アルゴリズム 71−10
（2000．1．17）

葉数最適整列法LOASの実用化方式

伍 偉鴻[†]　　二村良彦[††]

[†]早稲田大学理工学研究科
[††]早稲田大学 理工学部 情報学科

概要

現在，実際的に最速と考えられている整列法はBentleyのQuicksort（BQ法）である．本稿では，整列済みに近いデータに対してはBQ法の約2倍高速であり，かつ一様乱数列に対してもBQ法よりも高速な整列法LOAS（Leaves Optimal Adaptive Sort）の２つの実現法について報告する．一つは高速であるがスペースをO(*N*)要し、もう一方は性能は多少落ちるが、スペースをO($\sqrt{N}$)要するものである。LOASは，数列の葉（数列において自分より小さい隣接要素を持たない要素）の数について最適な整列法である．即ち，数列の長さと葉数を各々*N*および*m*とすると，LOASはO(*N* log *m*)時間で整列を完了する．実用に供されている４つの整列法(BQ法，GNU Quicksort，GNU Merge sort，多重分割ソートMPS)を含むいくつかの整列法と比較することにより，LOASの高速性を示す．

Practical Implementations of Leaves-Optimal Adaptive Sort

Wai Hong NG[†]　　Yoshihiko FUTAMURA[††]

Graduate School of Science and Engineering, Waseda University[†]
School of Science and Engineering, Waseda University[††]

Abstract

Two implementations of LOAS(Leaves Optimal Adaptive Sort) are proposed. One implementation is optimized in running time which needs O(*N*) extra working space and is faster than the fastest Quicksort known, Bentley's Quicksort, by a factor of 2 in practice, the other needs O($\sqrt{N}$) extra working space which is more practical but loss in efficiency. LOAS runs in O(*N* log *Leaves*) time and is optimal with respect to the presortedness measure *Leaves* which is the number of elements smaller than their neighbors in a given sequence. Evaluation of LOAS together with four other sorting algorithms (Bentley's Quicksort, GNU Quicksort, GNU Merge Sort and Multi Partition Sort MPS) is conducted to show the efficiency of LOAS.

1．はじめに

葉数最適整列法LOAS（Leaves-Optimal Adaptive Sort）[2]は，マージに基づく整列法である．それは与えられた数列をまず葉数個の上昇部分列に分割する（ただし数列の葉とは，数列において自分より小さい隣接要素を持たない要素である）．次に，マージソートと同様に，分割された部分列を2つずつマージし，最終的に1つの上昇列が得られるまでマージを繰り返す．数列の長さと葉数を各々Nおよびmとすると，その計算量は$O(N \log m)$であり，これは葉数について最適である[3]．従って，LOASを実現するに際して，適正なマージ戦略を選択することが重要である[2]．従来のマージ戦略の多くは，素朴な方法に基づき，比較回数と使用スペースの減少を重点において改良したものであった．しかし我々は，部分列をマージする際に起こるデータコピーおよびデータの参照局所性が，マージに基づく整列法をクイックソートなどより遅くしている決定的要因であることを理論的考察および実験により確認した．本稿では，我々がLOASを実現する際に解決した，データコピーオーバーヘッドおよびデータの参照局所性に関する問題およびその解決法について述べる．

数列の長さと葉数を各々Nおよびmとすると，LOASの計算量は$O(N \log m)$であり，これは葉数について最適である[3]．さらに$1 \leqq m \leqq \lceil (N+1)/2 \rceil$かつ一様乱順列の平均葉数は$(N+1)/3$であることが知られている[3]．葉数の定義より正順および逆順の数列の葉数は1である．また，整列済みに近い数列の葉数も1に近ずく．Knuth[7]にある通り，現実の整列は整列済みに近いデータに対して行われる場合が多いので，整列法の評価は葉数の少ない数列に対しても行われる必要がある．従って我々は，長さNの順列の葉数mを$1 \leqq m \leqq (N+1)/2$の間で変えながら，ランダムに生成し，それを用いて4つの整列法(現在最速と思われているBQ法[4]，広く使われているGNU QuicksortとGNU Merge sortおよび国産では最速と考えられる多重分割ソートMPS[1])のキー比較回数および実行時間について比較評価を行った．その結果，各種のテクニックを施したLOASが，比較回数および実行時間において他の整列法よりも優れていることを示せた．なお，葉数を制御した乱順列の生成法については文献[3]およびランダムデータサーバーhttp://www-gpc.futamura.info.waseda.ac.jp/~yanoh/RDS/を参照されたい．

2．LOASのアルゴリズム

LOASは与えられた長さNの数列を葉数個の区間に$O(N)$時間で分割し、次に得られた葉数個の区間を$O(N \log$ 葉数$)$時間でマージする整列法である．例えば数列X=<7,1,2,6,5,3,4>を次のような2つのフェーズにより整列する．

