Vol. 35 No. 12 情報処理学会論文誌 Dec. 1994

多重分割ソートアルゴリズム

河村知行† 江口賢和†
小笠原基泰† 重村哲至†

主記憶上の高速なソートアルゴリズムとして，クイックソートがよく知られている．小量の作業領域を使用することで，より高速なソートアルゴリズムを実現したのでこれについて報告する．このアルゴリズムにより，クイックソートの約70%の処理時間でソートを実行できた．実際に使用されている7種類のUNIXオペレーティングシステムのクイックソート関数(qsort)との比較も行っている．UNIXのqsort関数は非効率な実現のものが多いので，本アルゴリズムはqsortの約50%の処理時間でソートを実行できた．

Multi Partition Sort Algorithm

Tomoyuki Kawamura,† Yoshikazu Eguchi,†
Motoyasu Ogasawara† and Tetsuji Shigemura†

Quicksort is known as a fast sorting algorithm on main-memory. We developed a faster sorting algorithm using a small amount of working area. We achieved to sort data in about 70% processing time as compared with quicksort. We compared this algorithm with quicksort functions (qsort) which are used in actual UNIX operating systems. In this case, it sorted data in about 50% processing time as compared with qsorts of UNIX, because many qsorts in UNIX are badly implemented.

1. はじめに

主記憶上の高速なソートアルゴリズムとしてクイックソート[1)~4)]が知られている．しかし，主記憶上のクイックソートに限定しても，そのキーの種類や要素の比較方法の与え方などにより，種々の方式が考えられる．本論文におけるソートアルゴリズムは，UNIXのqsort関数と同じ仕様のものとする．

このように限定した場合でも，クイックソートが最速のアルゴリズムであるとは言えない．問題の大きさをnとしたときその処理時間のオーダがクイックソートと同じ$n \cdot \log_2 n$であっても，クイックソートより高速なアルゴリズムであれば有用なものとなる．

我々は以上の前提のもとに，クイックソートより高速のソートアルゴリズムを新規に開発したのでこれに付いて報告する．本ソートを多重分割ソート(Multi partition sort)と呼ぶ．以下mpsと記す．mpsは，第一段階で配列を2のべき乗個の区間に分割する．第二段階で各区間をソートすることで全体をソートする．mpsは，多少の余計な主記憶(作業領域)を用いることによりクイックソートの約70%の処理時間でソートを行うことができる．

† 徳山工業高等専門学校情報電子工学科
Department of Computer Science and Electronic Engineering, Tokuyama College of Technology

2. mpsの概略

mpsは，与えられた配列Aの要素をその配列上でいくつかの区間に分割することによりソートを行う．いくつかの区間に分割する処理は基数ソートに似ているが，mpsでは区間の個数と分割の基準になる要素を配列Aの中から自動的に選び出す．

例として**図1**(a)の12個の要素より成る配列A(添え字0〜11)を4分割することによりソートを行う．ここで，要素はキーのみから成るものとする．

60，20，40の3つの要素(図1(a)の下線)がサンプルとして採られ，これをソートして20，40，60の分割の基準になる値(しきい値)得る．このしきい値により分割を行い図1(b)を得る．しきい値としきい値の間の区間(図1(b)の下線)はまだソートが完了していないので再度ソートを行う．これによりソートされた配列図1(c)を得る．

以上をまとめると次のような9つの処理になる．

1) 分割数の決定
2) サンプルの採集
3) サンプルのソート

	添え字	0	1	2	3	4	5	6	7	8	9	10	11
(a)	ソート前の配列A	31	50	52	60	51	70	20	30	71	40	10	20
(b)	分割終了後の配列	10	20	20	31	30	40	51	52	50	60	71	70
(c)	ソート後の配列	10	20	20	30	31	40	50	51	52	60	70	71

図 1 分割ソートの概略
Fig. 1 Outline of mps.

