内部排序 即在内存中装入所有的数据

稳定性：任何两个相对的数据，排序前后的相对位置不发生改变

没有一种排序是任何情况下都表现最好

void X\_Sort(ElementType A[],int N)

最好：顺序T=O() 最坏逆序T=O() 稳定

一、简单排序

冒泡排序：从上到下标记两个相邻的数 ，一趟排序完成之后，最大的一定在最后

void Bubble\_Sort(ElementType A[],int N)

{

for(p=N-1;p>=0;p--){

flag=0;

for( i=0;i<p;i++)//一趟冒泡

{

if(A[i]>A[i+1]) Swap(A[i],A[i+1]);

flag=1;//标识发生了交换

}

if(flag==0)break;//全程无交换

}

}

最好：顺序T=O(n) 最坏逆序T=O(n2) 稳定

插入排序：抓牌

void Insertion\_Sort(ElementType A[],int N)

{

for(p=1;p<N;p++)

{

Tmp=A[p];//摸下第一张牌

for(i=p;i>0&&A[i-1]>Tmp;i--){

A[i]=A[i-1];//移出空位

}

A[i]=Tmp;//新牌落位

}

}

最好：顺序T=O(n) 最坏逆序T=O(n2) 稳定

给 {8,34,51,32,64}分别用冒泡和插入排序 都是9次 ，逆序对是9对，交换两个相邻元素正好消除一个逆序对

时间复杂度下界

逆序对：下标i<j,如果A[i]>A[j]

插入排序：T(N,I)=O(N+I) I=逆序对

如果序列基本有序，则插入排序简单且高效

定理：任意N个不同元素组成的序列平均具有N(N-1)/4个逆序对

定理：任何仅以交换相邻两元素来排序的算法，其平均时间复杂度为Ώ(N2)

这意味着：要提高算法效率，我们必须

每次消去不止一个逆序对

每次交换相隔较远的2个元素

1. 希尔排序 by Donald Shell

利用插入排序的简单，同时克服了每次只交换相邻两个元素的缺点

void Shell\_Sort(ElementType A[],int N)

{

for(p=D;D>0;D/=2)

{

for(p=D;P<N;p++)

{

Tmp=A[p];

for(i=p;i>=D&&A[i-D]>Tmp;i-=D)

{

A[i]=A[i-D]

}

A[i]=Tmp;

}

}

}

最好：顺序T=O() 最坏逆序T=θ(n2) 既是上界也是下界

增量元素不互质，则最小增量可能根本不起作用

1. 堆排序

选择排序

void Selection\_Sort(ElementType A[],int N)

{

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

{

Minposition=ScanForMin(A,i,N-1);

/\*从A[i]到A[N-1]中找最小元，并将其位置献给MinPosition\*/

Swap(A[i],A[MinPosition]);

/\*将未排序部分的最小元换到有序部分的最后位置\*/

}

}

最好：顺序T=O() 最坏逆序T=θ(n2)

如何找到最小元，堆的根就是最小元

堆排序

1. 归并排序

两个有序的序列的归并 指针Aptr Bptr Cptr

若两个子烈一共有N个元素，则归并的时间复杂度是T(N)=O(N) 因为每个元素都比较了一遍

void Merge(ElementType A[],ElementType TmpA[],int L,int R,int RightEnd)

{//L=左边的起始位置，R=右边起始位置，RightType右边终点位置

LeftEnd=R-1;//左边终点位置，假设左右两边挨着

Tmp=L;//存放结果的数组的起始位置

NumElements=RightEnd-L+1;

while(L<=LeftEnd&&R<=RightEnd){

if(A[L<A[R])TmpA[Tmp++]=A[L++];

else TmpA[TmpA++]=A[R++];

}

while(L<=LeftEnd)TmpA[Tmp++]=A[L++];//直接复制左边剩下的

while(R<=RightEnd)TmpA[Tmp++]=A[R++];//直接复制右边剩下的

for(i=0;i<NumElements;i++,RightEnd--)

A[RightEnd]=TmpA[RightEnd];

}

快速排序：比较快 分而治之 先选个主元 每次正好中分 T(N)=O(NlogN)

选主元 随机取rand() 取头中尾的中位数

ElementType Median3(ElementType A[],int Left,int Right)

{

int Center=Lleft+right)/2;

if(A[Left]>A[Center]) Swap(&A[Left],&A[Center]);

if(A[Left]>A[Right]) Swap(&A[Left],&A[Right]);

if(A[Center]>A[Right]) Swap(&A[Center],&A[Right]);

//A[Left]<=A[Center]<=A[Right]

Swap(&A[Center],&A[Right-1]);//将pivot藏到最右边

//只需要考虑A[Left+1]到A[Right-2];

return A[Right-1];//返回pivot

}

每次都能把主元放到正确的位置

如果有元素正好等于pivot怎么办

停下来交换 NlogN

不理它，继续移动指针 坏处每次主元都被放到一个断点处 n2

小规模数据处理

快速排序的问题

用递归，

对小规模的数据(例如N不到100)可能还不如插入排序快

解决方案

当递归规模充分小，则停止递归，直接调用简单排序(例如插入排序)

在程序中定义一个Cutoff的阀值--实践

void Quicksort(ElementType A[],int Left,int Right)

{

if(Cutoff<=Right-Left){

Pivot=Median3(A,Left,Right);//Pivot藏在Right-1的位置

i=Left;j=Right-1;

for(;;){

while(A[++i]<Pivot){}

while(A[--j]>Pivot){}

if(i<j)

Swap(&A[i],&A[j])

else break;

}

Swap(&A[i],&A[Right-1])

Quicksort(A,Left,i-1);

Quicksort(A,i+1,Right);

}

else

Insertion\_Sort(A+Left,Right-Left+1);

}

接口不对

void Quick\_Sort(ElememtType A[],int N)

{

Quicksort(A,0,N-1);

}