さらに、ここで収縮率をより小さな幅で変えていくとより速度の向上がはかられる。例えば収縮率を1.3から始めて0.05だけ増やしていくと、上の場合なら7秒程度で処理してしまう。収縮率を固定せずにデータに合わせて変えることと、実行時に少しずつ変えることが重要であると思われる。

5．速度の比較

以上述べたソートのプログラムを作り、その速度を測定したものを表9に示す。ただし、時間の単位はすべて秒である。紙面の都合上、プログラムは示さない。データは降順に500個並べたものを使い、それを昇順に並べることを目標とした。このようなものはデータ移動が最も多いはずである。なお、1000個の場合は別に測定したのでここでは述べない。

bubble sort	3分13秒	(193秒)
Stephen／Richardの方法	6分45秒	(405秒)
quick sort	5～6秒	(C言語で作れば0.5秒程度であると思われる)
comb sort	12秒	

表9　速　度　の　比　較

comb sortの場合、収縮率を変えると7秒に縮まるが、ここではそれを1.3として測定した。この結果からcomb sortは十分実用になると思われる。今後さらに検討を加える予定である。

6．まとめ

既に述べたように、comb sortはbubble sortよりも十分に速く、実用になることが期待できるために、そのテクニックをより詳しく調べる必要がある。太田氏によれば「クイックソートより速いことがある」ので、工夫次第によってはさらによい方法があるかも知れない。例えば構造体や連結リストなどを使い、方法を工夫すればより改善されるであろう。

7．参考文献

1．Stephen Lacey & Richard Box：バブル・ソートが簡単な手直しで劇的に速くなる
日経バイト　1991.11,pp.305－312　日経BP社

2．岡本　茂、加藤木　和夫：PADによるプログラム技法　1990.4、啓学出版

3．岡本、仙波、中村、高橋：パソコン用語事典　'92－'93年版

4．野崎　昭宏：bit,1986.8,pp.92－99

5．N88－日本語BASIC(86)(Ver6.0)リファレンスマニュアル、1988、日本電気株式会社

使用したコンピュータはPC−9800である［文献５］)。

収　縮　率	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
コム　ソート	24	15	11	9	8	8	8	8	7	15
収　縮　率	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0
コム　ソート	40	66	88	109	124	145	154	168	179	190