4) 各要素の入る区間の決定
5) 各区間の大きさの計算
6) 各区間の先頭の位置の決定
7) 各区間の先頭の要素の退避
8) 要素の移動
9) 区間内のソート

3. 各処理の説明

3.1 分割数の決定

配列の要素数を n，分割数を b とするとき，b は $2 \leqq b < n$ の整数である．そして，4番目の処理「各要素の入る区間の決定」を2分割探索により効率よく行うために，$b=2^k$ ($k=2, 3, 4, \cdots$) とする．実際の処理では b は 16〜2048 ($k=4$〜11) 程度となる．b の値の決定方法は第4章で述べる．

以下では，簡単のために $b=4$ として説明を行う．

3.2 サンプルの採集

分割を行うためのしきい値を得るために，3個（b−1個）の要素を取り出して配列S（添え字1〜b−1）に複写する．図1(a)では A[3], A[6], A[9] を S[1], S[2], S[3] の値とする．

一般に，サンプル間隔を d ($=n/b$) とするとき A[d], A[2d], A[3d]〜A[(b−1)d] を S[1], S[2], S[3]〜S[b−1] に複写する．

3.3 サンプルのソート

サンプルを複写した配列Sを何らかのソートアルゴリズムによりソートする．

3.4 各要素の入る区間の決定

配列Aの各要素と配列Sを比較することにより，各要素がどの区間に入るかを決定してその区間番号を記憶する．ここで区間と区間番号の関係は次のようなものである．

キー＜20————区間番号1
キー＝20————区間番号2
20＜キー＜40————区間番号3
キー＝40————区間番号4
40＜キー＜60————区間番号5
キー＝60————区間番号6
60＜キー————区間番号7

一般に，b 分割の場合は $2b-1$ 個の区間がある．

配列Aの各要素を配列S上で2分木探索することにより，その要素の入る区間を決定する．長さ n の配列Bを別に用意して，A[i] の区間番号を B[i] に代入する．この処理には約 $n \cdot \log_2(b)$ 回のキーの比較が必要となる．

配列Bの要素の大きさは $\log_2(2b)$ ビット (=5〜12) 必要となる．以後，配列Xの要素の大きさ（バイト数）を se(X) で表す．上記の例では se(B)=1〜2 となる．

3.5 各区間の長さの計算

配列Bの値の集計により各区間に入る要素数を得る．この要素数を記憶するために長さ $2b-1$ の配列Cを別に用意する．

この処理は前の「各要素の入る区間の決定」と同じ繰り返しの中で行っている．

3.6 各区間の先頭の位置の決定

配列Cの各要素を先頭から加算していくことにより，各区間の先頭の位置を得る．この位置を記憶するために長さ $2b-1$ の配列Hを別に用意する．

また，この処理は次の「各区間の先頭の要素の退避」と同じ繰り返しの中で行っている．

3.7 各区間の先頭の要素の退避

配列Hの値により，各区間の先頭に位置している要素を別の配列に退避する．このために長さ $2b-2$ の配列Tを用意し，この配列をスタックとして使用して対象となる要素をプッシュする．アルゴリズムの性質により最左端の区間の先頭要素のプッシュは必要ない．

配列Tの要素の大きさは，次式のようにする．

$$se(T)=se(A)+se(B)$$

se(B) の部分はプッシュした要素の区間番号を記憶するのに使用される．

3.8 要素の移動

図2のように要素の移動を行って分割を完成させる．ここで太い下線は作り上げるべき区間を表す．矢印は要素の代入を表す．四角は空の要素を表す．

まず，最左端の要素(31)を「代入元」とする．「代入元」の区間番号を配列Bより得る．その区間の空き要素(60)を「代入先」とする．「代入元」の要素を「代入先」へ代入する．「代入元」と「代入先」が同じ区間の場合には代入は行わない．

次に「代入先」の右隣を「新しい代入元」として，

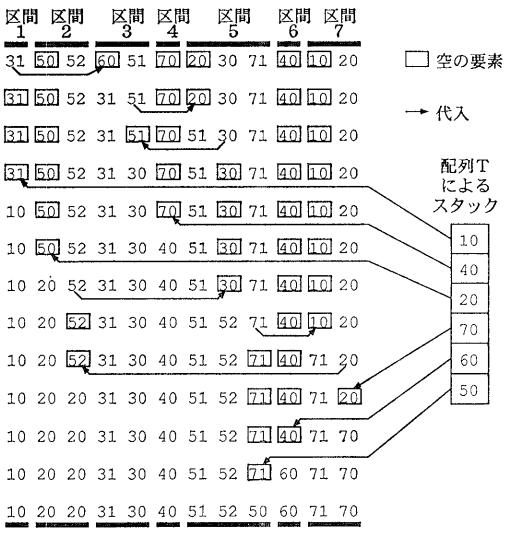

図 2 分割の過程
Fig. 2 Process of division.

