# 算法分析第2次作业

算法分析题

2-3

void BinarySearch(const int *a*[],int *left*,int *right*,int &*i*,int &*j*,int *x*)

{

    while(*left*<=*right*)

    {

        int mid=*left*+(*right*-*left*)/2;

        if(*a*[mid]==*x*)

        {

*i*=*j*=mid;

            return;

        }

        else if(*a*[mid]>*x*)

        {

*j*=mid;

*right*=mid-1;

        }

        else

        {

*i*=mid;

*left*=mid+1;

        }

    }

}

*//需要的两个参数采用地址传参进行返回，初始值i=-1,j=n*

*//基本思想与原二分搜索算法基本一致 时间复杂性为O(logn)*

2-4

string add(string *num1*,string *num2*)

{

    string res="";

    bool isMinor=false;

    if(*num1*[0]=='-'&&*num2*[0]!='-')

    {

        return substract(*num2*,*num1*);

    }

    else if(*num1*[0]!='-'&&*num2*[0]=='-')

    {

        return substract(*num1*,*num2*);

    }

    else if(*num1*[0]=='-'&&*num2*[0]=='-')

    {

*num1*=*num1*.substr(1);

*num2*=*num2*.substr(2);

        isMinor=true;

    }

    int carry=0;

    int i=*num1*.length()-1;

    int j=*num2*.length()-1;

    while(i>=0||j>=0||carry>0)

    {

        int digit1=(i>=0)?*num1*[i]-'0':0;

        int digit2=(j>=0)?*num2*[j]-'0':0;

        int sum=digit1+digit2+carry;

        carry=sum/10;

        res=to\_string(sum%10)+res;

        i--;

        j--;

    }

    if(isMinor)

    {

        res="-"+res;

    }

    return res;

}

string substract(string *num1*,string *num2*)

{

    string res="";

    if(*num1*[0]!='-'&&*num2*[0]=='-')

    {

        return add(*num1*,*num2*.substr(1));

    }

    if(*num1*[0]=='-'&&*num2*[0]!='-')

    {

        return "-"+add(*num1*.substr(1),*num2*);

    }

    if(*num1*[0]=='-'&&*num2*[0]=='-')

    {

        return substract(*num2*.substr(1),*num1*.substr(1));

    }

    int borrow=0;

    int i=*num1*.length()-1;

    int j=*num2*.length()-1;

    if(i<j)

    {

        return "-"+substract(*num2*,*num1*);

    }

    if(i==j)

    {

        for(int t=0;t<i;t++)

        {

            if(*num1*[t]<*num2*[t])

            {

                return "-"+substract(*num2*,*num1*);

            }

        }

    }

*//num1>=num2*

    while(i>=0)

    {

        int digit1=*num1*[i]-'0';

        int digit2=(j>=0)?*num2*[j]-'0':0;

        int diff=digit1-digit2-borrow;

        if(diff<0)

        {

            diff+=10;

            borrow=1;

        }

        else

        {

            borrow=0;

        }

        res=to\_string(diff)+res;

        i--;

        j--;

    }

*//去除前导0*

    size\_t pos=res.find\_first\_not\_of('0');

    if(pos!=string::npos)

    {

        res=res.substr(pos);

    }

    return (res.empty())?"0":res;

}

string multiply(string *num1*,string *num2*)

{

    int n=*num1*.length();

    int m=*num2*.length();

    if(n==0||m==0||*num1*=="0"||*num2*=="0")

    {

        return "0";

    }

    if(*num1*=="1")

    {

        return *num2*;

    }

    if(*num2*=="1")

    {

        return *num1*;

    }

    if(n<=2||m<=2)

    {

*//只有在极端情况下，一个数小于2位数，另一个数或结果大于long long的范围会出错误*

        long long result=stoll(*num1*)\*stoll(*num2*);

        return to\_string(result);

