数理解析研究所講究録
第835巻 1993年 194-199

数理計画法におけるＲＡＣソートの時間計算量 －最悪時間計算量O(n)の実数値配列整列法－

仲川　勇二　*(Yuji NAKAGAWA)*
岡山理科大学工学部電子工学科

あらまし

１９８１年に仲川らによって提案されたＲＡＣ (Revised Address Calculation)ソートが，k ビットで表現された実数値要素 n 個から成る配列を，最悪の場合でも $O(kn)$ で整列することを示す．また，３２または６４ビット実数の場合に対して計算機実験を行い，ＲＡＣソートが，配列要素を前処理することでデータの偏りにあまり影響されなくなり，ほぼ $O(n)$ で整列を実行することを示した．特に，６４ビット実数の場合は，ＲＡＣソートがクイックソートよりかなり有利であることが分かった．

1. まえがき

数理計画法の分野で解法アルゴリズムを設計する場合，その中心的な部分で実数値の整列を必要とすることが多い．その際，整列の最悪の場合の時間計算量は，$O(n\log n)$ として取り扱われている[たとえば，Zemel(1980), Dyer(1984), Ibaraki and Katoh(1988)]．しかし，整列の最悪の場合の計算量について Ahoら(1983)は，n 個の要素をソートするのに $O(n\log n)$ 時間必要であるという神話があり，これがかならずしも正しくないことを指摘し，たとえば，基数法(radix sort)や箱ソート(bin sort)がうまく使えるような型の要素であれば $O(n)$ 時間で十分であると述べている．そこで，本論文では，実数値配列の場合に、最悪の時間計算量が，$O(n\log n)$ であるのかまたは $O(n)$ であるかについて考える．

要素間の比較を基本とした整列法として重要なものに，ヒープソート(heap sort)がある．この整列法は，最悪の場合でも比較の回数が $O(n\log n)$ となることが証明されている．このことが比較を基本とした整列法の時間計算量上限 $O(n\log n)$ の理論的な根拠となっている．しかし、実用的にはヒープソートよりはるかに高速なクイックソート(quick sort)が使われる．クイックソートの時間計算量は，平均が $O(n\log n)$ で上限は $O(n^2)$ である．

要素間の比較を基本とせず，基数(radix, base) または番地計算(address calculation)を中心技術として用いて整列を実行する方法がある．基数法は，k ビットからなる n 個の要素の配列を最悪の場合でも $O(kn)$ 時間で整列することができる．しかし実用面では，実数値配列，すなわち $k=32$ や $k=64$ の場合は、基数法はクイックソートよりもかなり遅い．このことは，本論文の計算機実験でも明らかにされる．

バケツソート(bucket sort)は，Issacら(1956)により番地計算法(address calculation sort)として提案された．この方法は，箱ソート(bin sort)，分配ソート(distributive sort)とも呼ばる．この整列法の理論面的視点からの計算時間は，一様分布の配列に対しては $O(n)$ であるが，特殊な例に対しては $O(n^2)$ 時間が必要であると言われている．Dobosiewicz(1978)は，分配法を改良した分配分割整列法(distributive partitioning sort)を提案した．この方法は，配列の中央値を用いて二分割しながら，バケツソートを実行するというもので，最悪の場合でも $O(n\log n)$ となるよう工夫されている．また，一様分布の配列に対しては $O(n)$ が期待できる．
実用的な観点からは，Issacらの整列法は，整列すべき配列の要素の値が一様に分布していると，要素

数の約2倍の分割数をとれば，かなり高速に整列を終了することができ，内部整列法としては計算速度の面ではもっとも優れていた．しかし一様分布以外の配列に対しては効率が著しく低下し，番地計算法は汎用の整列法としては使えなかった[Flores(1969)]．仲川ら(1981)は，この欠点を軽減したＲＡＣソート(Revised Address Calculation sort)を提案した．この整列法は，番地計算法と同様一様分布の実数値配列に対して $O(n)$ で整列を終了し，他の分布（正規及び指数）の実数の配列に対しても効率がほとんど低下しないことを計算機実験で示した．また，大きな作業領域が使えるときには，クイックソートよりもかなり高速であることも示した．しかし，最悪の場合の時間計算量は，$O(n^2)$ と考えられていた．