フェーズ1：まず数列を左からスキャンし下降列と上昇列のペア列に分ける．次に各ペアをマージし葉数個（この場合は2個）の上昇列を作る．（<1,2,6,7>，<3,4,5>）

フェーズ2：フェーズ1で得られた葉数個の上昇列に対してマージを繰り返し、最終的に一つの上昇列を得る．

このアルゴリズムの最悪計算量は$O(N \log$ 葉数$)$であることは明らかである．そして，それが文献[6]等の意味において葉数に関して最適であることも証明されている[3]．またメモリ所要量は高々葉数個の部分列を管理するために$\lceil (N+1)/2 \rceil$，そしてマージのためのコピー領域に$\lceil N/2 \rceil$の合計Nである．

3．LOASの弱点

葉数最適整列法LOASは基本的にはマージに基づいた整列法で，マージ戦略の選択は重要であるとされてきた．しかし，LOASは比較回数が他の整列法より少ないものの実行時間についてはQuicksort等に負けていた[2]．その一因は，部分列をマージする際に起こるデータコピーのオーバーヘッド，およびマージに基づいた整列法に特有な局所参照性の悪さにあると考えられる．以下では，これらの問題の解決方法について報告する．

4．データコピーオーバーヘッドの削減

データコピーオーバーヘッドを削減するには2本の部分列をマージするより何本かをまとめて

いわゆるマルチウェイマージという方法が効果的である．マルチウェイマージを形式的に実現するアルゴリズムとしてはリプレースメントセレクション（置換え選択）が知られている[5]．しかし，リプレースメントセレクションによるマルチウェイマージはレコードのサイズが極端に大きくなければ，改善にならないだけでなく，逆に素朴なLOASより2，3倍遅いことが実験により判明した．なぜなら，m－ウェイマージを具現する時に使う順位キューの高さは$\lceil \log m \rceil$であるため一つの要素を出力するのに平均$\lceil \log m \rceil$回の置換えが必要だからである．これは葉数m長さNの配列をリプレースメントセレクションでm－ウェイマージをすれば約$N\lceil \log m \rceil$回のキーの置換えつまり$3N\lceil \log m \rceil$回のキーのコピーが必要になる（ヒープへの入出力に各1回，比較の際に最低1回）．これは，改良される前のLOASのキーのコピー回数$N\lceil \log m \rceil$回より3倍も多く，理論的観点からも改良にはならない．

リプレースメントセレクションのオーバーヘッドを無くすにはDecision Tree（図１）を使って最小要素を見つけることによりマルチウェーマージを実現すればよい．Decision Treeによるマルチウェイマージはリプレースメントセレクションで状態を覚える順列キューの代わりにプログラムに覚えさせる．それにより，リプレースメントセレクションの置換えオーバーヘッドを無くすことができる．その上，順位キューを生成する際の比較のオーバーヘッドもない．しかし，m－ウェイマージのDecision Treeのパスの長さはm-1であり葉の数は2^{m-1}になる．従って，mが5以上の場合は現実的にはプログラムでの具現が困難であるという問題がある．

もう一つの改善方法としてはコピー領域の交代使用が考えられる．在来のLOASは2本の部分列のうち，短い方をコピー領域にコピーし元の配列にマージすることで所要コピー領域はN/2だった[2]．コピー領域の交代使用によりデータコピーの回数を1/2に削減できるものの，在来のLOASの所要コピー領域$N/2$より大きく最悪Nになるというデメリットがある．