    }

    string num1High=*num1*.substr(0,n-n/2);*//A*

    string num1Low=*num1*.substr(n-n/2);*//B*

    string num2High=*num2*.substr(0,m-m/2);*//C*

    string num2Low=*num2*.substr(m-m/2);*//D*

    string z1=multiply(num1High,num2High);*//AC*

    string z2=multiply(num1Low,num2Low);*//BD*

    string z3=substract(num1High+string(n/2,'0'),num1Low);*//A\*10^(n/2)-B*

    string z4=substract(num2High+string(m/2,'0'),num2Low);*//C\*10^(m/2)-D*

    return add(substract(add(z1+string((n+m)/2,'0'),z1+string((n+m)/2,'0')),multiply(z3,z4)),add(z2,z2));

}

*//教材中仅适用于相同位数的整数，这里使用string字符串则适用于更多情况*

*//首先实现了高精度的加法和减法函数*

*//在高精度乘法中，先对特殊情况特别判断，在出现两位数时直接计算*

*//按照公式推导XY=(A\*10^(n/2)+B)\*(C\*10^(m/2)+D)*

*//             =AC\*10^((n+m)/2)+AD\*10^(n/2)+BC\*10^(m/2)+BD*

*//             =2\*AC\*10^((n+m)/2)+(A\*10^(n/2)-B)\*(D-C\*10^(m/2))+2\*BD*

*//进行了三次乘法，时间复杂性为O(n^log3)*

string multiply(string *n*,long long *m*)

{

    if(*n*=="0"||*m*==0)

        return "0";

    if(*n*=="1")

        return to\_string(*m*);

    if(*m*==1)

        return *n*;

    int len=*n*.length();

    string strs[100];

    int t=0;

    for(int i=0;i<len;i+=*m*)

    {

        if(i+*m*>len)

        {

            strs[t]=*n*.substr(i);

        }

        else

        {

            strs[t++]=*n*.substr(i,i+*m*);

        }

    }

    string res=multiply(strs[t],to\_string(*m*));

    len=strs[t].length();

    int i=0;

    while(t--)

    {

        add(res,multiply(strs[t],to\_string(*m*))+string(len+i\**m*,'0'));

        i++;

    }

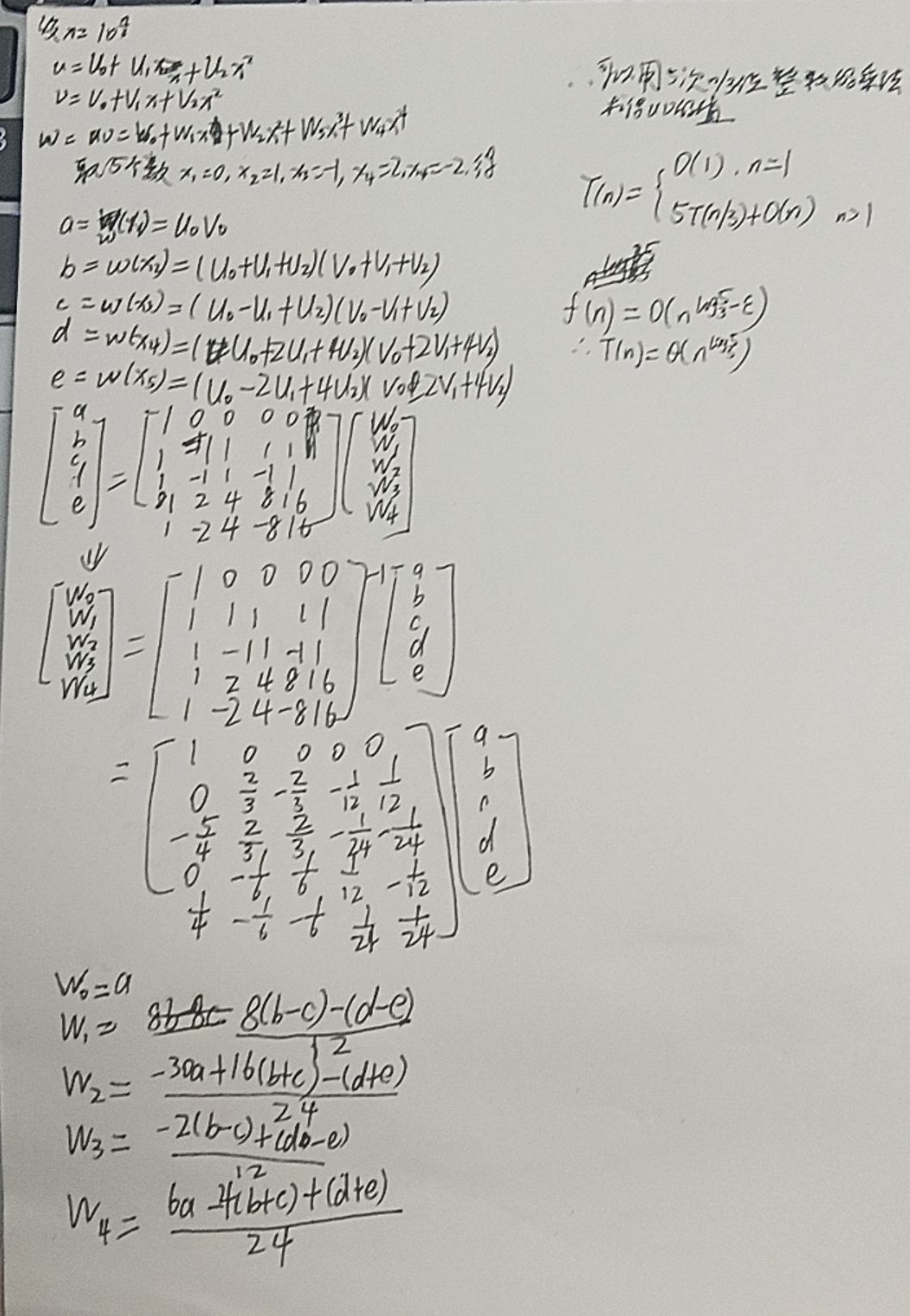
    return res;

