データソートのアルゴリズム

著者	久保 忠延
雑誌名	長野県短期大学紀要
巻	42
ページ	1-6
発行年	1987-12
URL	http://id.nii.ac.jp/1118/00000583/

データソートのアルゴリズム

久保忠延

1 緒言

パーソナルコンピュータ（パソコン）は年々その性能が向上し，広く普及してきている。このように身近になったパソコンを用いて成績処理等を行うとき，得点の順位づけや並べかえをする必要がしばしば生じる。このような目的のために，これまで多くのアルゴリズムが考えられている[1]。先にそれらのアルゴリズムの特徴を調べ報告したが[2]，各種のアルゴリズムの中で度数ソート法が最もソートに要する時間が短く，またデータ構造の影響もうけにくいことがわかった。しかし，度数ソート法の場合には，本質的に取扱える数値が整数であるという制約がある。成績処理においては，平均値などを計算した際に生じる小数値も扱うことがあるため，この場合には度数ソート法をそのままでは使うことができない。そこで，このような小数値を含む数値データを扱え，しかも度数ソート法の長所を取り入れたアルゴリズムを考えた。

このアルゴリズムは度数ソート法を基本にし，これにもう一つのソートアルゴリズムを組み合せたものである。本報ではこのアルゴリズムとその特徴について調べた結果を報告する。

なお，このアルゴリズムによりソートを実行させる具体的なプログラム言語としては，パソコンで広く用いられているBASICを使った。

2 ソートのアルゴリズム

度数ソート法により小数値を扱う最も簡単な方法は，データがすべて整数となるように数値データを10の累乗倍してその整数化したデータを処理することである。しかし，この方法ではたとえデータ数が少ない場合でも，整数化したデータの最大の値と同じ大きさの配列を必要とするため，パソコンの記憶容量の点で限界が生じやすい。最近ではパソコンの記憶容量が以前よりもかなり増大しているとはいえ配列のとれる桁数は4桁程度であり，少し旧型のものでは3桁程度のものが多い。したがってこの方法によって扱える整数化した数値も3～4桁と限定される。このような記憶容量の限界に対処するために，度数ソート法を二段階に用いる方法および度数ソート法と他のソート法を組み合せた方法を考えた。

前者の方法（以下これを「二段度数ソート法」と呼ぶことにする）の処理手順を図1に示した。この方法では，まずデータの整数部のみの大小にもとづき度数ソート法で並べかえを行い，つぎに同一の整数部をもつデータの間でそれらのデータの小数部の大小により並べかえを行う。この結果全体のデータが大きい順に並ぶというものである。具体的なプログラムの例をリスト1に示した。