上記の処理を繰り返す．これを繰り返すと，いずれは右隣が区間をはみ出る状態となる．この場合はスタックTの先頭を「新しい代入元」として上記の処理を繰り返す．

スタックTが空になったときに要素の移動が完了する．

3.9 区間内のソート

各区間内を何らかのソートアルゴリズムによりソートする．ここでは区間番号2，4，6の区間は，しきい値だけの要素から成るのでソートの必要はない．この処理において mps を再帰的に呼び出すことも可能である．

mps を再帰的に呼び出す場合でも，作業領域の節約のために配列S，B，C，H，Tはトップレベルで確保したものを再利用するようになっている．このような場合にも mps が正しく動作するように，「要素の移動」開始直前に配列Cを別に用意した配列Dに複写しておく．本処理ではこの配列Dより各区間の「先頭と長さ」を求めている．このDは再帰呼び出しごとに領域を確保する必要がある．

配列CとDは統合することも可能であるが，処理が複雑になる．また，配列C，Dの長さ $(2b-1)$ は n に比べて小さいのでこの統合は行っていない．

4. 分割数の決定方法

与えられた配列Aに対する分割数は，mps の性能を大きく左右する．mps の性能は se(A) の大きさにも依存する．本 mps では実験により処理時間に関して最適な分割数を求めて，その結果を mps のアルゴリズムに埋め込んだ．

表 1 要素数とその最適な分割数
Table 1 The best numbers of divisions for the number of elements.

要素数 (n)	分割数 (b)	条 件
～90	なし	—
～250	16	se(A)≧100
～400	32	se(A)≧ 40
～700	64	se(A)≧ 40
～1000	64	se(A)≧ 20
～2000	128	se(A)≧ 20
～4000	256	se(A)≧ 20
～10000	512	se(A)≧ 20
～40000	1024	se(A)≧ 20
40001～	2048	se(A)≧ 20

すなわち，配列Aの要素の大きさ 8～200 バイトと要素数 50～200 万個の種々の組み合わせに対して，13 通りの分割（4～16384 分割）で実験を行い最適な分割数を求めた．キーは乱数関数による整数値とした．比較関数はキーの単純な大小比較とした．要素の代入は言語Cの memcpy 関数で行った．実験はすべて仮想記憶が働かない状態で行った．

その結果，要素の大きさが 20 バイト未満や要素数が 100 未満の場合ではクイックソート（以後 qs と略す）より遅かったので，この場合は qs を呼び出すようにした．

要素数が 100 以上の場合の最適な分割数を**表 1** に示す．se(A) の大きさにより最適な分割数は多少変動するので，表1はその平均的な値である．「条件」の式が成立するときに mps を実行し，それ以外では qs を実行する．

5. 性 能 評 価

5.1 クイックソートとの比較

se(A)＝100 の場合の qs と mps の実行結果を**表 2** に示す．実験は，SONY NEWS NWS-5000 VI で行った．qsG は GNU[5] システムの中にある言語Cの qsort 関数である．qs 5 は mps の開発に際して作られた qs の高速版である．qs 5 の代入回数が qsG より少ないのは要素の交換を行わないアルゴリズム[2]を採用しているためである．mps 中の qs はすべて qs 5 を使用している．

「mps/qs 5」の行は，処理時間の比である．mps が

表 2 クイックソートと mps の比較
Table 2 Comparison of quicksort and mps.