}

*//将n分成n/m份，每份为m位，分别与m做乘法，然后再加起来*

*//时间复杂性为O((n/m)\*m^log3)+O(n/m)=O((n/m)\*m^log3)=O(n\*m^log(3/2))*

2-5



string divide(string *num*,int *k*)

{

    string quotient="";

    if(*num*[0]=='-')

    {

        quotient="-";

    }

    int remainder=0;

    for(char digit:*num*)

    {

        int currentDigit=digit-'0';

        int currentDivident=remainder\*10+currentDigit;

        int currentQuotinent=currentDivident/*k*;

        remainder=currentDivident%*k*;

*//跳过前导0*

        if(quotient.empty()&&currentQuotinent==0)

        {

            continue;

        }

        quotient+=to\_string(currentQuotinent);

    }

    return quotient;

}

string multiply3(string *u*,string *v*)

{

    int n=*u*.length();

    if(n==0||*u*=="0"||*v*=="0")

    {

        return "0";

    }

    if(*u*=="1")

    {

        return *v*;

    }

    if(*v*=="1")

    {

        return *u*;

    }

    if(n<=2)

    {

        long long result=stoll(*u*)\*stoll(*v*);

        return to\_string(result);

    }

    string uHigh=*u*.substr(0,n-n/3\*2);*//U2*

    string uMid=*u*.substr(n-n/3\*2,n-n/3);*//U1*

    string uLow=*u*.substr(n-n/3);*//U0*

    string vHigh=*v*.substr(0,n-n/3\*2);*//V2*

    string vMid=*v*.substr(n-n/3\*2,n-n/3);*//V1*

    string vLow=*v*.substr(n-n/3);*//V0*

*//由于有出现负号的可能，会有局限性*

*//当前代码并不完善*

    string a=multiply3(uLow,vLow);

    string b=multiply3(add(uLow,add(uMid,uHigh)),add(uHigh,add(vLow,vMid)));

    string c=multiply3(substract(add(uLow,uHigh),uMid),substract(add(vLow,vHigh),vMid));

    string d=multiply3(add(add(uLow,multiply(uMid,2)),multiply(uHigh,4)),add(add(vLow,multiply(vMid,2)),multiply(vHigh,4)));

    string e=multiply3(substract(add(uLow,multiply(uHigh,4)),multiply(uMid,2)),substract(add(vLow,multiply(vHigh,4)),multiply(vMid,2)));

    string w0=a;

    string w1=divide(substract(multiply(substract(b,c),8),substract(d,e)),12);

    string w2=divide(substract(substract(multiply(add(b,c),16),add(d,e)),multiply(a,30)),24);

    string w3=divide(substract(substract(d,e),multiply(substract(b,c),2)),12);

    string w4=divide(add(substract(add(d,e),multiply(add(b,c),4)),multiply(a,6)),24);

    return add(add(add(add(w4+string(n/3\*4,'0'),w3+string(n/3\*3,'0')),w2+string(n/3\*2,'0')),w1+string(n/3,'0')),w0);

