**实验报告1 基本算法设计练习**

## 一、实验目标

1. 熟悉算法实现的基本方法和步骤

2. 学会基本算法的实现方法和技巧

## 二、实验项目

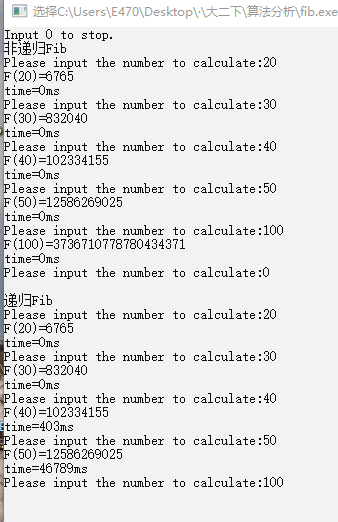
**一**

1. **实验内容**

实现 fibonacci 的递归和非递归。要求计算F（100）的值，比较两种方法的性能。

要求1）有合适的提示从键盘输入数据；例如“Please input the number to calculate:” 2）有输出结果（下同）

1. **实验结果**

****

1. **源代码**

#include<iostream>

#include<ctime>

using namespace std;

long long fib1(int n){//递归

if(n<=2) return 1;

return fib1(n-1)+fib1(n-2);

}

long long fib2(int n){//非递归

if(n<=2) return 1;

long long f1=1,f2=1,sum;

for(int i=3;i<=n;i++){

sum=f1+f2;

f1=f2;

f2=sum;

}

return sum;

}

int main(){

int n; clock\_t start,end;

cout<<"Input 0 to stop."<<endl;

cout<<"非递归Fib"<<endl;

while(1){

cout<<"Please input the number to calculate:";

cin>>n; if(n==0) break;

start=clock();

cout<<"F("<<n<<")="<<fib2(n)<<endl;

end=clock();

cout<<"time="<<(double)(end-start)\*1000/CLOCKS\_PER\_SEC<<"ms"<<endl;

}

cout<<endl;

cout<<"递归Fib"<<endl;

while(1){

cout<<"Please input the number to calculate:";

cin>>n; if(n==0) break;

start=clock();

cout<<"F("<<n<<")="<<fib1(n)<<endl;

end=clock();

cout<<"time="<<(double)(end-start)\*1000/CLOCKS\_PER\_SEC<<"ms"<<endl;

}

return 0;

}

1. **实验体会**

由实验结果可知，fibonacci的递归函数比非递归函数运行效率低，随着运算数值的增大变得明显。计算这些大数的运算时，一开始用int超范围了，改为long long可以接受1018左右的数，非递归函数就可以快速算出F(100)了，不过递归函数太耗时。

**二**

1. **实验内容**

实现全排列的算法

a) 实现数字1，3，5，7，9五个数字的全排列。

b) 实现前20个奇数的全排列。

1. **源代码**

#include<iostream>

using namespace std;

void Perm(int list[],int k,int m){//全排列

if(k==m){

for(int i=0;i<=m;i++)

cout<<list[i]<<" ";

cout<<endl;

}

else{

for(int i=k;i<=m;i++){

std::swap(list[k],list[i]);

Perm(list,k+1,m);

std::swap(list[k],list[i]);

}

}

}

long long F(int n){//阶乘 递归

if(n==0) return 1;

return n\*F(n-1);

}

int main(){

int i,n,x,list[100];

cout<<"数组大小:";

cin>>n;

cout<<"输入数组:";

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

cin>>x;

list[i]=x;

}

cout<<"该数组的全排列数有"<<F(n)<<"组(输入1开始排列):";

cin>>x;

if(x==1) Perm(list,0,n-1);

n=20;

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

list[i]=2\*i+1;

}

cout<<"前20个奇数的全排列数有"<<F(n)<<"组(输入1开始排列):"; //20!=2.4..\*10^18

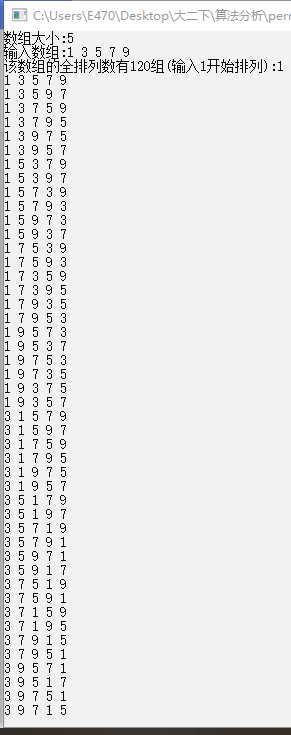
cin>>x;