本論文では，k ビットで表現された n 個の実数値要素から成る配列を，ＲＡＣソートが最悪の場合でも $O(kn)$ で整列することを証明する．また，実際の数値計算で使われる３２または６４ビット実数の場合に対して計算機実験を行った．実験結果より，ＲＡＣソートが，配列要素を前処理することでデータの偏りにあまり影響されなくなり，ほぼ $O(n)$ で整列を実行することが分かった．特に，６４ビット実数の場合は，ＲＡＣソートがクイックソートよりかなり高速であることが分かった．

2. 実数値配列

本論文では，n 個の実数値 $a_1, a_2, \cdots, a_n$ からなる配列を非減少順（または非増大順）に並べ換える整列を取り扱う．各実数値は計算機の中では k ビットを使って浮動小数点表示されているものとする．このとき，kビットの実数は，kビットの整数と，整列に関しては完全に同等であると考えられる．たとえば，多くのパーソナルコンピュータやワークステーションにおいて採用されているＩＥＥＥの標準形式の場合，任意の２個の正の浮動小数点において，その内部表現をそのまま整数の内部表現とみなし取り扱っても２数の大小関係には影響を与えない．実際に，浮動小数点どうしの大小の比較は整数比較命令を使って容易にかつ高速に行える．また，k を可変長とした場合，たとえば２個の実数の大小関係の判定に要する時間も k の関数となり，比較を基本としたヒープソート(heap sort)の時間計算量の上限が $O(n\log n)$ ではなくなってしまう．したがって，現実的な仮定として，k は固定長とする．

3. ＲＡＣソート

図１は，仲川ら(1981)により提案されたアルゴリズムを、再帰関数を用いて書き変えたものである。図１のアルゴリズムを新らたにＲＡＣソート(Recursive Address Calculation sort)と呼ぶことにする。アルゴリズムの中の**DEFDATA**と**ENDDEF**の間では、データの階層的な定義を行なっている。ＣまたはＣ＋＋では構造体、PascalまたはModula-2ではレコード型を用いて実現できる。

パラメータ α は、再度ＲＡＣソートを適用するかどうかを決めるための基準値で、整列すべき要素数が α 以上の時ＲＡＣソートを再帰的に用い、その他の場合は他の整列法（本論文ではクイックソートまたは直接挿入法）を用いる。パラメータ τ は，要素数に比例した数に分割数を決定するために用いる．

記号 $\Leftarrow$ は関数の出力を意味する．たとえば，

$$\{A, B\} \Leftarrow Func(C, D);$$

は，データ列 C, D を関数 Func の入力として，データ列 A, B を出力とする．

関数 Partition は，配列 $\{a_1, a_2, \cdots, a_n\}$ の i^{FST} から i^{LST} 番目までの要素を，次の番地計算関数を使って計算したグループ番号を用いて，n^{DIV} 個のグループに分割する．

$$d = \lfloor (n^{DIV}-0.01)(a_i-a_{MIN})/(a_{MAX}-a_{MIN}) \rfloor + 1$$

ただし，

$$a_{MIN} = \min\{a_i \mid i = i^{FST}, i^{FST}+1, \cdots, i^{LST}\},$$

$$a_{MAX}= \max\{a_i|\ i= i^{FST}, i^{FST}+1, \cdots, i^{LST}\},$$

そして $\lfloor y \rfloor$ は y を越えない最大整数である．分割はグループ 1 から n^{DIV} までが順番になるよう $ARRAY$ 内で要素を入れ替え，各グループの最初の位置を $BUCKETS$ 内に記録する．この関数内では $BUCKETS$ は作業領域節約のためカウンターとしても利用する．この関数の出力は $AandS$ である．