表７　コムソートにおける速さ

次にcomb sortでデータがどのように変わるかを図示する（図２)。

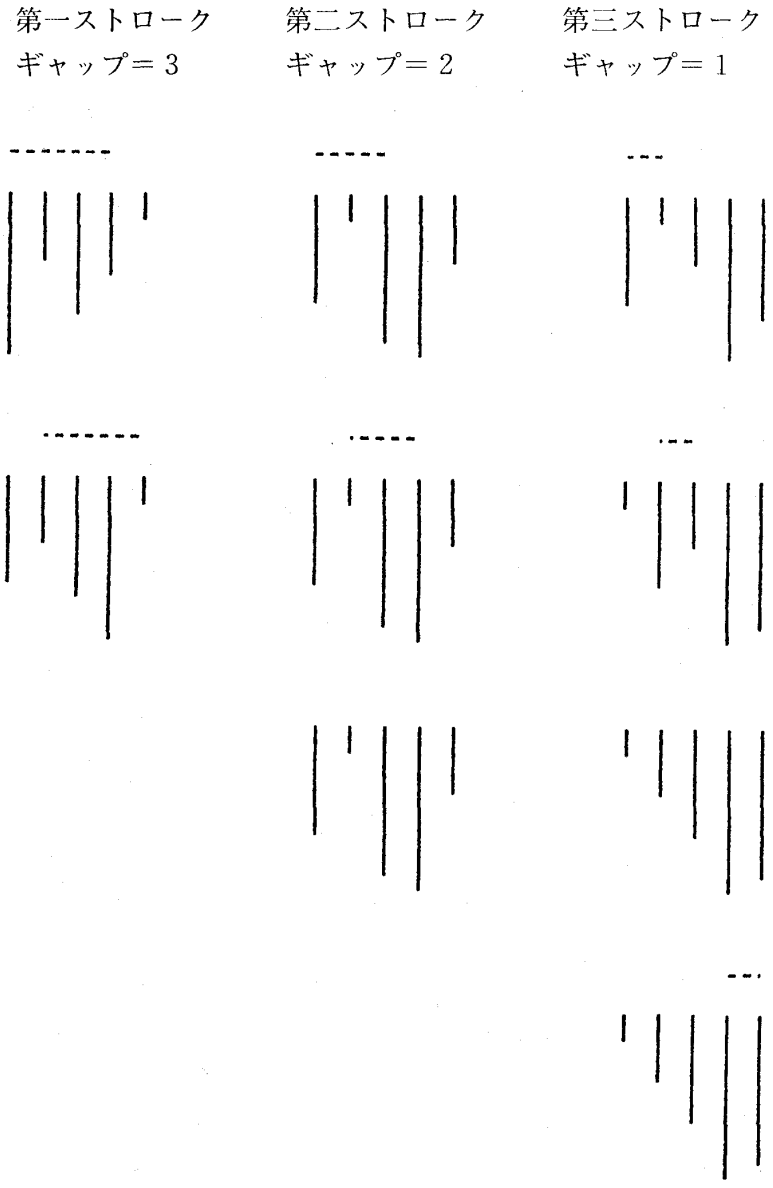


図２　Comb Sortにおけるデータの移動

数を記録しておく。この回数が0なら要素における位置が全く変わらないということになるから、これで終了の判定が半分済む。プログラムは次の表5のようになる［文献1］。

```
200  '
210  ' コムソート
220  '
230  GAP=N
240  '
250  GAP=INT(GAP/1.3)
260  IF GAP<1 THEN GAP=1
270  SW=0
280  FOR I=1 TO N-GAP
290  J=I+GAP
300  IF T(I)<T(J) THEN SWAP T(K),
     T(J) : SW=SW+1
310  NEXT I
320  IF SW< >0 OR GAP<1 THEN 250
```

表5　comb sortのプログラム

実際にはquick sortと同様にこれに適当なドライバ・ルーチンをつければよい。この1.3という数は1.1から1.45の間で変化させて得た数値である。このテストには1000個程度のデータを用い、その結果はグラフで描かれているが、これによれば1.25から1.3の範囲でソート時間が少なくなる。これを更に改善するため、収縮率1.3の付近でギャップを調べ、ギャップが9や10のときは11とすればよいとStephen／Richardは結論している。それを組み込んだプログラムは、行260を次のようにおきかえればよい。

```
260  IF GAP<1 THEN GAP=1:GOTO 270
265  IF GAP=9 OR GAP=10
     THEN GAP=11
```

表6　追加ルーチン

これを彼らはcomb sort11といい、それによって速度が多少改善されること、場合によってはquick sortより高速になることを確かめた。我々の追試ではそのようにならなかったが、太田洋氏はTurbo C＋＋で成功している［文献1］。

4．1　収縮率の改善について

comb sortにおいては亀が飛び跳ねることが重要で、表6の265の行は平均化に貢献していると思われる。我々は次のように実験を行なった。まずcomb sortのプログラムで収縮率を1.1から3.0まで0.1のステップで変え、時間を測定した。その結果が次の表7である。comb sortの原形では収縮率を変えると速度も変わるということと、その様子がよくわかる。これは単に「亀の飛び跳ね」によるものである。ここでは収縮率を1.4から1.9とするのが適当なようである。一方、comb sort11では常に12秒でてき、時間の変化がなかった。ここでは亀をうまく制御していることになろう。これから結論するのは乱暴だが、要するに「収縮率には1.3が適当だ」というStephen／Richardの結論は少し問題があり、コンピュータの選択とコンパイラの選択も関係すると考えられる。