尺度	方式	要素数		
		1000	10000	100000
比較回数(回)	qsG	11488	154532	1945550
	qs5	9620	135599	1727544
	mps	9519	130155	1636446
代入回数(回)	qsG	8178	105560	1291327
	qs5	5305	68744	847851
	mps	3928	41342	463974
処理時間(秒)	qsG	0.0683	0.8987	11.7867
	qs5	0.0413	0.5487	7.4800
	mps	0.0336	0.3877	4.9433
時間	mps/qs5	81.4%	70.7%	66.1%

se(A)=100 %=処理時間比 (mps/qs5)

表 3 要素の大きさに対する mps の性能
Table 3 Performance of mps for various element sizes.

se(A)	要素数		
	1000	10000	100000
20	97.8%	86.4%	86.3%
40	88.5%	76.9%	75.7%
100	81.4%	70.7%	66.1%
200	78.8%	69.0%	63.4%

%=処理時間比 (mps/qs5)

qs5 の約 70% の処理時間でソートを実行している．

要素数が多いほど mps は良い性能を示す．また，比較回数や代入回数も qs に比べて減少している．

5.2 要素の大きさによる比較

se(A) が異なる場合の mps の性能（処理時間の比 (mps/qs5)）を**表 3** に示す．se(A) が大きいほど mps は良い性能を示す．

5.3 種々のデータに対する性能

種々のデータに対する qs と mps の処理時間を**表 4** に示す．「乱数」は random 関数によりキーを決定している．「D10 (D100, D1000)」は random()%10 (100, 1000) によりキーを決定している．このようなキーは郵便番号や県別コードなどで現れる．「昇順」はキーが 1, 2, 3, 4, …であり，「降順」は…, 4, 3, 2, 1 である．いずれのデータに対しても mps が高速であることがわかる．

qs5 は，データが昇順（降順）になっていると予測したときに失敗を覚悟で昇順（降順）の検査を行っている．これを受けて，mps はサンプル配列 S が昇順

表 4 種々のデータに対するソート関数の性能
Table 4 Performance of sort functions for various data types.

データ	方式	要素数		
		1000	10000	100000
乱数	qsG	0.0683	0.8987	11.7867
	qs5	0.0413	0.5487	7.4800
	mps	0.0336	0.3877	4.9433
D10	qsG	0.0724	1.1283	16.1833
	qs5	0.0136	0.1370	1.8800
	mps	0.0104	0.1050	1.2700
D100	qsG	0.0614	0.9397	14.0800
	qs5	0.0284	0.2697	3.3367
	mps	0.0174	0.1417	1.6567
D1000	qsG	0.0650	0.8280	12.1633
	qs5	0.0396	0.4207	4.6767
	mps	0.0307	0.2207	1.9733
昇順	qsG	0.0048	0.0540	1.2567
	qs5	0.0004	0.0040	0.1200
	mps	0.0009	0.0097	0.1633
降順	qsG	0.0170	0.1770	2.6600
	qs5	0.0112	0.1150	1.3900
	mps	0.0125	0.1257	1.4533

se(A)=100 単位：秒

(降順) であったときに，分割を行わずすぐに qs5 を呼び出している．そのため，「昇順 (降順)」においては mps は qs5 よりわずかに遅くなっている．しかし，qsG に比べればかなり高速である．

5.4 比較回数の評価

キーを一様乱数により決定（すべての値は異なるとする）したときの配列（長さ n）に qs を適用した場合の比較回数の期待値を qs_cmp(n) とする．qs_cmp は次式で表せる[3]．

$$\mathrm{qs\_cmp}(n)=\alpha n\log_2(n) \qquad (\alpha \text{ は定数})$$

文献 3) では，配列の中央の要素をしきい値としているので $\alpha=2\ln(2)=1.3629$ である．qs5 では，配列中の 3 点の中央値をしきい値としているので $\alpha=\frac{12}{7}\ln(2)=1.188$ である．これは実測値 1.19 に一致している．

同じ配列に b 分割の mps を適用したときの期待値を mps_cmp(n, b) とする．mps の内部では qs を使用するものとする．mps_cmp は次式で表せる．

$$\begin{aligned}\mathrm{mps\_cmp}(n,b)=&\mathrm{qs\_cmp}(b-1)\\&+(n+1)\log_2(b)-b+1\\&+E(\mathrm{qs\_cmp},n-(b-1),b)\end{aligned}$$

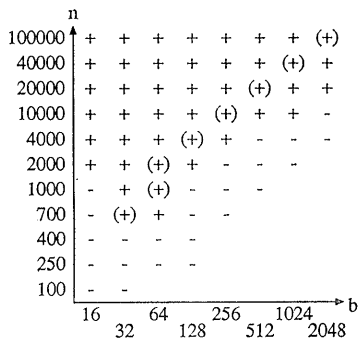

図 3 mps_cmp と qs_cmp の関係
Fig. 3 Relation of mps_cmp and qs_cmp.