関数 Classification は，要素数が α 以上の場合は，そのグループの $ARRAY$ 内での最初と最後の位置を $STACK$ に積み上げる(Push)．要素数が α 未満の場合は，そのグループを他の整列法（クイックソートまたは直接挿入法）を用い整列する．

4. ＲＡＣソートの時間計算量

ＲＡＣソートは，k ビット n 個の実数値要素からなる配列に対して次のような性質をもつ．

性質１　再帰の深さ（関数が呼び出された回数）は最大限 $p^{UB} = \lceil k/log_2(\tau\alpha) \rceil$ である．

（証明）常に分割数 n^{DIV} は $\tau\alpha$ 以上であるから，分割により得られる情報量が k ビットを越える再帰の深さ p^{UB} は，

$$\log_2((\tau\alpha)^{p^{UB}}) \geq k\ .$$

これより，

$$p^{UB} \geq k/\log_2(\tau\alpha)\ .$$

性質１を用いた実際的な例として，τ= 0.5，α= 1000の場合を考える．３２ビット浮動小数点のとき，p^{UB}= 4 で，６４ビット浮動小数点のとき，p^{UB}= 8 である．これらは実用的な数値である．

性質２　再帰の深さが $p \leq p^{UB}$ であるグループに対して，関数 Partition が各グループを分割して得られたサブグループの数の総合計は τn を越えることはない．

（証明）再帰の深さが $p-1$ のときに生成されたサブグループで，空でないグループの要素の総数は n を越えないことから明らかである．

再帰の深さ p において，関数 Partition が必要とする計算時間の総合計は性質２より τn に比例した時間である．また，要素数 α 未満のグループの個数は n を越えることはないので，関数 Classification において整列に必要な合計の計算時間は $c_1 n$ を上限値として持つ，ただし c_1 は α 個の要素を整列するのに必要な計算時間の上限値である．したがって，再帰の深さ p において関数 Partition と関数 Classification が使う計算時間 T_p に対して，

$$T_p < c_2 n$$

となる c_2 が存在する．性質１よりＲＡＣソートで n 個の要素の整列に必要な計算時間は

$$T = \sum_{p=1}^{p^{UB}} T_p < p^{UB} c_2 n < ckn,$$

但し，c は適当な定数である．ＲＡＣソートは，k ビット n 個の実数値配列を最悪の場合でも $O(n)$ で整列することがわかる．

5. 計算機実験

本章では，計算機実験を通して，ＲＡＣソートが実用面でも有効な整列法であることを示す．計算機実験で取り扱う配列の要素は，一語３２ビットまたは６４ビットで表現された実数である．その表現形式は，ＩＥＥＥ標準の浮動小数点形式である．この表現形式の特徴は，２つの数値の２進数の浮動小数点表示を対応した符号付き整数として取り扱っても，両方とも負数の場合に大小関係が逆転する以外は大小関係に変化がないことである．特に大きな利点は３２ビットの実数の場合は，整数型

の比較命令を使って大小関係の比較が高速に行えることである．

表１には，３２ビット実数の場合の各種整列法の計算結果を示す．配列要素の生成には一様乱数（uniform）を用いた． quick1 は Knuth(1973)の非再帰版クイックソートのアルゴリズムをＣ言語で実現したプログラムである．このアルゴリズムは FORTRAN で実現した場合最も高速なものの一つである．quick2 は Sedgewick(1988)のクイックソートの再帰版アルゴリズムでＣ言語で実現した．quick3 が quick2 と異なるところは，再帰的分割の結果要素数が９個以下になった部分では分割を停止し，後にまとめて単純挿入法を適用した点である．この方法は quick1 でも用いられている．表１の radix は Sedgewick(1988) の pascalで記述された straightradix をＣ言語で実現した．ビット単位の取扱は，Ｃ言語のビット操作命令を用いて高速化を計っている．ビット列の比較は８ビットごとに行った．表１の RAC ではパラメータは $\tau = 0.5$, $\alpha = 1000$ として実行した．要素数が α 以下の部分配列に対しては， quick1 を適用した． RACの作業領域としては，約 $1.5n$ 個の３２ビット整数型メモリーを使った．