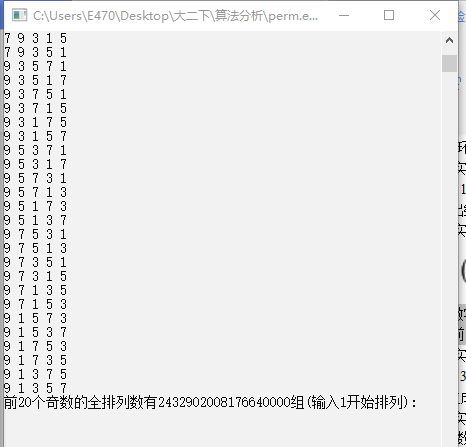
if(x==1) Perm(list,0,n-1);

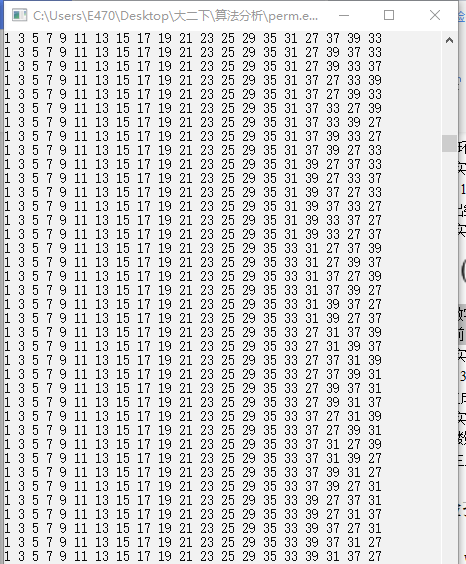
return 0;

}

1. **实验结果**

****





1. **实验体会**

排列的公式：

IMG_256

m=n时为全排列，共有n!种全排列数

当n很大时，n!会很大，用long long型数据存储阶乘结果才可以得到20!的结果

**三**

1. **实验内容**

实现二分查找

a) 在 1 3 3 4 5 5 7 8 8 9 10 中查找得到7，返回其所在的位置。

b) 随机生成10000个整数，查找数字 2025，返回其所在的位置。

1. **源代码**

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<set>

using namespace std;

const int N=1e4;

int BinarySearch(int a[],const int& x,int l,int r){

while(r>=l){

int m=(l+r)/2;

if(x==a[m]) return m;

if(x<a[m]) r=m-1; else l=m+1;

}

return -1;

}

int main(){

int n,x,i,a[N];

cout<<"数组大小:";

cin>>n;

cout<<"输入数组:";

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

cin>>x;

a[i]=x;

}

cout<<"输入要寻找的数:";

cin>>x;

cout<<"其所在位置:";

int local=BinarySearch(a,x,0,n-1)+1;

if(local==0) cout<<"无此数"<<endl;

else cout<<local<<endl;

//随机生成1e4个有序整数-通过set集合去重生成随机有序序列

cout<<"随机生成10000个有序整数(输入1开始):";

cin>>x;

if(x==1){

set<int> b;

set<int>::iterator it;

srand((unsigned)time(0));

clock\_t start,end;

start=clock(); // 计时器开始

//运行时set和list只能生成size()/2的递增随机数,所以用N\*2生成N个随机数

while(b.size()<2\*N){

b.insert(rand());

}

end=clock(); // 计时器停止

it=b.begin();

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

if(i%20==0) { cout<<endl; cout<<i+1<<"~"<<i+20<<":"<<'\t'; }

a[i]=\*(it);

cout<<\*(it++)<<'\t';

}cout<<endl;

cout<<"用时"<<(double)(end-start)\*1000/CLOCKS\_PER\_SEC<<"ms"<<endl;

cout<<"输入要寻找的数:";

cin>>x;

cout<<"其所在位置:";

local=BinarySearch(a,x,0,N-1)+1;

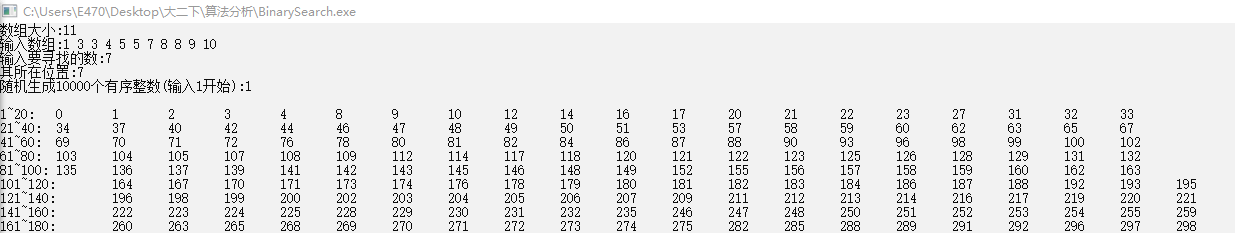
if(local==0) cout<<"无此数"<<endl;