ここで1行目は「サンプルのソート」中の，2行目は「各要素の入る区間の決定」中の比較回数である．3行目は「区間内のソート」中で qs により b 個の区間をソートしたときの比較回数の合計である．E の第2引数は要素数である．偶数番目の区間はソートを必要としないので，n より $b-1$ だけ小さくなっている．Eは次式で表せる．

$$E(f,n,b)=\frac{n+b}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\sum_{k=1}^{n}\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]f(k)$$

（付録1参照）

$$E(\text{qs\_cmp},n,b)\sim\alpha(n+b)(\log_2(n-1)-\log_2(b)+0.60995)$$

（付録2参照）

ここで $\left[\begin{matrix}n\\ b\end{matrix}\right]$ は組み合わせ（${}_nC_b$）を表す．

以上のとき，次の関係式を考える．

$$\text{mps\_cmp}(n,b)<\text{qs\_cmp}(n)$$

$\alpha=1.188$ のとき，上式が成立する n，b の関係を**図3**中の＋で示す．（＋）は与えられた n に対して差が最大になる b の位置を示す．これは，$b\cdot\log_2(b)\sim0.3n$ を満たす b である．－はこの関係式の不成立を示す．空白の部分は関係式が無効な所である．

表1（$n\geqq700$）は図3の＋の部分の関係を与えているので，mps の比較回数は qs より少ないと言える．

5.5 代入回数の評価

同様の議論を代入回数についても行うことができる．qs と mps の代入回数を与える関数 qs_ass と mps_ass は次式で表せる．

$$\text{qs\_ass}(n)=\beta n\log_2(n)\quad(\beta\text{は定数})$$

$$\text{mps\_ass}(n,b)=3b+\text{qs\_ass}(b-1)+n+E(\text{qs\_ass},n-(b-1),b)$$

β の値は文献3)では $\frac{1}{2}\alpha$，qs 5 では，$\frac{2}{5}\alpha$ である．

mps_ass の $3b$ の項は「サンプルの採集」と「各区間の先頭の要素の退避」中の，qs_ass の項は「サンプルのソート」中の，n の項は「要素の移動」中の，E の項は「区間内のソート」中の代入回数を表す．

以上のとき，前節と同様の議論により，表1において mps の代入回数は qs より少ないと言える．

6. 作業領域を減らす方法

mps が使用する作業領域の大きさ（バイト数）は次のとおりである．

配列S	se(A)·b	
配列B	2·n	(se(B)=2 とする)
配列C	4·2b	(se(C)=4 とする)
配列D	4·2b	(se(D)=4 とする)
配列H	4·2b	(se(H)=4 とする)
配列T	se(T)·2b	

この合計は次式のとおりである．

$$\text{se(A)}\cdot b+2n+24b+\text{se(T)}\cdot2b$$

これを小さくする方法を以下に述べる．

6.1 再帰呼び出しの排除

mps の「サンプルのソート」と「区間内のソート」では任意のソートアルゴリズムによる処理が可能であり，mps を再帰的に呼び出すこともできる．しかし，これを行うと配列Dが2個以上必要になる．実験の結果，「サンプルのソート」で mps を再帰呼び出しすると 1% 程度の改善が得られたが，「区間内のソート」では改善は得られなかった．前節の「性能評価」では，mps の再帰呼び出しを一切せずに実験を行っている．

6.2 配列SとTの共用

配列Sは「各要素の入る区間の決定」が終了すると不必要になる．また，配列Sは配列Tの半分以下の大きさである．そこで，配列Tの一部を配列Sとして使

表 5 小さな分割数での mps の性能
Table 5 Performance of mps for the small number of division.

