DOI:10.13657/j.cnki.gxkxyxb.1982.01.004

排序算法的优化

广西计算中心　　张正铀

摘　　要

通过对算法复杂性为O（N^2）的排序算法的分析，本文构造了尽可能利用排序过程中所产生的信息的两种较优算法，并对其复杂性作出估计。

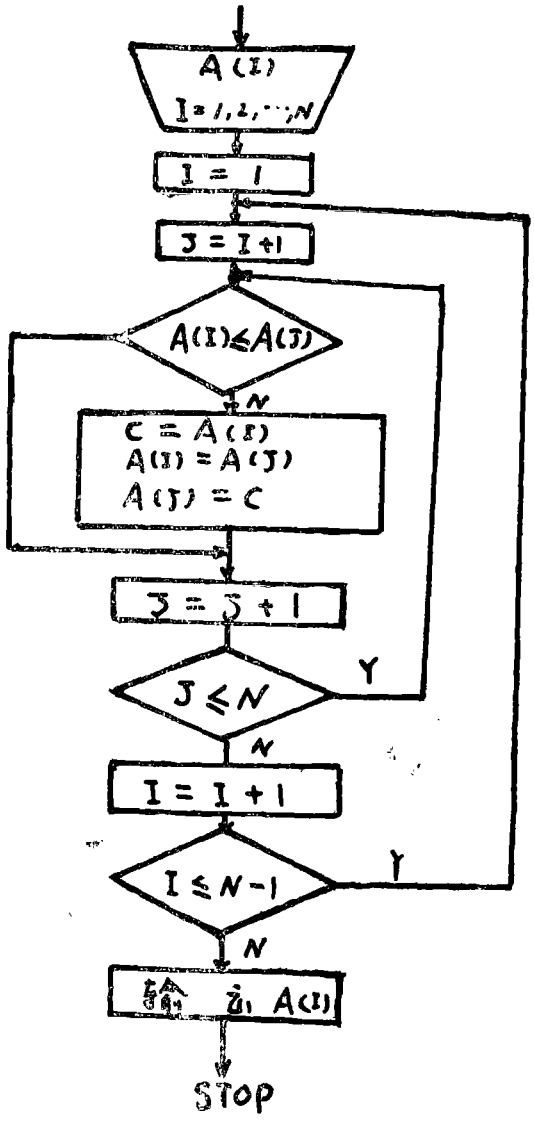
ARNE THESEN在讨论排序算法时指出：数据排序是极其有用的。如果没有排序，要修改磁盘上的数据几乎是不可能的。〔Ⅰ〕在已出版的Kunth的<计算机程序设计技巧>〔Ⅱ〕第三卷中，就以半卷的篇幅讨论六类二十五种排序算法（Sorting）。

算法的优劣通常用算法的时间复杂性和空间复杂性来衡量，即用程序执行时间和占用的存贮空间量来衡量。由问题长度的函数表示的算法执行时间称为该算法的时间复杂性；由问题长度的函数表示的占用空间称为算法的空间复杂性。排序问题的长度为排序量N，所以记其时间复杂性为T(N)；空间复杂性为S（N）。对时间复杂性的分析，有各种方法，其中期望复杂性更可反映算法的动态执行情况，但是期望复杂性很难估计。因此，很多算法都只能给出最好情况时间复杂性和最坏情况时间复杂性。〔ⅢⅨ〕本文在分析排序算法时，也对算法最好和最坏情况下的执行时间和占用空间作出分析，并根据实验数据对其一般情况作出估计。分析中定义T（N）以一次比较操作所用CPU时间为单位。由于排序算法中，主要是比较和赋值操作，而一次赋值所用CPU时间小于一个单位（见附表Ⅰ），因此在算法分析中只列出赋值次数作为参考值。S（N）以存贮一个数据所占用存贮单元数为单位。

对无特殊规律的数据集的排序算法，如“气泡漂浮法”，〔Ⅰ,Ⅵ〕“挑选插入”〔Ⅱ,Ⅴ〕等算法的时间复杂性一般为O（N^2），空间复杂性为O（N）。较好算法的时间复杂性虽然为O（$N\log_2 N$），但所用存贮空间一般较前述算法大（见〔Ⅱ、Ⅲ、Ⅲ、Ⅴ〕）。“气泡漂浮”、“挑选插入”等算法的共同特点是算法构造简单，易于实现，所用存贮单元最少。一般说，这类算法都要求每个参加排序的数据与其余N－1个（N为参加排序数据个数，下同）数据比较，然后决定该数据在所需序列中的位置。

