carry = 0;

while(len1 >= 0 || len2 >= 0 || len3 >= 0 || len4 >= 0){

num = carry;

if(len1 >= 0) num += a[len1]-'0';

if(len2 >= 0) num += b[len2]-'0';

if(len3 >= 0) num += c[len3]-'0';

if(len4 >= 0) num += d[len4]-'0';

carry = num/10;

num %= 10;

ans += (char)(num+'0');

--len1;--len2;--len3;--len4;

}

while(carry){

ans += (char)(carry%10 + '0');

carry /= 10;

}

reverse(ans.begin(),ans.end());

return ans;

}

//分治法求大数乘法

string LargeMultiply(string a,string b,int n1,int n2){

if(n1 == 1 || n2 == 1){

return product(a,b);

}

int len1 = n1/2;

int len2 = n2/2;

string A,B,C,D; //将a分为A、B两部分，将b分为C、D两部分

A.append(a,0,len1);

B.append(a,len1,n1-len1);

C.append(b,0,len2);

D.append(b,len2,n2-len2);

// cout << A << " " << B << " " << C << " " << D << endl;getchar();

string num1 = LargeMultiply(A,C,len1,len2);

string num2 = LargeMultiply(A,D,len1,n2-len2);

string num3 = LargeMultiply(B,C,n1-len1,len2);

string num4 = LargeMultiply(B,D,n1-len1,n2-len2);

num1.append(n1-len1+n2-len2,'0');

num2.append(n1-len1,'0');

num3.append(n2-len2,'0');

// cout << num1 << " " << num2 << " " << num3 << " " << num4 << endl;getchar();

return add(num1,num2,num3,num4);

}

int main(){

string a,b;

cout<<" 请输入两个乘数：";

while(cin >> a >> b){

int n1 = a.length();

int n2 = b.length();

cout << a<<" \* "<<b<<" = "<<LargeMultiply(a,b,n1,n2) << endl;

// cout << product(a,b) << endl;

// cout << add(a,b,a,b) << endl;

cout<<" 请输入两个乘数：";

}

return 0;

}

**法二：**

#include<iostream>

#include<cmath>

using namespace std;

//分治法计算大数乘法

long int LargeMultiply(long int X, long int Y,long int n){

int sign=((X>0)? 1 : -1)\*((Y>0)? 1: -1);

long int x=abs(X);

long int y=abs(Y);

if(x==0 || y==0)

return 0;

else if(n==1)

return sign\*x\*y;

else{

long int A=(int)x/pow(10,(int)(n/2));

long int B=x-A\*pow(10,n/2);

long int C=(int)y/pow(10,(int)(n/2));

long int D=y-C\*pow(10,n/2);

long int AC=LargeMultiply(A,C,n/2);

long int BD=LargeMultiply(B,D,n/2);

long int ABCD=LargeMultiply((A-B),(D-C),n/2)+AC+BD;

return sign\*(AC\*pow(10,n)+ABCD\*pow(10,(int)(n/2))+BD);

}

}

int main()

{

long int x,y,n;

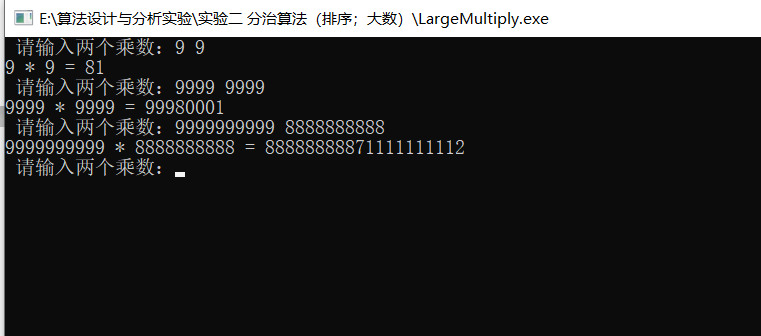
cin>>x>>y>>n;

cout << x<<" ＊ "<<y<<" = "<<LargeMultiply(x,y,n);

return 0;