データ	要素数		
	1000	10000	100000
乱数	0.0338	0.4140	5.1733
D10	0.0107	0.0953	1.1733
D100	0.0176	0.1343	1.6900
D1000	0.0309	0.2510	2.7133
昇順	0.0009	0.0047	0.1300
降順	0.0125	0.1167	1.4333

se(A)=100 単位：秒

表 6 種々の計算機上での mps の性能
Table 6 Performance of mps on various computers.

機種	CPU	OS	○−	○+
DELL 466 L (IBM-PC)	i 80486 DX 2	NetBSD-0.9	50.5%	42.5%
SONY NEWS-831	MC 68030	NEWS-OS-3.4	67.8%	69.5%
SONY NEWS-3460	R3000	NEWS-OS-3.91R	42.3%	42.6%
SONY NEWS-5000 VI	R4000	NEWS-OS-4.2.1R	70.9%	70.5%
SUN SS1+	SPARC	SunOS4.4.1B	57.4%	52.8%
PH HP 9000/720 GRX	PA-RISC	HP-UX-A.09.01	60.9%	54.3%
IBM POWER-station-520	RS6000	AIX3.0	48.9%	41.9%

要素数=10000 se(A)=100 %=処理時間比 (mps/qsort)

用することができる．前節の「性能評価」の mps はこれを行っている．この場合の作業領域の大きさは次式のとおりである．

$2n+24b+\mathrm{se(T)}\cdot 2b$

6.3 $b \leqq 128$ の場合

「配列SとTの共用」に加えて分割数 b の上限を 128 にすることにより，性能は多少低下するものの作業領域を大幅に減らすことができる．この場合の作業領域の大きさは次式のとおりである．

$n+28b+\mathrm{se(T)}\cdot 2b$

b の減少による作業領域の減少だけでなく，n の係数を2から1にできる．なぜなら，b が 128 の場合，区間番号は 1〜255 になり，それを1バイトに格納できるからである．$24b$ が $28b$ になったのは，「区間内のソート」において mps の再帰呼び出しを許したためである．$28b$ は $32b$ になる可能性もある．

b の上限を 128 にした場合の種々のデータに対する mps の処理時間を**表 5** に示す．表 4 と比べて性能はあまり低下していない．むしろ性能が向上している部分もある．これは表 4 の mps が必要以上のサンプルを収集している場合である．

6.4 配列Tの排除

「各区間の先頭の要素の退避」をなくし，「要素の移動」を変更することにより配列Tをなくすことができる．ただし，配列Sは必要である．

すなわち，「要素の移動」での「代入元」から「代入先」への代入を行わずにその「代入元」の位置をスタックKにプッシュする．このスタックKには配列Sが再利用できる．そして，「新しい代入元」は「代入先」の右隣ではなく「代入先」そのものとする．これを繰り返して，代入予定の鎖を伸ばして行く．鎖を伸ばすことができなくなったときにスタックKから「代入元」の位置をポップアップしながら配列Aの要素の代入を行う．以上の処理を分割が完成するまで繰り返す．

この方式の実験の結果，要素の代入回数を約 $1.8\times b$ 回減らすことができた．しかし，「要素の移動」中にあった繰り返しのほかに，この方式ではスタックKのポップアップのための繰り返しが必要になる．この繰り返し回数は n である ($n \gg b$)．実際には約 $15\times n$ の機械語命令の実行が増えた．se(A) が大きいときはこの変更は有益であるが，se(A)=40 程度では不利益をもたらす．この変更は se(A) の大きさにより採用されるべきである．

この場合の作業領域の大きさは次式のとおりである．

$\mathrm{se(A)}\cdot b+2n+24b$

6.5 配列SとTの排除

前節の「配列Tの排除」のアルゴリズムを修正して，配列SとTの両方を排除することも可能である．

すなわち，「サンプルのソート」で配列Sをソートする代わりに A[d], A[$2d$], A[$3d$], …, A[$(b-1)d$] をソートし，「各要素の入る区間の決定」で配列Sの代わりにこれを使用するのである．

しかし，実験の結果，性能はわずかながら低下した．これは，キャッシュメモリや TLB のヒット率が低下するためと考えられる．

この場合の作業領域の大きさは次式のとおりである．

$2n+24b+4m$ または $n+28b+4m$

ここで $4m$ は「配列Tの排除」で述べたスタックKの大きさである．4 は se(K) の値である．m はスタックKの最大長 (200 程度) である．これ以外に，「要素の移動」のための作業領域 (大きさ se(A)) が一つ必要である．しかし，これは qs が使用する作業領域と共用できるのでここでは無視している．