现以“挑选插入”为例（$T(N)=N^2/2$，$S(N)=N$），其框图如下：

本文1982年7月9日收到



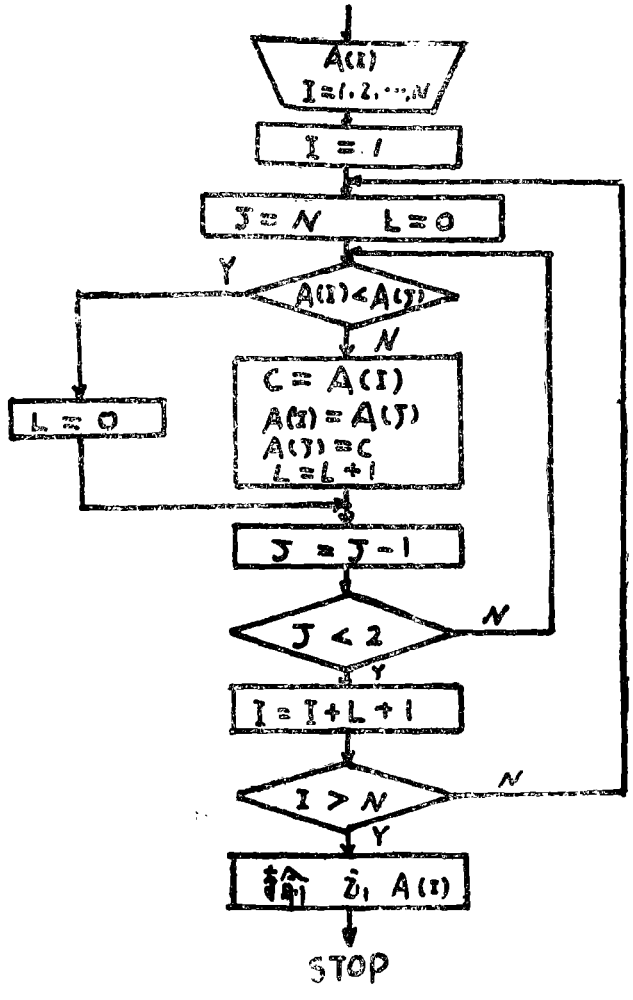
从算法描述可知a(i)必须与其余N－1个元素(即数据)比较后，才能决定a(i)的值应在新序列中的位置。

这类算法的构造，基本上只考虑每个元素与其它元素比较时得到的信息，而没有利用每次比较之前已得到的所有信息。故不能减少比较次数，节省CPU时间。

我们认为，在排序过程中，若能充分利用排序过程中得到的信息，可以有效地改善排序算法的复杂性。基于此思想，本文给出两个较好的排序算法及其复杂性的分析。

一、跳跃式排序算法

1.1 算法框图



输入的a(i)(i=1，2，…)为待排序处理的数据，输出的a(i) (i=1，2，…) 为已按由小到大顺序排列的数据．(下同)

1.2　实现此算法的程序

```
100  INPUT  "NO. ="; N
110  DIM  A1 (N)
120  FOR  I=1  TO  N
130  A1 (I)=INT (RND(x) * 1000)
150  NEXT  I
180  TI$= "000000"
190  I=1
200  IF  I>N  THEN  280
210  M=I+1
220  FOR  J=N  TO  M  STEP  -1
```

```
230 IF A1(J)>A1(I) THEN L=0 : GOTO 250
240 C=A1(J): A1(J)=A1(I): A1(I)=C: L=L+1
250 NEXT J
560 I=I+L+1: L=0
570 GOTO 200
580 T$=TI$
290 FOR I=1 TO N
300 PRINT A1(I);
310 NEXT I
320 PRINT : PRINT "TIME=" ; T$
```

1.3 算法分析

此算法的特点是：在比较过程中，若a(i+j)到a(i+1)(其中j=1,2,…,N−i)连续与第i位上的元素交换，则下一步就跳过j个元素，用a(i+j+1)(设N−(i+j)>1)继续与未成序的N−(i+j)个元素比较。因为，当完成第i−1个元素与N−i+1个元素比较后(i=1,2,…,N)，形成不等式*：a(1)≤a(2)≤…≤a(i−1)≤a(k)(k=i,i+1,……N)。接着用a(i)与未成序的元素a(N)，a(N−1)，……，a(i+1)比较，在a(i)与a(N)到a(i+j+1)比较交换完成后，形成a(i)≤a(k)(k=i+j+1, i+j+2, …, N)。此时，若从i+j位起到i+1位的元素连续与第i位元素交换，即：

a(i)≥a(i+j)，则{c=a(i); a(i=)a(i+j); a(i+j)=c};

a(i)≥a(i+j−1)，则{c=a(i); a(i)=a(i+j−1); a(i+j−1)=c};