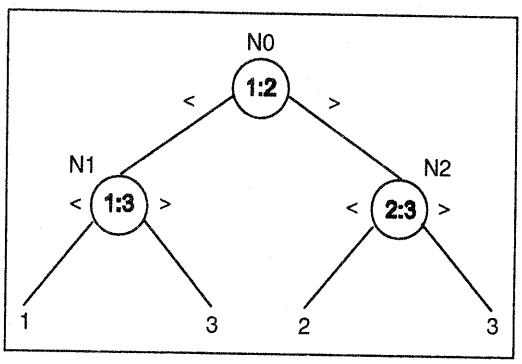

図1： 3-ウェイDecision Tree

5．参照局所性の改善

LOASが他の整列法，特にQuicksort，より遅いもう一つの原因としては参照局所性の悪さが考えられる．現行コンピュータのアーキテクチャでは，コストパフォーマンスのトレードオフをカバーするために階層メモリが使われている．低いレベルにあるキャッシュメモリは一般的にメインメモリより何倍も高速であるためキャッシュミスによるペナルティは相当大きいと考えられる．LOASはこの面においてはQuicksortに劣っている．LOASとQuicksortはともに一定の方向（図２）にキーを読みかつ比較してキャッシュミスを起こす．しかし良く調べてみると，LOASではマージをする際に使うコピー領域でキャッシュミスが起こる．一方，Quicksortではスワープをする際に使う領域でそれがほとんど無いということがわかる．即ち，LOASの方がキャッシュミスの機会が多いということである．これはマージに基づいた整列法の宿命である．これを改善するには，ソートする配列をできればキャッシュメモリの収まるサイズでいくつかの小さいな配列に分割する．そして各小配列をLOASでソートし，そこで得られた整列済みの配列をマージし最終的に一つの上昇列にすればよい．この方法の弱点は配列を分割するとき，切り口がu節(片方の隣人より大きくかつもう一方の隣人より小さい要素)[2]の場合は葉数が確実に1増えてしまうことである．従って，葉数が増える確率は$(N-2m+1)/N$以上であるので，元の葉数が小さいほど増加する葉数が多いことがわかる．言い換えると，葉数が小さい場合は無駄な

マージを行って比較回数とデータコピーの回数を増やす可能性が高い．例えば，元の配列を長さlの部分配列に分割する場合を考える．この時，元の配列における長さ$2l$以上の部分列を分割することにより，葉数を2以上増やす可能性がある．これを回避するには次のようにすれば良い．即ち，整列済みの部分配列をマージする際に，元々1つの配列が分割して出来たものであるか否かをチェックし，適応的マージを行う．ただし，適応的マージとは，例えば左右に隣接して並んだ2つの配列を，要素の移動をできるだけ少なくしてマージする方法である[6].

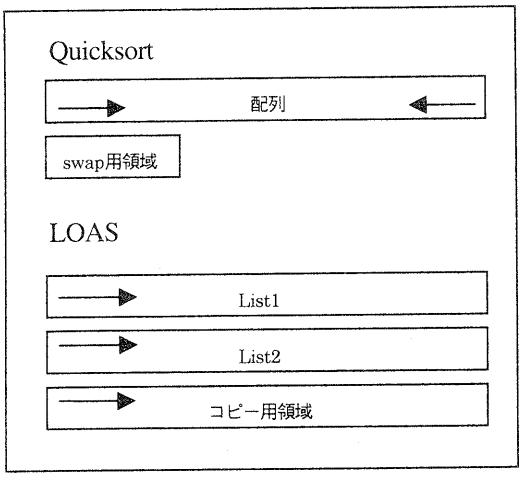

図2：QuicksortとLOASの参照局所性：QuicksortおよびLOASとも矢印の方向に配列をアクセスする．しかし，Quicksortではswap用領域がキー1つ分のサイズに固定されているのに対して，LOASではサイズO(N)のコピー用領域にも矢印の方向にアクセスしキャッシュミスを起こす．

6．改良後のLOASのアルゴリズム

以上に説明した改良方法を用いてLOASの諸弱点を改良し，LOASのアルゴリズムを変更した．改良後のLOASのアルゴリズムは次のとおりになる．

フェーズ1．数列をｓ個の部分数列に分割する．

フェーズ2．フェーズ1で得られたs個の部分数列それぞれに対してフェーズ2aとフェーズ2bの手順を取る．

フェーズ2a．左からスキャンし下降列と上昇列のペア列に分ける．各下降列と上昇列のペアをマージし葉数個の上昇列を作る．

フェーズ2b．フェーズ2aで得られた葉数個の上昇列をコピー領域交代使用マルチウェイマージでマージし一つの上昇列を得る．

フェーズ3．フェーズ2で得られたs個の上昇列をコピー領域の交代使用によるマルチウェイマージでマージし一つの上昇列を得る．

改良後のLOASアルゴリズムの最悪計算量は明らかに従来のLOASと等しくO(N log 葉数)である．メモリ所要量はフェーズ1とフェーズ2では高々葉数個の部分列を管理するために[(s+1)/2]と,マージのためのコピー領域にsである．フェーズ3ではs個の部分列を管理するためにs，コピー領域にNである．フェーズ1とフェーズ2の操作はフェーズ3の操作とは独立であるため，メモリの共用は可能である．従って，総メモリ所要量は$s+N$である．