}

2-8

template <typename T>

void change(T *a*[],int *k*,int *left*,int *right*)

{

    if(*k*-*left*-1>*right*-*k*)

    {

        for(int i=0;i<=*right*-*k*;i++)

        {

            T temp=*a*[*k*+i];

*a*[*k*+i]=*a*[*left*+i];

*a*[*left*+i]=temp;

        }

        change(*a*,*k*,*left*+*right*-*k*+1,*right*);

    }

    else if(*k*-*left*-1<*right*-*k*)

    {

        for(int i=0;i<=*k*-1-*left*;i++)

        {

            T temp=*a*[*k*+i];

*a*[*k*+i]=*a*[*left*+i];

*a*[*left*+i]=temp;

        }

        change(*a*,*k*,*left*,*right*-*k*+*left*);

    }

    else

    {

        for(int i=*left*;i<*k*;i++)

        {

            T temp=*a*[i];

*a*[i]=*a*[*k*+i-*left*];

*a*[*k*+i-*left*]=temp;

        }

    }

}

*//每次将数组分为左半和右半*

*//将小的部分交换后递归大的部分，当两边相等时，交换并停止递归*

*//极端情况下，会将数组全部遍历 所用计算时间为O(n)*

*//仅在交换时创建的临时变量 所用辅助空间为O(1)*

2-9

template <typename T>

void reverse(T *a*[],int *left*,int *mid*,int *right*)

{

    for(int i=*left*,j=*mid*-1;i<j;i++,j--)

    {

        T temp=*a*[i];

*a*[i]=*a*[j];

*a*[j]=temp;

    }

    for(int i=*mid*,j=*right*;i<j;i++,j--)

    {

        T temp=*a*[i];

*a*[i]=*a*[j];

*a*[j]=temp;

    }

    for(int i=*left*,j=*right*;i<j;i++,j--)

    {

        T temp=*a*[i];

*a*[i]=*a*[j];

*a*[j]=temp;

    }

}

template <typename T>

void merge(T *a*[],int *k*,int *n*)

{

    int i=0;

    int j=*k*;

    int mid=*k*;

    while(i<*n*&&i<j)

    {

        if(*a*[i]>*a*[j])

        {

            mid=j;

            while(j<*n*&&*a*[j]<*a*[i])

            {

                j++;

            }

            reverse(*a*,i,mid,j-1);

            i+=(j-mid);

        }

        i++;

    }

}

*//使用双指针进行遍历，当左边元素小于右边元素时，左指针i递增*

*//否则右指针j递增至右边元素小于左边元素，然后进行三次翻转操作，使mid至j-1的有序数列排至正确位置*

*//翻转操作时间复杂度为O((mid-1-left)/2)+O((right-mid)/2)+O((right-left)/2)=O(right-left) 空间复杂度O(1)*

*//合并操作时，最坏情况下，遍历所有元素，时间复杂度为O(n) 空间复杂度O(1)*

*//综合起来，时间复杂度为O(n),使用辅助空间为O(1)*

算法实现

2-1

#include<iostream>

#include<fstream>

using namespace std;

int maxcnt=0;

int base=-1;

int n=0;

void swap(int a[],int i,int j)

{

int temp=a[j];

a[j]=a[i];

a[i]=temp;

}

int Partition(int a[],int left,int right,int &cnt)

{

int point=a[left];

cnt=1;

while(left<right)

{

while(a[right]>=point&&left<right)

{

if(a[right]==point)

cnt++;

right--;

}

swap(a,left,right);

while(a[left]<=point&&left<right)

{

if(a[left]==point)

cnt++;

left++;