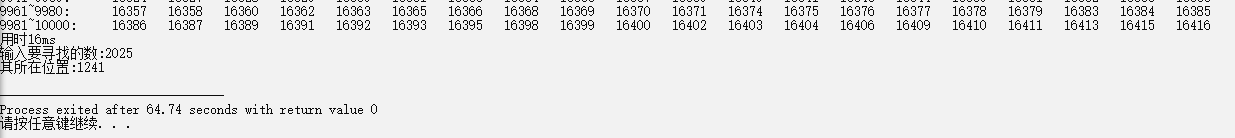
else cout<<local<<endl;

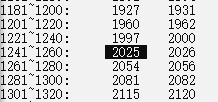
}

return 0;

}

1. **实验结果**

****

****在1241位置找到2025

1. **实验体会**

①rand()%n获得[0,n-1]的随机整数的缺点:

(1)当n不是2的若干次幂时,产生的随机数不是等概率的

(2)rand()函数有上限,只能生成[0,32767]的伪随机数,所以n<=RAND\_MAX+1=32768

通过多个随机数拼接生成随机数可解决上述问题

生成[0,n-1]之间的整数

long long Rand(long long n){

long long result=0;

// 这里的随机数上限是1e18,考虑时间效率与数据范围选择每次左移10位

for(int i=0;i<6;i++){

result=((result<<10)+rand()%1024)%n;

}

return result;

}

②set集合是去重有序序列。

list集合可包含重复元素，它不能自动排序，若要使list集合的元素升序排列，则要用list.sort()函数。

list有两个版本的sort成员函数：

一个是不带参数的sort()，用来实现升序排列；另一个是带参数的sort(greater<T> pr)，用来实现降序排列。

**四**

1. **实验内容**

实现合并排序和快速排序，比较算法的性能。

a) 输入数据为：3 2 5 7 8 9 5 0 1

b) 随机生成10000个整数。

1. **源代码**

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <iomanip>

#define N 10000

using namespace std;

//归并-非递归 nlogn

void Merge(int c[],int d[],int l,int m,int r){

int i=l,j=m+1,k=l,q;

while( i<=m && j<=r ){

if(c[i]<=c[j])

d[k++]=c[i++];

else

d[k++]=c[j++];

}

if(i>m)

for(q=j;q<=r;q++)

d[k++]=c[q];

else

for(q=i;q<=m;q++)

d[k++]=c[q];

}

void MergePass(int x[],int y[],int s,int n){

int i=0,j;

while(i<=n-2\*s){//合并大小为s的相邻子段到y

Merge(x,y,i,i+s-1,i+2\*s-1);

i+=2\*s;

}

if(i+s<n) //处理剩下的元素(大于s个，少于2s个)

Merge(x,y,i,i+s-1,n-1);

else //处理剩下的元素(少于s个)

for(j=i;j<=n-1;j++)

y[j]=x[j];

}

void MergeSort(int a[],int n){

int s=1;

int \*b=new int[n];

while(s<n){

MergePass(a,b,s,n);//合并到临时存储数组b

s+=s;

MergePass(b,a,s,n);//合并到数组a

s+=s;

}

delete []b;

}

//快排-递归 nlogn

void Swap(int& a,int& b){int t=a; a=b; b=t;}

int Partition (int a[], int l, int r){

int i = l, j = r,x=a[l];

//将<x的交换到左边区域，>x的交换到右边区域

while (i<j) {

while (a[j]>x && i<r)j--;

if(i<j) {

Swap(a[i++], a[j]);

}

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

if (i<j){

Swap(a[i], a[j--]);

}

}

return i;

}

void QuickSort(int a[], int l, int r)

{

if (l<r) {

int q=Partition(a,l,r); //以a[p]分基准为左右两半

QuickSort (a,l,q-1); //对左半段排序

QuickSort (a,q+1,r); //对右半段排序

}

}

void Display(int x[],int n){

for(int i=0;i<n;i++){

if(i%20==0) cout<<endl;

cout<<setw(5)<<x[i]<<" ";

}cout<<endl;

}

void ResetData(int x[],int y[],int n){//使用y重置x

for(int i=0;i<n;i++) x[i]=y[i];