⋮ ⋮

a(i)≥a(i+1)，则{c=a(i); a(i)=a(i+1); a(i+1)=c};

于是，形成：a(i)≤a(i+1)≦…≦a(i+j)。又因为当前的a(i+j)已存放不等式*中的a(i)，故有：

a(1)≦…≦a(i)≦…≦a(i+j)≦a(k)(k=i+j+1, i+j+2, …, N)

显然，接着应跳过j个元素，用a(i+j+1)与剩余的N−(i+j+1)个未成序的元素比较即可。

跳跃式排序法最好情况是：当i=1时，从第N位到第2位上的元素均与第1位的元素连续交换，则整个排序只需作N−1次比较操作(另有3*(N−1)次赋值操作)。最坏情况是：在整个排序中，不出现跳跃而必须作$\frac{N(N-1)}{2}$次比较操作(另有3N(N−1)/2−3(N−1)次赋值。)

通过对若干组随机产生的数据分别用“跳跃式排序”和“挑选排序”算法进行排序，可知在一般情况下，“跳跃式排序法”较“挑选排序法”少用23.97%的运行时间(详见附表Ⅰ第4、5、6列)，即较之少约23.97%的比较次数，因此可知跳跃排序算法的复杂性为：

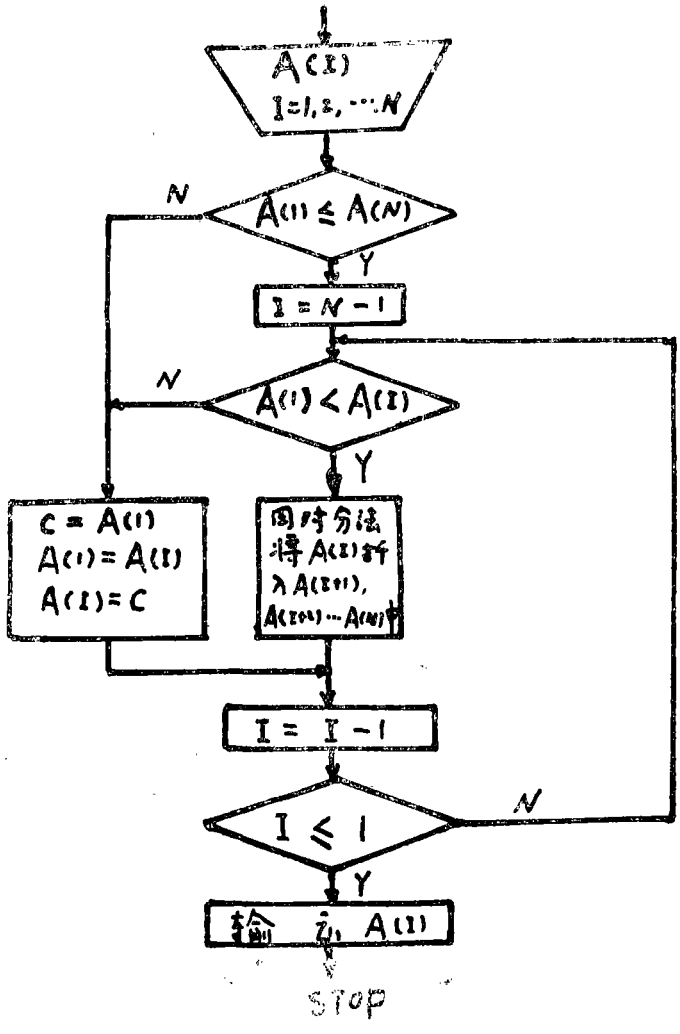
$$T(N)=\frac{N\cdot(N-1)}{2}\cdot(1-0.24)\approx 0.76N^2$$

由于本算法不必多占用存贮单元，所以S(N)≈N。

二、跳跃二分插入排序算法

从本文一给出的跳跃式排序算法分析可知，当a(1)与其余N－1个元素比较时，若a(N)到a(N－j)(j=1,2,…,N－2)连续与第一位上的元素交换后，有(a(N－j)≦a(N－j+1)≦a(N－1+2)……≦a(N))**，若a(N－j－1)＞a(1)(设N－j－1≥2)，就用对分法将a(N－j－1)插入**序列，形成a(N－j－1≦)…≦a(N－j+k)≦…≦a(N)(设原a(N－j－1)的值已插入在a(N－j+k)处，其中k＝－1, 0, 1, …, j)。于是出现了类似a(N)到a(N－j－1)连续与第一位元素交换的情况。当a(1)与其余N－1个元素比较后(即第一位的元素)，交换成最小元素时，排序工作就完成了。