}

swap(a,left,right);

}

return left;

}

void FindMaxCnt(int a[],int left,int right)//base是当前众数

{

int cnt;

int point=Partition(a,left,right,cnt);

if(cnt>maxcnt)

{

maxcnt=cnt;

base=point;

}

if(point>maxcnt)

{

FindMaxCnt(a,0,point-1);

}

else if(n-point>maxcnt)

{

FindMaxCnt(a,point+1,n-1);

}

}

int a[1000];

void read()

{

ifstream ifs;

ifs.open("input.txt",ios::in);

ifs>>n;

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

{

ifs>>a[i];

}

ifs.close();

}

void write()

{

ofstream ofs;

ofs.open("output.txt",ios::out);

ofs<<a[base]<<endl;

ofs<<maxcnt<<endl;

}

int main()

{

read();

FindMaxCnt(a,0,n-1);

write();

return 0;

}

//使用快排的思路，一边排序的过程中一边计算结果

//在最坏的情况下，重数为1，每个数都遍历一遍，T(n)=O(n)

2-7

int n;

long long cnt=0;

int dp[1000][1000]={0};

long long dfs(int n,int m)

{

if(m==1||m==n)

{

dp[n][m]=1;

return 1;

}

if(dp[n][m]!=0)

return dp[n][m];

dp[n][m]=dfs(n-1,m-1)+dfs(n-1,m)\*m;

return dp[n][m];

}

void read()

{

ifstream ifs;

ifs.open("input.txt",ios::in);

ifs>>n;

ifs.close();

}

void write()

{

ofstream ofs;

ofs.open("output.txt",ios::out);

ofs<<cnt<<endl;

}

int main()

{

read();

for(int i=1;i<=n;i++)

{

cnt+=dfs(n,i);

}

write();

return 0;

}

//m表示分成的份数

//分成m份时的子集数量F(n,m)可以看做

//把前n-1个数分成m-1份，再插入第n个数F(n-1,m-1) 和

//把前n-1个数分成m份，再在这m份集合中选一份插入第n个数F(n-1,m)\*m 之和

//于是得到递推公式F(n,m)=F(n-1,m-1)+F(n-1,m)\*m

//增加dp数组，实现记忆化搜索，相当于遍历下三角矩阵 T(n)=Θ(n^2)

补充题

template <typename T>

T Partition(T *a*[],int *left*,int *right*)

{

    if(*left*==*right*)

    {

        return *a*[*left*];

    }

    int mid=*left*+(*right*-*left*)/2;

    T n1=Partition(*a*,*left*,mid);

    T n2=Partition(*a*,mid+1,*right*);

    if(n1==-1)

    {

        return n2;

    }

    if(n2==-1)

    {

        return n1;

    }

    if(n1==n2)

    {

        return n1;

    }

    else

    {

        int cnt1=0,cnt2=0;

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

        {

            if(*a*[i]==n1)

            {

                ++cnt1;

            }

            else if(*a*[i]==n2)

            {

                ++cnt2;

            }

        }

        if(cnt1>*right*/2-*left*/2)

        {

            return n1;

        }

        else if(cnt2>*right*/2-*left*/2)

        {

            return n2;

        }

        else

        {

            return -1;

        }

    }

}

template <typename T>

bool hasMainElement(T *a*[],int *left*,int *right*)

{

    if(Partition(*a*,*left*,*right*)==-1)

    {

        return false;

    }

    else

    {

        return true;

    }

}

*//将一个有主元素的数组分为两份，其中至少一份有相同的主元素*

*//当元素只有一个时，该元素即为该数组的主元素*

*//时间复杂度可参考快速排序的最好情况*

*//T(n)=O(1) n<=1 || 2T(n/2)+O(n) n>1*

*//时间复杂度为O(nlogn)*

template <typename T>

int Partition(T *a*[],int *left*,int *right*)

{

    int pivot=*a*[*left*];

    while(*left*<*right*)

    {

        while(*left*<*right*&&*a*[*right*]>=pivot)

        {

*right*--;

        }

*a*[*left*]=*a*[*right*];

        while(*left*<*right*&&*a*[*left*]<=pivot)