}

1. **输出结果**



1. **性能分析**

用分治法进行大数乘法运算可以把每个大数分解为长度相等的两个大整数X=(A\*10^(n/2)+B),同理Y=(C\*10^(n/2)+D)，再不断递归分解直到每个乘数的长度n=1为止，一共要进行4次 n / 2的乘法（AC2次， AD， BC）和 3次加法，因而该算法的时间复杂度T(n) = 4 \* T(n / 2) + θ(n) ，通过master定理可得 T(n) = θ(n ^ 2)。

要优化算法就要减小乘数，即减小T(n/2)前面的常数项，化简X\*Y，得：XY=AC2^n+[(A-B)(D-C)+AC+BD]2^n/2+BD

现在的时间复杂度为T(n) = 3 \* T(n / 2) + θ(n)，通过master定理求得，T(n) = O(n^log2(3) ) = O(n^1.59 )。

1. **实验体会**

总的来说，如果用之前学的通过数组保留每一位按位乘的结果，时间复杂度为O(n^2)。而用分治算法可以把大数分解为长度相同的两项大数，把问题分解为若干个小问题，逐一求每个子问题的最优解。在本题中还可以尽可能减小T(n/2)的常数项达到优化算法的目的。

在方法二中用long int作为参数，导致乘数的大小有限，而方法一中用string作参数，就可以计算任何大小的乘数。

**问题三**

1. **问题描述**

线性时间选择问题：

1. 在 4 59 7 23 61 55 46中找出最大值，第二大值，和第四大的值（要求不允许采用排序算法），并与第1题实现的快速排序算法进行比较。
2. 随机生成10000个数，要求找出其中第4999小的数，并与第1题实现的快速排序算法进行比较。
3. **代码实现**

**（1）**

#include<iostream>

using namespace std;

/\*以5为基，五数取分的线性选择法\*/

const int maxm=100; //以5划分

int mid[maxm]; //用于存放中位数

//对每组的5个元素进行插入排序

void InsertSort(int \*a,int left,int right){

int i,j;

for(i=left+1;i<=right;i++){ //从第二个数开始插入排序，因为第一个数已排好序

for(j=left;j<i;j++){ //第二层循环对已排好序的序列进行扫描，和要插入的数进行比较，决定插入到哪里

if(a[j]>a[i])

swap(a[i],a[j]);

}

}

}

//查找中位数的中位数

int FindMedian(int \*a,int left,int right){

if(left==right)

return a[left];

int i,j;

//对每个5个一组的元素进行插入排序，找出中位数

if(right-left+1>=5){

for(i=left;i<=right-4;i+=5){

InsertSort(a,left+i,left+i+5);

int num=i-left;

mid[num/5]=a[i+2]; //保存到中位数数组中

}

}

//处理剩下的不足5个的元素

int remain=right-i+1; //剩下的元素个数

if(remain>0){

InsertSort(a,left+i,right);

int num=i-left;

mid[num/5]=a[i+(remain>>1)]; //保存到中位数数组中

}

int cnt=(right-left+1)/5; //有几组5个一组的数

if((right-left+1)%5==0) {

cnt--; //下标是从0开始，所以需要-1

}

if(cnt==0)

return mid[0];

else

return FindMedian(mid,0,cnt); //递归查找中位数的中位数

}

//线性快速查找第k小的数

int QSelect(int a[],int left,int right,int k){

int x=FindMedian(a,left,right); //查找中位数的中位数

int i=left-1,j=right+1;

while(true){

while(a[++i]<x && i<right);

while(a[--j]>x); //两个都为false跳出循环

if(i>=j) break;

swap(a[i],a[j]);

}

int remain=i-left+1; //剩下的数

if(k==remain)

return a[i];

else if(k<remain)

return QSelect(a,left,i-1,k);

else

return QSelect(a,i+1,right,k-remain);

}

int main() {

int a[]={4,59,7,23,61,55,46};

cout<<" 数组a的元素分别为：4,59,7,23,61,55,46"<<endl;

cout<<" 在数组a中最大的元素为："<<QSelect(a,0,6,7)<<endl;

cout<<" 在数组a中第二大的元素为："<<QSelect(a,0,6,6)<<endl;

cout<<" 在数组a中第四大的元素为："<<QSelect(a,0,6,4)<<endl;

return 0;