後者の方法では，度数ソート法と組合せるもう一つのソート法に何を用いるかが問題であるが，

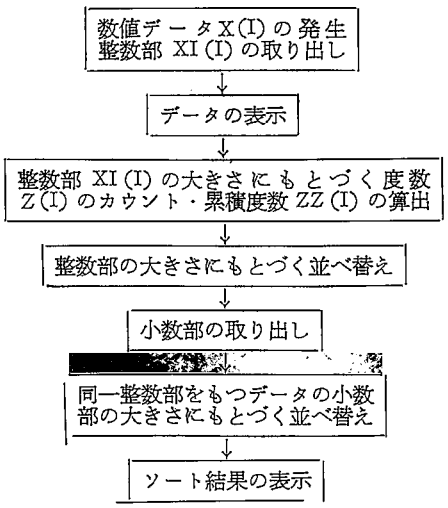

図1　二段度数ソート法の処理手順

```
100 INPUT "DATA SU N = ";N
110 INPUT "DATA MAX VAL Q = ";Q
120 INPUT "PLACE DECIMAL NP = ";NP : SP=10^NP
130 DIM X(N),XI(N),XD(N),XS(N),XSS(N),Z(Q+1),ZZ(Q+1)
140 DIM ZD(SP+1),S(N),SS(N),T(N),TT(N)
150 '
160 FOR I=1 TO N
170 X(I)=(INT(Q*RND(1)*SP))/SP:XI(I)=INT(X(I))
180 NEXT I
190 FOR I=1 TO N:PRINT USING"####.###";X(I);:NEXT I
200 '
210 TIME$="00:00:00"
220 FOR I=0 TO Q+1 : Z(I)=0:ZZ(I)=0:NEXT I
230 FOR I=1 TO N:K=XI(I):Z(K)=Z(K)+1: NEXT I
240 FOR K=Q TO 0 STEP -1:ZZ(K)=Z(K):NEXT K
250 FOR J=Q TO 0 STEP -1:Z(J)=Z(J)+Z(J+1):NEXT J
260 FOR I=1 TO N
270 W=Z(XI(I)+1)+1
280 IF S(W)=0 THEN S(W)=I:T(W)=W+1:GOTO 300
290 S(T(W))=I:T(W)=T(W)+1
300 NEXT I
310 NN=0
320 FOR I=1 TO N:XS(I)=X(S(I)):NEXT I
330 FOR I=0 TO SP-1 :ZD(I)=0:NEXT I
340 FOR I=Q TO 0 STEP -1
350 IF ZZ(I)=0 THEN 510
360 IF ZZ(I)=1 THEN NN=NN+1:XSS(NN)=XS(NN):GOTO 510
370 FOR J=1 TO ZZ(I)
380 XD(J)=(XS(NN+J)-INT(XS(NN+J)))*SP
390 KK=XD(J):ZD(KK)=ZD(KK)+1
400 NEXT J
410 FOR J=SP-1 TO 0 STEP -1:ZD(J)=ZD(J)+ZD(J+1):NEXT J
420 FOR J=1 TO ZZ(I)
430 WW=ZD(XD(J)+1)+1
440 IF SS(WW)=0 THEN SS(WW)=J :TT(WW)=WW+1:GOTO 460
450 SS(TT(WW))=J:TT(WW)=TT(WW)+1
460 NEXT J
470 FOR J=1 TO ZZ(I):XSS(J+NN)=XS(SS(J)+NN):NEXT J
480 FOR J=0 TO SP :ZD(J)=0:NEXT J
490 FOR J=1 TO ZZ(I):SS(J)=0:NEXT J
500 NN=NN+ZZ(I)
510 NEXT I
520 LPRINT TIME$
530 '
540 FOR I=1 TO N:PRINT USING"####.###"; XSS(I);:NEXT I
550 END
```

リスト1　二段度数ソートのプログラム

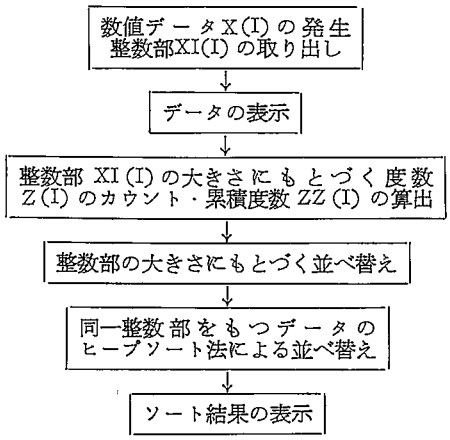

図2　度数・ヒープソート法の処理手順

```
100 INPUT "DATA SU N = ";N
110 INPUT "DATA MAX VAL Q = ";Q
120 INPUT "PLACE DECIMAL NP = ";NP : SP=10^NP
130 DIM X(N),XI(N),XS(N),Z(Q+1),ZZ(Q+1),S(N),T(N)
140 '
150 FOR I=1 TO N
160 X(I)=(INT(Q*RND(1)*SP))/SP:XI(I)=INT(X(I))
170 NEXT I
180 FOR I=1 TO N: PRINT USING"####.###";X(I);:NEXT I
190 '
200 TIME$="00:00:00"
210 FOR I=0 TO Q+1 : Z(I)=0:ZZ(I)=0:NEXT I
220 FOR I=1 TO N:K=XI(I):Z(K)=Z(K)+1: NEXT I
230 FOR K=Q TO 0 STEP -1:ZZ(K)=Z(K):NEXT K
240 FOR J=Q TO 0 STEP -1:Z(J)=Z(J)+Z(J+1):NEXT J
250 FOR I=1 TO N
260 W=Z(XI(I)+1)+1
270 IF S(W)=0 THEN S(W)=I:T(W)=W+1:GOTO 290
280 S(T(W))=I:T(W)=T(W)+1
290 NEXT I
300 FOR I=1 TO N:XS(I)=X(S(I)):NEXT I
310 FOR I=Q TO 0 STEP -1
320 M=ZZ(I):Z=Z(I+1)
330 IF M<2  THEN 540
340 FOR II=Z+2 TO Z+M
350 J=II
360 D=INT(Z+(J-Z)/2)
370 IF XS(D)<=XS(J) THEN 410
380 SWAP XS(D),XS(J)
390 J=D
400 IF J>Z+1 THEN 360
410 NEXT II
420 II=Z+M
430 FOR K=Z+M TO Z+1 STEP -1
440 SWAP XS(Z+1),XS(II)
450 II=II-1:J=Z+1:S=Z+2
460 IF S>II THEN 530
470 IF S=II THEN 490
480 IF XS(S+1)<XS(S) THEN S=S+1
490 IF XS(J)<=XS(S) THEN 530
500 SWAP XS(J),XS(S)
510 J=S:S=(J-Z)*2+Z
520 GOTO 460
530 NEXT K
540 NEXT I
550 LPRINT TIME$
560 '
570 FOR I=1 TO N:PRINT USING"####.###"; XS(I);:NEXT I
580 END
```