        {

*left*++;

        }

*a*[*right*]=*a*[*left*];

    }

*a*[*left*]=pivot;

    return *left*;

}

template <typename T>

T getMidNumber(T *a*[],int *left*,int *right*,int *n*)

{

    while (true)

    {

        int pivot=Partition(*a*,*left*,*right*);

        if(*n*&1)

        {

            if(pivot==*n*/2)

            {

                return *a*[pivot];

            }

            else if(pivot>*n*/2)

            {

*right*=pivot-1;

            }

            else

            {

*left*=pivot+1;

            }

        }

        else

        {

            if(pivot==*n*/2||pivot==*n*/2-1)

            {

                return *a*[pivot];

            }

            else if(pivot>*n*/2)

            {

*right*=pivot-1;

            }

            else

            {

*left*=pivot+1;

            }

        }

    }

    return -1;

}

template <typename T>

bool hasMainElement(T *a*[],int *left*,int *right*,int *n*)

{

    T midNumber=getMidNumber(*a*,*left*,*right*,*n*);

    int cnt=0;

    for(int i=*left*;i<=*right*;i++)

    {

        if(*a*[i]==midNumber)

        {

            cnt++;

        }

    }

    return cnt>*n*/2?true:false;

}

*//如果数组存在主元素，则其必为中位数*

*//通过快速排序求中位数，再遍历数组进行验证*

*//最坏情况下时间复杂度为O(n^2) 平均时间复杂度为O(n)*

*//遍历时间复杂度为O(n)*

*//总时间复杂度O(n)+O(n)=O(n);*

*//下面考虑无序的情况*

template <typename T>

T findCandidate(T *a*[],int *n*)

{

    T candidate;

    int cnt=0;

    for(int i=0;i<*n*;i++)

    {

        if(cnt>*n*/2)

        {

            return candidate;

        }

        if(cnt==0)

        {

            candidate=*a*[i];

            cnt++;

        }

        else

        {

            if(*a*[i]==candidate)

            {

                cnt++;

            }

            else

            {

                cnt--;

            }

        }

    }

    return candidate;

}

template <typename T>

bool hasMainElement(T *a*[],int *n*)

{

    T candidate=findCandidate(*a*,*n*);

    int cnt=0;

    for(int i=0;i<*n*;i++)

    {

        if(*a*[i]==candidate)

        {

            cnt++;

        }

    }

    return cnt>*n*/2?true:false;

}

*//与有序时一样想方法求得可能的主元素，再进行验证*

*//如果存在主元素，其数量一定是数组中最多的*

*//第一遍遍历，找出数量最多的元素*

*//第二遍遍历，验证其是主元素*

*//时间复杂度O(n)+O(n)=O(n)*

template<typename T>

struct Candidate

{

    T element;

    int cnt;

};

template<typename T>

struct Candidate findCandidate(T *a*[],int *left*,int *right*)

{

    if(*left*==*right*)

    {

        return {*a*[*left*],1};

    }

    int mid=*left*+(*right*-*left*)/2;

    struct Candidate leftCandidate=findCandidate(*a*,*left*,mid);

    struct Candidate rightCandidate=findCandidate(*a*,mid+1,*right*);

    if(leftCandidate.element==rightCandidate.element)

    {

        return {leftCandidate.element,leftCandidate.cnt+rightCandidate.cnt};

    }

    else

    {

        return leftCandidate.cnt>rightCandidate.cnt?leftCandidate:rightCandidate;

    }

}

template<typename T>

bool hasMainelement(T *a*[],int *left*,int *right*)

{

    struct Candidate res=findCandidate(*a*,*left*,*right*);

    int cnt=0;

    for(int i=*left*;i<=*right*;i++)

    {

        if(*a*[i]==res.element)

        {

            cnt++;

        }

    }

    return cnt>(*right*-*left*)/2;

}

*//分治，但是f(n)=O(1)*

*//T(n)=O(1) n<=1 || 2T(n/2)+O(1)*

*//递归时间复杂度为O(n)*

*//遍历验证主元素时间复杂度为O(n)*

*//总时间复杂度O(n)+O(n)=O(n)*