表１では，３２ビット実数要素を，その２進数内部表現に対応した３２ビットの整数として取り扱った．このとき，quick1, 2, 3, および radix では実数を操作する部分は全く無くなるが，RAC では番地計算を行う部分で実数計算が残るため相対的にＲＡＣソートにとては不利になる．表１の結果より，クイックソートの中では若干 quick3 が優れていることがわかる．quick3 と radix の比較から，基数法は最悪の場合でも $O(n)$ の整列法ではあるが，非現実的なほど大きな要素数（表１からの概算で $n = 7.8\times10^{11}$）にならないかぎり，基数法はクイックソートには勝てないことがわかる．ＲＡＣソートの場合は，データの分布にも多少依存するが，表１からの概算で $n = 5.3\times10^{6}$ 程度の要素数の時にクイックソートよりも早くなる．

クイックソートや基数法は，うまく用いればデータのばらつき（分布）にあまり影響をされない特質をもつ．しかしバケツソート類はデータの偏りに影響され易いという欠点をもつ．そこで，ＲＡＣソートに対してのみさらに配列要素の生成に指数分布の乱数（exp），そして指数部が極端に変化する乱数（unreal）を用いて，計算機実験を行った．乱数 unreal は，式 $d\times10^{e}$ でそれぞれ $0.0 \leq d < 1.0$, $-35 \leq e \leq 35$ の一様乱数を用いて発生した．表１の結果より，実数をその内部表現に対応した整数として取り扱うことで，ＲＡＣソートのデータの分布の偏りの影響を受け易いという欠点がかなり緩和できることがわかった．

表２には６４ビット実数の場合のクイックソートとＲＡＣソートの実験結果を示す．本実験では，６４ビットの実数をそのまま倍精度実数として取り扱っている．クイックソートの中では，quick3 がやはり若干早いことがわかる．また乱数 uniform での実験結果の比較では， RAC が quick3 よりかなり高速であることがわかる．表２には，RAC の乱数 exp や unreal に対する実験結果も示している．ここでの乱数 unreal の指数部 e は， $-305 \leq e \leq 305$ の一様乱数として発生した．この実験結果から，ＲＡＣソートが分布の偏りの影響を受け易いことがわかる．特に乱数 unreal に対する結果は，ＲＡＣソートの実用性を否定し，最悪の場合はＲＡＣソートが $O(n^2)$ の整列法となることの証拠のように見える．しかしこれは実数データの取扱いのまずさに起因している．ディジタル計算機では実数も離散値にしかすぎないにもかかわらず，数学的な連続量として取り扱っていることが原因である．表３には整列すべきデータの対数をとった場合（RAClog）と，内部表現に対応した整数を再度倍精度実数として取り扱った場合（RACint）の実験結果について報告している．上記の前処理をすることにより，一部情報量は減少するがＲＡＣソートのデータの偏りに依存する欠点はかなり解消できることがわかった．

表１，２，３よりＲＡＣソートがほぼ $O(n)$ であることがわかる．計算機実験で $O(n)$ を完全に示すことは難しい．仲川ら（1981）により議論されたように，計算機本体のアーキテクチャーに起因した原因，すなわち配列のランダムアクセス等がキャシュメモリ等のため等時間で行われないことによる．

参考文献

Aho A. V., J. E. Hopcroft, and J. D. Ullman(1983), Data structures and Algorithms, Addison-Wesley.

Dobosiewicz W.(1978), Sorting by distributive partitioning, Information Processing Letters, 7, 1-6.

Dyer M. E.(1984), An $O(n)$ algorithm for the mulpiple-choice knapsack linear program, Math. Program., 29, 57-63.

Ibaraki T. and N. Katoh(1988), Resource allocation problems, The MIT Press.