$m=200$, $b=128$ としたとき，上式は次のようになる．

$n+28\cdot128+4\cdot200=n+4384$

この値は概要で述べた「小量の作業領域」といえる大きさであると考える．

7. 種々の計算機上での性能評価

種々の計算機上での mps の性能を**表6**に示す．各計算機上で実際に動いている qs と比較するために，mps と UNIX の qsort 関数の処理時間の比 (mps/qsort) を示している．この mps 中の qs は，UNIX の qsort を使用している．「O−」は mps を普通にコンパイルした場合で，「O+」は最適化オプションつきでコンパイルした場合である．mps の処理時間は qsort の約 50% であり，mps が高速であることが分かる．

8. ま と め

mps は要素の大きさが 20 バイト以上，要素数が 700 以上であれば種々のデータに対して qs 以上の性能を発揮する．そうでない場合も qs の性能は得られる．多くの計算機上の qsort 関数が mps アルゴリズムに置き換えられることを期待する．

参 考 文 献

1) Hoare, C. A. R.: Quicksort, *Comput. J.*, Vol. 5, No. 4, pp. 10-15 (1986).
2) Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, p. 722, Addison-Wesley, Massachusetts (1982).
3) Wirth, N.: *Algorithms+Data Structures=Programs*, p. 366, Prentice-Hall, New Jersey (1976).
4) Sedgewick, R.: Implementing Quicksort Programs, *Comm. ACM*, Vol. 21, No. 10, pp. 847-857 (1978).
5) GNU, Free Software Foundation, Inc., Cambridge, Massachusetts.

付録1

$E(f,n,b)=\dfrac{n+b}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\sum_{k=1}^{n}\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]f(k)$ の証明

長さ n の配列を b 個の区間に分割したとき，各区間に評価関数 f を適用した値の合計の期待値を $E(f,n,b)$ と表す．

$E(f,n,b)$ を求める問題は，正整数 n を b 個の非負整数に分割する問題に帰着できる．

$$\left\{S_1,S_2,\cdots,S_b\,\middle|\,\sum_{i=1}^{b}S_i=n\right\}\qquad(S_i:\text{非負整数})$$

分割方法の数は $\left[\begin{matrix}n+b-1\\ b-1\end{matrix}\right]$ であることから，分割がランダムに起こる場合

$$E(f,n,b)=\frac{1}{\left[\begin{matrix}n+b-1\\ b-1\end{matrix}\right]}\sum_{\left\{S_1,S_2,\cdots,S_b\,\middle|\,\sum_{i=1}^{b}S_i=n\right\}}\cdot\sum_{i=1}^{b}f(S_i)$$

である．ここで全分割方法を対象にすれば正整数 S_i が j 個ある場合，正整数 k $(1\leqq k\leqq n)$ はその中に

$$\begin{aligned}&\sum_{j=1}^{b}\left[\begin{matrix}b\\ j\end{matrix}\right]j\left[\begin{matrix}n-k-1\\ j-1-1\end{matrix}\right]\\&=b\sum_{j=1}^{b}\left[\begin{matrix}b-1\\ j-1\end{matrix}\right]\left[\begin{matrix}n-k-1\\ n-k-j+1\end{matrix}\right]\\&=b\left[\begin{matrix}n+b-k-2\\ n-k\end{matrix}\right]=b\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]\end{aligned}$$

回現れるから

$$\begin{aligned}E(f,n,b)&=\frac{1}{\left[\begin{matrix}n+b-1\\ b-1\end{matrix}\right]}\sum_{k=1}^{n}b\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]f(k)\\&=\frac{n+b}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\sum_{k=1}^{n}\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]f(k)\end{aligned}$$