}

**（2）**

#include <iostream>

#include<ctime>

#include<cstring>

#include<algorithm>

using namespace std;

/\* 随机生成10000个数，要求找出其中第4999小的数 \*/

//寻找基准并进行划分

template <class Type>

int Partition(Type a[],int p,int r) {

int i = p,j = r + 1;

Type x = a[p];

while(true)

{

//两个while循环都为false时跳出

while(a[++i]<x && i<r);

while(a[--j]>x);

if(i>=j)

break;

swap(a[i],a[j]);

}

a[p] = a[j];

a[j] = x;

return j; //返回基准元素的位置

}

//随机选择基准位置

template<class Type>

int RandomizedPartition(Type a[],int p,int r) {

int i = p+rand()%(r-p);

swap(a[i],a[p]); //让基准位于第一个

return Partition(a,p,r);

}

//查找第k小的元素

template <class Type>

Type RandomizedSelect(Type a[],int p,int r,int k) {

if(p == r) {

return a[p];

}

int i = RandomizedPartition(a,p,r);

int j = i - p + 1;

if(k <= j) {

return RandomizedSelect(a,p,i,k);

}

else {

//要找的a[p:r]中第k小元素是a[i+1，r]中第k-j小元素

return RandomizedSelect(a,i+1,r,k-j);

}

}

int main(int argc, char\*\* argv) {

int i,num[10000];

srand((unsigned)time(NULL)); //用时间做种子，每次产生的随机数不同

for(i=0;i<10000;i++){

num[i]=rand();

}

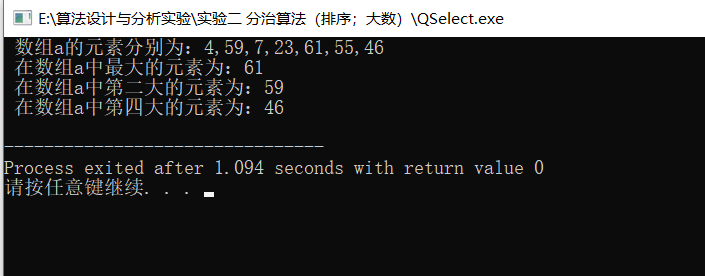
cout<<" 在随机生成的10000个数中，第4999小的数为："<<RandomizedSelect(num,0,10000,4999);

return 0;

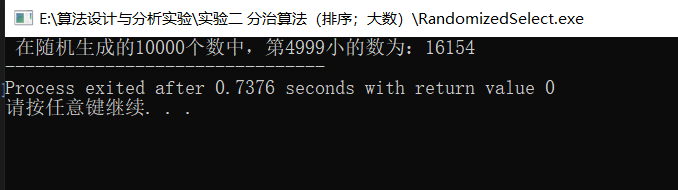
}

1. **输出结果**

**（1）**



**（2）**



1. **性能分析**

用分治法计算第k小的数的基本思路如下：1) 当规模小于阈值时，直接用排序算法返回结果。2) 当n大于阈值时，把n个元素划分为5个一组，共n/5组，分别排序，挑出每一组的中位数，把他们集中在一起，再取中位数的中位数。3) 分两种情况进行递归调用本函数：若基准前的元素（比基准小的元素）数量大于等于K，在a[p,i]中递归查找第k小元素；若基准前的元素（比基准小的元素）数量小于k，表示第K个元素在基准元素之后，所以在a[i+1,r]中递归寻找第（k-|A1、A2元素数量之和|）小元素。因此，线性选择算法的时间复杂度为O(n)。

1. **实验体会**

总的来说，线性时间选择算法(以5为基，五数取分)可以在线性时间内找到第k小的元素，即时间复杂度为O(n)，算法效率非常高。但在进行该算法设计FindMedian（）函数时，我遇到了一些小问题，通过查阅资料找到了解决办法：定义一个全局数组用来存放查找到的中位数，再对该数组不断递归，得到结果。可见，要实践与学习相结合才能真正的掌握一个算法。