例えばPC－9800では「収縮率を1.9くらいにする」方が適当であると思われる。（表7における時間の単位は秒、データの個数は500とした。

これを再帰的に繰り返すか、またはスタックを使って繰り返せばよい。C言語によるプログラムを表4に示す［文献2］。

```
void qsort(left, right)
int t[ ], left, right;
{
int i,j,x,tmp;
x=0;
for(i=left;i<=right;i++)x=x+t[i];
x=x/(float)(right-left+1);  /*  基準値の決定  */
/*  ここから実質的なループが始まる  */
i=left;
j=right;
do
   {
      while(t[i]<x)
            {
            i++;
            if(i>right) break;
            }
      while(t[j]>x)j--;
      if(i<=j)
            {
            tmp=t[i];        /*  これはswapである  */
            t[i]=t[j];
            t[j]=tmp;
            i++;             /*  i の更新  */
            j--;             /*  j の更新  */
            }
   } while(i<=j);
   if(left<j) qsort(left, j);
   if(i<right) qsort(i, right) ;
}
```

表4　C言語によるquick sortのプログラム

4．comb sortおよびStephen／Richardの結果について

2．で述べたように、亀の動く幅を適当に大きくすれば全体として移動回数が減るはずなので、それだけソートが高速になることが期待できよう。即ち「適当に隣り合った要素の比較」が問題である。そこでどのくらい離れたらよいかという、ギャップの設定が問題になる。Stephen／Richardはこの適当なギャップを求め、その結果、「ギャップを適当に設定すると亀が飛び跳ね、高速ソートができる」ことを発見した。これを追試した結果と併せて報告する。

最初のストロークにおけるギャップは（配列の要素数）／1.3とする。この1.3という数を収縮率といい、一般にはそのべき乗である。即ち次には前回のギャップを更に収縮率で割り、商が1を下回ったら1とする。ストローク中における要素の位置が全く変わらなくなったら、ソートを終わる。この1.3という数は経験的に得られたものと思われる。

比較するものは、J=I+GAPとすると、T(I)とT(J)である。大きさが適当でないときは、スワップするものとしてその回

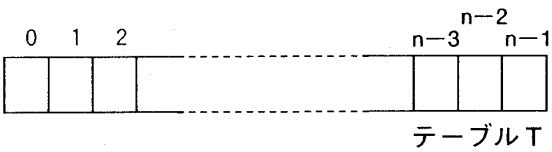
よるものを示す。なお、このプログラムの実質的な作者は仙波一郎氏［文献3］で、ここではスタックを使っている。実際にはこれに適当なドライバ・ルーチンをつければよい。このプログラムを表2に示す。

これは高速であり、現在の所これより速いソート法はないとされる。データ数が多いときは特に優れており、200個以下のデータならbubble sortでやってもそれほど違わない。Tの要素数はNとして、比較回数は$N \times \log_2 N$とされているが、実際にはもう少し時間を要する。配列の状態でこの時間が変化し、bubble sortとさほど変わらないこともある。その他、heap sortは余計なストレージも食わず、比較回数は$2N \times \log_2 N$で、しかもデータの配置状態にあまり影響されない。また、我々の用いた番号法ではbubble sortと似た方法を使ったが、データの移動が全くないので、基本的にはかなり速い。まだ実験を行なっていないが、これに適当な高速ソート法を併用すると実際はさらに高速になるかも知れない。