となる．

付録2

$E(\text{qs\_cmp},n,b)\sim\alpha(n+b)(\log_2(n-1)-\log_2(b)+0.60995)$ の証明

付録1において $f(n)=\text{qs\_cmp}(n)=\alpha n\log_2(n)$ の場合

$$\begin{aligned}E(f,n,b)&=\frac{n+b}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\sum_{k=1}^{n}\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]\alpha k\log_2(k)\\&=\frac{\alpha(n+b)}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\left[\sum_{k=1}^{n}\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]k\ln(k)\right]/\ln(2)\end{aligned}$$

ここで $\left[\begin{matrix}a+1+n\\ a+1\end{matrix}\right]\sim\dfrac{a^n}{n!}$ を使って，$\left[\begin{matrix}n+b-k-2\\ b-2\end{matrix}\right]\sim\dfrac{(n-1-k)^{b-2}}{(b-2)!}$ を代入し，$(n-1-k)^{b-2}$ を2項展開すると

$$E(f,n,b)\sim\frac{\alpha(n+b)}{\left[\begin{matrix}n+b\\ b\end{matrix}\right]}\left[\frac{1}{(b-2)!}\cdot\sum_{r=0}^{b-2}\left[\begin{matrix}b-2\\ r\end{matrix}\right](-1)^r(n-1)^{b-2-r}\sum_{k=1}^{n}k^{r+1}\ln(k)\right]\Big/\ln(2)$$

さらに $\sum_{k=1}^{n} k^r \ln(k) \sim \frac{n^{r+1}}{r+1}\ln(n) - \frac{n^{r+1}}{(r+1)^2} + \frac{1}{2} n^r \ln(n)$

を使い

$$\sum_{r=1}^{n} \left[\begin{matrix} n \\ r \end{matrix} \right] (-1)^{r-1} \frac{1}{r} = \sum_{r=1}^{n} \frac{1}{r} \sim \ln(n) + \gamma$$

（ここで γ はオイラー定数 0.57721… である.）

を代入し，整理すると

$$E(f, n, b) \sim \frac{\alpha(n+b)}{\left[\begin{matrix} n+b \\ b \end{matrix} \right]} \frac{(n-1)^b}{b!} (\ln(n-1) - \ln(b) + 1 - \gamma)/\ln(2)$$

ここで再び $\frac{(n-1)^b}{b!} \sim \left[\begin{matrix} n+b \\ b \end{matrix} \right]$ を代入して

$$E(f, n, b) \sim \alpha(n+b)\left(\log_2(n-1) - \log_2(b) + \frac{1-\gamma}{\ln(2)}\right)$$
$$= \alpha(n+b)(\log_2(n-1) - \log_2(b) + 0.60995)$$

となる.

（平成 6 年 2 月 23 日受付）
（平成 6 年 9 月 6 日採録）

河村 知行（正会員）

1953 年生．昭和 51 年東京教育大学理学部応用数理学科卒業．昭和 54 年筑波大学大学院博士課程数学研究科修士取得退学．昭和 54 年徳山工業高等専門学校情報電子工学科助手．昭和 56 年同講師．昭和 60 年同助教授．現在に至る．理学博士．計算機アーキテクチャ，オペレーティングシステムに興味をもつ．

江口 賢和（正会員）

1946 年生．1969 年山口大学工学部電気工学科卒業．1971 年同大学院電気工学専攻修士課程修了．1974 年九州大学大学院電子工学専攻博士課程単位取得退学．工学修士．同年九州工業大学情報工学科助手．現在，徳山工業高等専門学校情報電子工学科助教授．データベースシステムにおける並列実行制御に関する研究に従事．電子情報通信学会会員．

小笠原基泰

1939 年生．昭和 38 年九州大学理学部数学科卒業．昭和 38 年日本大学理工学部数学科副手．昭和 40 年同助手．昭和 41 年日本大学生産工学部統計学科助手．昭和 44 年日本電子工学院電子計算機部ソフトウェア科教師．昭和 51 年徳山工業高等専門学校情報電子工学科助教授．昭和 60 年同教授．統計モデリングに興味をもつ．日本数学会，日本統計学会，応用統計学会，日本オペレーションズリサーチ学会各会員．

重村 哲至（正会員）

1964 年生．1987 年豊橋技術科学大学工学部情報工学科卒業．1989 年豊橋技術科学大学大学院工学研究科情報工学専攻修了．工学修士．1989 年より徳山工業高等専門学校助手．