茨木(1989), アルゴリズムとデータ構造, 昭晃堂

Knuth D. E.(1973), The art of computer programming Vol. 3: Sorting and searching, Second printing, Addison-Wesley, Reading, Mass.

仲川, 疋田(1981), 改訂番地計算分類法, 電子情報通信学会論文誌, J64-D, 8, 737-741.

仲川, 疋田(1982), 作業領域を縮小した改訂番地計算分類法, 電子情報通信学会論文誌, J65-D, 7, 942-943.

仲川, 中村(1990), 再帰番地計算（ＲＡＣ）サーチアルゴリズム, 電子情報通信学会論文誌, J73-A, 10, 1724-1725.

Sedgewick R.(1988), Algorithms, 2nd ed., Addison-Wesley.

van der Nat M.(1980), A fast sorting algorithm, a hybrid of distributive and merge sorting, Information Processing Letters, 10, 163-167.

Zemel E.(1980), The linear multiple choice knapsack problem, Operations Research, 28, 1412-1423.

DEFDATA
$PARA = \{\alpha, \beta, \tau\}$;
$ARRAY = \{ i^{FST}, i^{LST}, \{a_1, a_2, \cdots, a_n\}\}$;
$STACK = \{n^S, \{s_1, s_2, \cdots, s_{nS}\}\}$;
$AandS = \{ARRAY, STACK\}$;
$BUCKETS = \{n^B, \{b_1, b_2, \cdots, b_{nB}\}\}$;
ENDDEF
FUNCTION RACSort()
INPUT $PARA, AandS$;
BEGIN
$n^{DIV} \leftarrow \lfloor \tau(i^{LST} - i^{FST} + 1) \rfloor$;
$\{BUCKETS, ARRAY\} \Leftarrow$ Partition($n^{DIV}, ARRAY$);
$\{AandS\} \Leftarrow$ Classification($\alpha, BUCKETS, AandS$);
IF $n^S > 0$ **THEN**
$\{i^{FST}\} \Leftarrow$ Pop($STACK$);
$\{i^{LST}\} \Leftarrow$ Pop($STACK$);
$\{AandS\} \Leftarrow$ RACSort($PARA, AandS$);
ENDIF
RETURN $AandS$;
END

Figure 1. A recursive procedure of RAC sort.

Table 1
Comparison of execution times (sec) for 32-bit real numbers

No. of elements(n)	3000	10000	30000	100000	300000
quick1(uniform)	0.100	0.384	1.29	4.85	15.9
quick2(uniform)	0.109	0.410	1.37	5.05	16.4
quick3(uniform)	0.098	0.372	1.25	4.67	15.2
radix(uniform)	0.266	0.878	2.70	9.21	27.8
RAC(uniform)	0.150	0.513	1.63	5.62	17.3
RAC(exp)	0.146	0.504	1.59	5.51	17.0
RAC(unreal)	0.140	0.484	1.53	5.34	16.3

Table 2
Comparison of execution times (sec) for 64-bit real numbers

No. of elements(n)	3000	10000	30000	100000	300000
quick1(uniform)	0.267	1.031	3.50	13.2	43.4
quick2(uniform)	0.281	1.075	3.63	13.5	44.2
quick3(uniform)	0.246	1.017	3.46	13.0	42.5
RAC(uniform)	0.177	0.615	1.91	6.58	20.0
RAC(exp)	0.210	0.727	2.28	7.80	24.1
RAC(unreal)	11.2	39.3	108.4	323.7	884.1

Table 3
Execution times (sec) of RAC sort for 64-bit real numbers

No. of elements(n)	3000	10000	30000	100000	300000
RAClog(uniform)	0.210	0.727	2.27	7.81	24.1
RAClog(exp)	0.119	0.693	2.16	7.45	23.0
RAClog(unreal)	0.180	0.629	1.96	6.77	20.5
RACint(uniform)	0.194	0.718	2.28	7.85	23.8
RACint(exp)	0.177	0.614	1.91	6.58	19.9
RACint(unreal)	0.178	0.614	1.91	6.58	20.0