21　算法框图



2.2 实现此算法的程序

```
510 INPUT "NO ="; N%
520 DIM A(N%)
530 FOR I=1 TO N%
540 A(I)=INT(RND(X)*1000)
550 NEXT I
560 T1$="000000": 1N%=N%
570 IF A(1)>A(N1%) THEN C=A(1): A(1)=A(N1%): A(N1%)=C
580 N1%=N1%-1
590 FOR J=N1% TO 2 STEP -1
600 IF A(1)>A(J) THEN C=A(1): A(1)=A(J): A(J)=C : GOTO 740
610 II%=J : NN%=N% : JJ%=0
620 JK%=(II%+NN%+1)/2
630 IF JJ%=JK% THEN 680
635 JJ%=JK%: K%=JJ%
640 IF A(JJ%)<A(J) THEN II%=JJ : GOTO 620
650 NN%=JJ% : GOTO 620
680 IF A(J)<A(K%) THEN K%=K%-1
690 KK%=K%-1 : C=A(J)
700 FOR J1=J TO KK%
710 A(J1)=A(J1+1)
720 NEXT J1
730 A(K%)=C
740 NEXT J
745 T$=T1$
750 FOR I=1 TO N%
760 PRINT-A(I);
770 NEXT I
780 PRINT: PRINT "TIME=" ; TS
```

2.3 算法分析

此算法最好情况是从a(N)到a(2)均连续与第一位置上的元素交换，不出现a(1)<a(i)(i=N, N-1, …, 2,)的情况，则总的比较为N-1次(及3·(N-1)次赋值)。最坏的情况是当i=N, …, 3, 2, 时，均有a(1)<a(i)，则比较次数为：

$$(N-1)+\log_2 2+\log_2 3+\cdots+\log_2(N-1)=(N-1)+\log_2(N-1)!$$

(若a(i)均需插在a(N)位置上，则赋值次数为N·(N-1)/2)。

由于该算法也利用了跳跃的技巧，故在一般情况下减少的比较次数也约为23.97%，则“跳跃二分扦入”排序算法的复杂性为：

$T(N)=0.76((N-1)+\log_2(N-1))!\approx N+0.76N\log_2 N$

此算法不必多占用工作单元，故其空间复杂性$S(N)\approx N$。

诚然，此算法由于节省了内存量及减少了比较次数，而付出了数据迁移较频繁的代价。

本算法与目前公认较好的$T(N)\approx N\log_2 N$型排序算法比较（如“合并整队”等排序算法[Ⅳ、Ⅴ]）：

设“跳跃二分法”时间复杂性为T_1；

“合并整队法”时间复杂性为T_0；

有$T_1=N+0.76\log_2 N!\approx N+0.76N\log_2 N$

$=N(1+0.76\log_2 N)$

$T_0=N\log_2 N$

则$(T_0-T_1)/T_0=1-(1+0.76\log_2 N)/\log_2 N$

$=0.24-1/\log_2 N$

当$N>2^{(1/0.24)}$，即$N>18$时有$(T_0-T_1)/T_0>0$，所以当$N>18$时，“跳跃二分插入法”比“合并整队”等算法好。另外，“合并整队”等$N\log_2 N$型算法还需增加一定的内存量建栈或作别的形式的中间工作单元，而“跳跃二分扦入法”的$S(N)\approx N$，也较它们为好。

三、小　结

我们在应用软件的设计过程中，曾多次在不同的计算机上应用“挑选插入法”、“跳跃排序法”及“跳跃二分法”，得到的结果与本文分析的结论相符。附表Ⅱ列出这三种算法在COMMODORE 4000型机上应用的结果。附表Ⅲ列出“挑选插入”、“跳跃排序”及“跳跃二分排序”算法分析结果。附表Ⅳ列出了40次运行时间对比情况。

综上所述，在排序过程中，合理利用其中的信息，可以有效地改善算法复杂性。

参加算法分析及试验的还有刘连芳同志。

罗海鹏同志曾对本文提出过宝贵意见，在此表示感谢。

附表Ⅰ：

赋值操作、比较操作所用CPU时间*

赋　值　操　作	14×10^{-4}秒／次
比　较　操　作	77×10^{-4}秒／次

* 在Commodore　4040机上运行结果

附表Ⅱ：

算法名称	所用CPU时间*		
	N=10	N=100	N=500
挑选插入**	5″	1′21″	33′56″
跳跃排序	6″	1′09″	28′16″
跳跃二分	6″	54″	14′15″