}

int main(){

int n,x,i,a[N],b[N];

cout<<"数组大小:";

cin>>n;

cout<<"输入数组:";

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

cin>>a[i];

b[i]=a[i];

}

cout<<endl;

cout<<"归并排序后:"<<endl;

MergeSort(a,n);

Display(a,n);

cout<<"重置:"<<endl;

ResetData(a,b,n);

Display(a,n);

cout<<"快速排序后:"<<endl;

QuickSort(a,0,n-1);//原来写成n耽误了很久！！！！这里的n-1指数组最后一项

Display(a,n);

srand(time(0));//随机数种子初始化

for(i=0;i<N;i++)

{ a[i]=rand()%10000; //随机生成0-10000内的数值作为排序对象

b[i]=a[i];

}

clock\_t start,end,duration[2];

//排序前显示数据

cout<<"排序前，随机生成10000个整数，输入1开始输出:";

cin>>x;

if(x==1){

Display(a,N);

//归并排序

start=clock();

MergeSort(a,N);

end=clock();

duration[0]=end-start;

cout<<"归并排序后:"<<endl;

Display(a,N);

cout<<"用时"<<duration[0]<<"ms"<<endl;

//快速排序

ResetData(a,b,N);

start=clock();

QuickSort(a,0,N);

end=clock();

duration[1]=end-start;

cout<<"快速排序后:"<<endl;

Display(a,N);

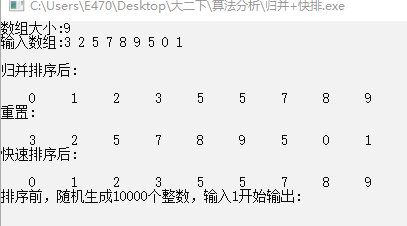
cout<<"用时"<<duration[1]<<"ms"<<endl;

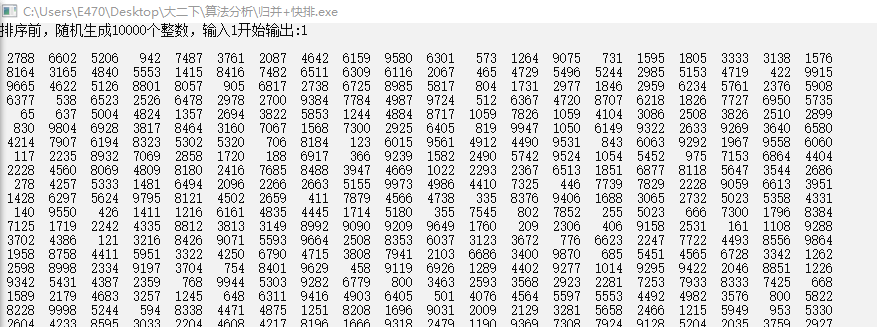
}

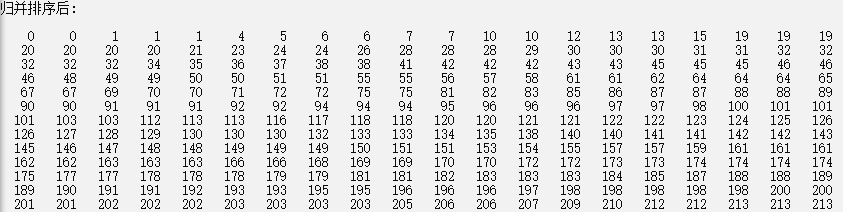
return 0;

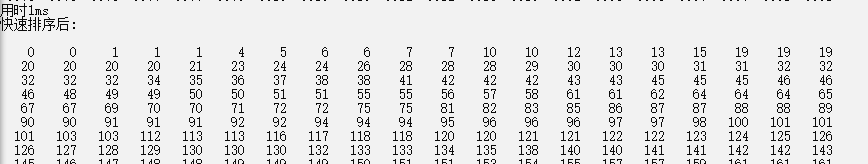
}

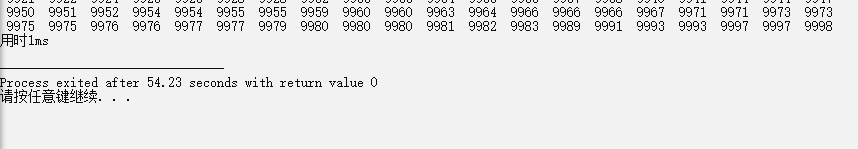
1. **实验结果**

****

****

****

****

****

归并和快排用时差不多

1. **实验体会**

归并和快排的算法复杂度都是nlogn，是排序方法中较快的。快排不稳定，但是平均下来比归并快。