quick sortはC言語を使うとより高速で、しかも再帰法が使えるためプログラムもスマートである。その基本的アイデアを表3に示す。

```
400  DIM T(1000), L(100)' R(100)
410  '
420  ' クイックソート
430  '
440  ' L(K)は配列の左端，R(K)は配列の右端を示す。
450  ' Xとして中央値を使う。
460  '
470  L(1)=1：R(1)=N：K=1
480  '
490  WHILE K>0
500      I=L(K)：J=R(K)：K=K-1
510  '
520  ' 配列{T(I),...,T(J)}を分割する。
530  '
540      X=T (INT((I+J)/2)) ' Xの決定
550      WHILE T(J)>X
560          WHILE T(I)<X
570              I=I+1
580          WEND
590          J=J-1
600      WEND
610  '
620  ' T(I)とT(J)の交換
630  '
640      IF I<J THEN SWAP T(I)、T(J)：I=I+1：
         J=J-1：GOTO 550
650      LK(K+1)：RK=R(K+1)
660      IF I<RK THEN K=K+1：L(K)=I：R(K)=RK
670      IF LK<J THEN k=K+1：L(K)=LK：R(K)=J
680  WEND
690  '
700  ' これで整列が終わった。
710  '
```

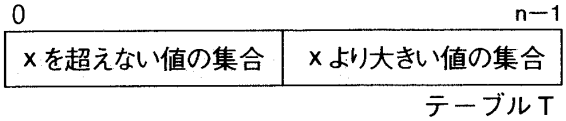
表2　quick sortのプログラム



テーブルT

手順0．i=0、j=n−1とする。
手順1．t［i］とxを比較し、x＞t［i］ならi←i＋1；これを繰り返す。
手順2．t［j］とxを比較し、x＜t［j］ならj←j－1；これを繰り返す。
手順3．t［i］とt［j］を交換し、i←i＋1、j←j－1もしi≦jなら1へ。
手順4．i＞jとなったら基準値を変える。

これで次のようになっている。



テーブルT

表3　quick sortの基本アルゴリズム

```
200  '
210  ' バブルソート
220  '
230  FOR I=2 TO N
240    FOR J=N TO I STEP-1
250      IF T(J-1)>T(J)  THEN SWAP T ( J -1),
       T(J)
260  NEXT J
270  NEXT I
```

表1　bubble sortの基本プログラム

トロークとよぶこととする。配列における要素を交換する必要がなくなるまで、ストロークを続ける必要がある。これを図1に示す。

ところで、Stephen／Richardの方法はこれよりも比較が多く、その回数はn^2である。その点は改良したことになっていよう。ただし、どちらも比較回数はnの2次式で基本的には大きな違いはない。

昇順にソートする場合、配列の先頭近くにある値の大きな要素は、「1回のストロークで一つしか動かない」から、これが最後の方にくるのは大変な手間を要する。このような要素を亀ということにしよう。一方、配列の最初の方にある小さい要素は「比較的無害」で、これを兎ということにしよう。亀の動く幅を大きくすれば全体として移動回数が減るであろうから、「隣り合った要素の比較」が欠点かも知れない。比較する要素の間に1以上の差があってもよいかも知れないというのが、comb sortの発端である。