* 包括从磁盘读入数据原用的时间

** 实现此算法的程序如下：

```
100 INPUT "NO. =" ; N
110 DIM A2(N)
120 FOR I=1 TO N
130 A2(I)=INT ( RND(X) * 1000 )
140 NEXT I
160 TI$= "000000"
170 FOR I=1 TO N-1
180 FOR J=I+1 TO N
190 IF A2(I)>A2(J) THEN C=A2 ( I ) : A2 ( I ) =A2 ( J ) : A2 ( J ) =C
200 NEXT J,I
210 T$=TI$
220 FOR I=1 TO N
230 PRINT A2 ( I ) ;
240 NEXT I
250 PRINT : PRINT "TIME=" ; T$
```

附表Ⅲ

算法名称	T(N)			S(N)
	最好情况	最坏情况	一般情况	
挑选插入	N^2	N^2	N^2	N
跳跃排序	N−1	N^2	$0.76N^2$	N
跳跃二分	N−1	$N+N\log_2 N$	$N+0.76N\log_2 N$	N

附表Ⅲ

序号	N	$T_1(N)$（跳跃二分）	$T_2(N)$（跳跃）	$T_3(N)$（挑选）	$\frac{T_3-T_2}{T_3}$	序号	N	$T_1(N)$（跳跃二分）	$T_2(N)$（跳跃）	$T_3(N)$（挑选）	$\frac{T_3-T_2}{T_3}$
1	100	47″	61″	78″	0.2179	21	300	301″	506″	681″	0.2569
2	100	44″	65″	78″	0.1667	22	300	308″	528″	683″	0.2632
3	100	48″	60″	84″	0.2857	23	300	305″	515″	699″	0.2551
4	100	42″	55″	77″	0.2857	24	300	301″	508″	682″	0.238
5	100	45″	54″	78″	0.3077	25	300	302″	525″	689″	0.2636
6	100	44″	52″	76″	0.315	26	400	506″	891″	1210″	0.2457
7	100	46″	61″	77″	0.2077	27	400	517″	912″	1209″	0.2795
8	100	47″	65″	80″	0.1875	28	400	507″	884″	1227″	0.2678
9	100	45″	60″	76″	0.2105	29	400	518″	913″	1247″	0.25
10	100	45″	60″	77″	0.2208	30	400	518″	930″	1240″	0.234
11	200	141″	229″	303″	0.2442	31	400	513″	933″	1218″	0.2522
12	200	156″	244″	319″	0.2351	32	500	781″	1441″	1927″	0.2285
13	200	149″	238″	311″	0.2347	33	500	806″	1496″	1939″	0.2328
14	200	148″	230″	314″	0.2675	34	600	1078″	2084″	2716″	0.254
15	200	140″	227″	300″	0.2433	35	700	1501″	2810″	3769″	0.2247
16	200	145″	244″	306″	0.2026	36	800	1960″	3724″	4802″	0.2216
17	200	150″	235″	306″	0.232″	37	800	1929″	3736″	4802″	0.2154
18	200	149″	241″	312″	0.2276	38	900	2407″	4759″	6065″	0.2243
19	200	158″	244″	313″	0.2204	39	1000	2873″	5714″	7366″	0,187
20	200	153″	233″	313″	0.2556	40	1500	6320″	13241″	16286″	0.2397

$\frac{T_3-T_2}{T_3}$ 的平均值为0.2397

参考文献：

〔Ⅰ〕 ARNE THESEN, "COMPUTER METHODS IN OPERATIONS RESEARCH" P39～58, 1978。

〔Ⅱ〕 D.E.Khuth, "THE ART OF COMPUTER PROGRAMMING" (Volume 3/sorting and searching) P1～379, 1973。

〔Ⅲ〕 EDWARD M·REINGOLD、JURG NIEVERELT、NARSINGHDEO, "Combinatorial Algorithms: Theory and Practice" P278～310, 1977。

〔Ⅳ〕 卢开澄"组合数学与组合算法" P13, P259～267(清华大学) 1980

〔Ⅴ〕 SAKTI P·GHOSH, "Data Base Organization for Data Management" P99～103

〔Ⅵ〕 文瑜、朱其亮，"小型电子计算机——基本原理与程序初步" P469 (人民邮电出版社)