リスト2　度数・ヒープソートのプログラム

ここでは文献(2)～(4)等から知られるように，各種ソート法の中でデータ構造の影響を受けにくくかつ高速なソート法であるヒープソート法を選んだ。以下このソート法を「度数・ヒープソート法」と呼ぶことにする。

度数・ヒープソート法による処理手順を図２に示す。この方法では，先に述べた二段度数ソート法と同様，まずデータの整数部の大小にもとづき度数ソート法で並べかえを行い，つぎに同一の整数部をもったデータ間で小数部を含めたデータの大小にもとづきヒープソート法によって並べかえを行う。プログラムの例をリスト２に示した。

以上述べた二つの方法のいずれの場合も，数値データとしてはパソコンにより発生させた乱数を所定の条件に合うように加工して用いている。

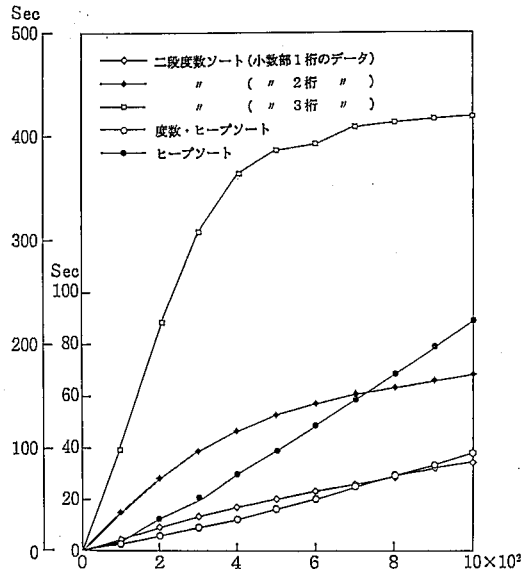

図４　データ処理時間（その２）

3　結果と考察

前項で述べた二つの方法により数値データを大きい順に並べたときに要した時間とデータ数の関係を図３，図４に示した。

図には比較のため，ヒープソート法による処理時間の結果も示してある。この場合の数値データとしては，成績処理を念頭において，整数部の最大値を100としている。そして小数点以下の桁数としては小数第１位，第２位および第３位までのものの３種類を与えている。

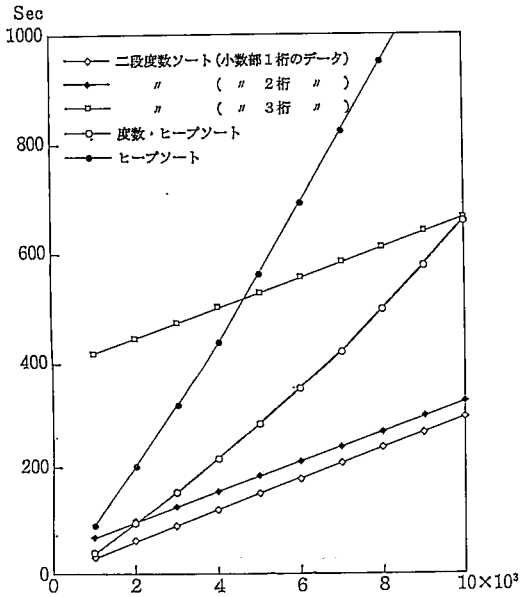

図３　データ処理時間（その１）

図３はデータ数1000以上の場合について処理に要した時間を縦軸（単位は秒）に，データ数を横軸にとってその関係を示したものである。

二段度数ソート法の場合には，データの桁数が並べかえの時間に大きく影響する。小数点以下の桁数（すなわち，数値の全体の桁数）が１つ増すごとに度数をカウントするループの配列の桁数も１つずつ大きくなるため，データ数とは別のファクターとしてこの桁数は処理時間に影響を及ぼす。二段度数ソート法の場合，処理対象の中に含まれる数値の整数部の桁数と小数部の桁数の和が同一であれば，同じデータ数を処理するのに要する時間にほとんど差が出ない。たとえば，整数部が２桁で小数部が２桁の数値データの集りと整数部が１桁で小数部が３桁のデータの集り（いずれも桁数の合計が４）を処理する時間はほとんど同じである。