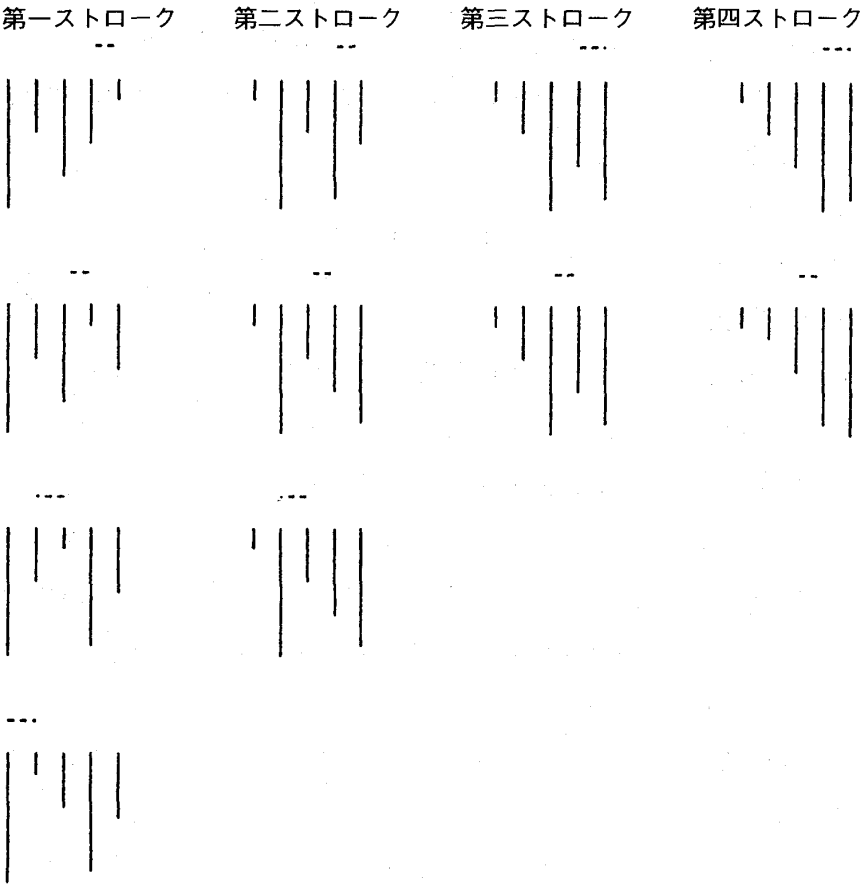


図1　ストロークについて

3. quick sort

これは配列を二つに分割していくことを基本とする方法で、配列に応じて適当な値Xを定め、それによって配列をX以上とX以下で分ける。Xをどう選ぶかに付いては定説がない。現在の所、配列を構成する値の平均値、または中央値などを使うのがよいとされているが、配列の要素を適当に使ってもよく、例えば先頭の値でプログラムを作ったこともある。ここには中央値に

Bubble Sort と Comb Sort

岡本　茂、田口　功、高橋和子
*渋川美紀

Bubble Sort and Comb Sort

Sigeru Okamoto・Isao Taguchi・Kazuko Takahashi・*Miki Shibukawa

1．はじめに

ソートは有限個の数集合を順序よく並べることをいい、そのために多くの方法が工夫された。ここでは最も標準的でわかりやすい方法として「bubble sort」を取り上げ、その方法と特徴を簡単に説明した後、最も高速なソートとして「quick sort」について述べる。次に、本来の目的である「comb sort」について述べ、それらとの比較を論ずる。「comb sort」は、テキサス大学南西医学センターのStephen Lacey助教授とBritish Gas社に勤務しているRichard Box（地球物理学と数学を専攻）によって開発された方法で［文献1］、「bubble sort」を少し手直ししただけだが、速度は大幅に改善されている。従って今後は、やさしい方法として大いに使われることが期待できよう。なお本稿では、comb sort, bubble sortともにStephn／Richardが論じた方法を多少修正したものも用いた。このため計算時間は彼らが示したものと多少異なることを了承されたい。

＊ 白鴎大学助手

2．bubble sort

このアルゴリズムは単純である。一次元配列Tで「隣り合った要素を比較し、必要ならば順序を訂正するためその位置を交換する」というのが基本である。もし比較をなるべく少なくとった次のプログラムで考えると、Tの要素数をnとして、

$$(n-1)+(n-2)+\cdots\cdots+1=n(n-1)/2 \qquad (1)$$

が比較の回数になる。更に、最悪の場合は、交換を同じ回数見込む必要があり、ここで多くの時間を要する。手続きからいえば、全体としてnの2次式だけの手順を考える必要があり、特にデータの移動で時間がかかる。このため移動を減らすと大幅に改善されるが、この点が難しい。

まず基本的なプログラムをBASICで示す（表1）。ここでは配列T(N)を並べ直すことを目的とした［文献2］。

まず配列を先頭から末尾まで見直すことをス