二段度数ソート法の場合，データ数がある大きさ以上では処理時間はデータ数の増加に伴って直線的に増加するようになるが（図３），データ数が少ない場合にはこの直線関係が破れ，図４に示すようにデータ数が減少すると処理時間が急減する。これはデータ数がある程度以下になると度数をカウントするための配列が空になるところが多くなり，この配列の処理が不要になるために生じると考えられる。なお，図４で二段度数ソート法による小数部３桁のデータの処理時間を示す縦軸のスケールは他の場合の4/10に縮めてある。

度数・ヒープソート法による処理時間の場合には，そのアルゴリズムから当然のことながら，数値データが小数点以下何位までかということには無関係に，いずれのデータの場合にも処理時間をプロットした点は同一曲線上に乗る。図３ではデータ数の増加に伴う処理時間の増加の割合は，度数・ヒープソートの方が二段度数ソート法の場合より大きい。

二段度数ソート法の場合には整数部の桁数と小数部の桁数の和が同じであればそれらの各々のデータを処理する時間にほとんど差の出なかったのに比べ，度数・ヒープソート法の場合の処理時間は先に述べたように小数部の桁数には無関係でデータの整数部の桁数にのみ依存している。図５に整数部の桁数が１桁，２桁，３桁のデータを度数・ヒープソート法で処理したときの結果を示す。この結果から，度数・ヒープソート法によりデータを処理する場合は度数カウント用の配列の大きさをできるだけ大きくとり，この配列の大きさの許す範囲内でデータの整数部ができるだけ大きくなるようにもとのデータの小数点の位置を移動したものについて処理を行った方が有利であることがわかる。図には比較のため二段度数ソート法によるデータの処理時間の結果も示した。

図３～図５に示したグラフの各曲線はいくつかの点で交差しており，この交差した点を境に各ソート法の優劣が逆転する。

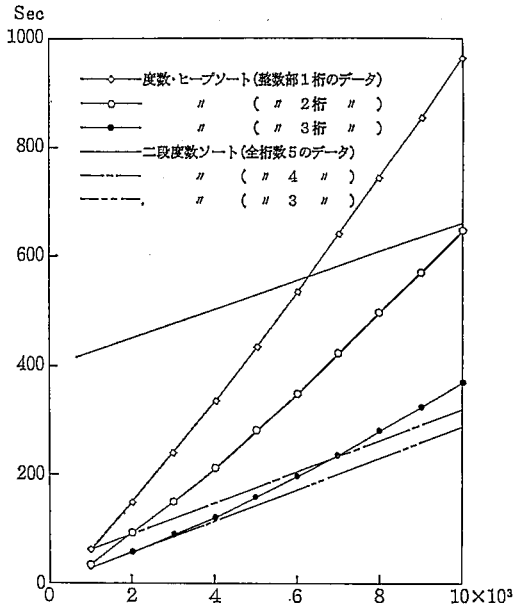

図５　データ処理時間（その３）

以上のことから全般的につぎのようなことが言える。すなわち，二段度数ソート法の処理時間を示す直線の勾配は度数・ヒープソート法の曲線の勾配より小さいので，データ数が充分大きくなれば二段度数ソート法が必ず有利となる。しかし，データの桁数の点に関しては度数・ヒープソート法の方が制限がゆるく，データの小数点移動を行えば充分大きな桁数のデータの処理が行える。

なお，本研究で使用したパソコンは PC—9801 Vm21である。

4　結　言

本研究では，度数ソート法の優れた特徴を生かしつつ，かつ小数値データの処理の行えるソートアルゴリズムを考え，このアルゴリズムの特性をパソコンを使って実験的に調べた。

その結果，成績処理等で現われる数値の範囲を充分にカバーできる桁数のデータに対してこのソート法は他の高速ソート法より広範囲の条件下で優れた特性を有することがわかった。更に，このソート法は小数値データの整数化による単純な度

数ソート法と比べると処理対象とする数値の桁数の制約がはるかにゆるいといった点でも優れている。

文　献

1）涌井良幸：高速ソートプログラミング，誠文堂新光社 10（1984）
2）久保忠延：長野県短期大学紀要，401（1985）
3）野崎昭弘：計算機数学，共立出版 176（1984）
4）大河内・古賀・鍛島：基礎電子計算機，実教出版 129（1982）