7．メモリ所要量の改善

改良後のLOAS（LOAS4CA）は従来のLOAS（LOAS）と比べて実行時間（図3）の面において改良にはなっているが，しかし，メモリの所要量は約N/2増加し$s+N$になってしまった．実用性を高めるには実行時間だけでなくメモリ所要量にも配慮する必要がある．メモリ所要量を減少するには，マージアルゴリズムの改良が必要である．しかし，メモリ消費の小さいマルチウェイマージアルゴリズムは我々の調べた範囲では見つけることができなかった．従って，我々が新規にメモリ消費が低くかつ高速なマージアルゴリズムを開発している．現段階では，フェーズ3のマージアルゴリズムをメモリ消費O($\sqrt{N}$) [5]のマージアルゴリズムと置換し，LOASのメモリ消費をO($\sqrt{N}$)まで抑えることができた（LOAS4CAM）．しかし，このマージアルゴリズムはマルチウェイマージではないので，データコピーのオーバーヘッドが大きいため，性能の面においてはメモリ消費の大きい（LOAS4CA）より多少劣っている．しかし，実用領域においては依然として他方式より断然高速であるため，我々がこのLOAS（LOAS4CAM）でも，十分実用性が高いと考えている．

8．他整列法との性能比較評価

本節では，Bentley Quicksort（BQ法）[4]，GNU Quicksort（GNU Q法），GNU Merge Sort（GNU M法），MERGE4C(LOAS4Cと同じマージ方式をとり，我々自身が開発したもの)，多重分割ソートMPS[1]，従来の葉数最適整列法LOAS，Decision Treeを利用した4－ウェイマージのLOAS4，LOAS4に参照局所性を高める改良を施したLOAS4CおよびLOAS4Cに適応マージを追加したLOAS4CA，LOAS4CAのメモリ消費を減少したLOAS4CAMの10種類の整列法の性能比較を行う．性能評価は普通のパソコンで行った．具体的には，CPUはインテルアーキテクチャであるAMD K-6 233mhz，メインメモリ128MBと二次キャッシュ512KBのスペックのマシンを選んだ．OSはWindows NT 4.0 SP5である．コンパイラはVisual C++ 6.0で，すべてのプログラムはmaximize speed オプションをつけてコンパイルした．各整列法をプログラミング言語Cのライブラリにあるqsort関数の仕様で実装し，葉数を制御した一様乱数列を用いて評価した．LOAS4CとLOAS4CAでは部分配列の長さは$\sqrt{N}$で分割した．性能評価の比較対象として選ばれた整列法およびその選定理由は下記の通りである．BQ法[4]は現在最速と思われるQuicksortである．これは文献[9]よりも最適化されており，軸の選び方は3要素の中間値法でなく配列のサイズによって適応的に3要素の中間値法と9要素の中間値法をトグルで切り替える．また等しいキーの場合でも従来のQuicksort（GNU Q法など）と違って等しいキーを配列の中央に集めるより，配列の両端に集める方法を採用している．この改良によりBQ法は一般の最適化されたQuicksort（例えばGNU Q法等）よりも安定している．即ち，一様乱数列のように葉数が$N/3$の周辺の入力だけが高速になるといった偏った性質を持たず，全ての葉数領域において高速に整列できるようになっている（図4と図5）．GNU Q法とGNU M法はGNUのgccコンパイラのCライブラリにあるqsort関数のインプリメンティションである．これらは一般に広く使用されているQuicksortとMerge Sortのインプリメンティションと考えられる．多重分割ソートは基本的にはQuicksortではあるが，国産の高速整列法の代表として，評価対象に取り上げた．

図4は比較回数を示す．LOASは葉数適応なので，葉数が小さいときには他の葉数適応でない整列法より比較回数が少ないのは当然である．しかし，葉数が最大である$N/2$のときでも比較回数が他の整列法より少ないことに注意されたい（数列の葉数は$N/2$以下であり,かつ一様乱数列における平均葉数は約$N/3$である[2]）．図5は実行時間を示す．素朴なLOASは葉数が比較的少ないとき以外はBQ法[4]に負けているが，各種の改良を施したLOASではLOAS4，LOAS4C，LOAS4CAの順で速くなっていく．特にLOAS4CAはLOAS4Cより，葉の数が少ない場合でもLOASの長所を保持することができた．LOAS4Cと同じマージ方式を採用したMERGE4Cは，GNU M法よりも各段に速くなり，LOAS4CAに続く高速性を実現している．LOAS4CAMはLOAS4CAよりデータコピーオーバーヘッドが大きいため，LOAS4CAより多少劣っているが，実用領域においては他方式より高速である．図5から明らかな通り，LOAS4CAが全領域において他方式より早く，かつ葉数が比較的少ない現実的な領域においては現在最速とみなされているBQ法[4]より約2倍速い．我々は実際に現実的にソートの対象となるデータの葉数を計測してはいない．しかし，Knuth[7]にも述べられている通り，現実的データは整列済みに近いものが多く，その葉数は一様乱数列の葉数$N/3$より，遥かに小さい（$\sqrt{N}$ 以下）と考えている．

9．おわりに

他の一般に使われている整列法より高速なLOASを実現するための方法を2つと，その比較評価結果を報告した．しかし，現在の最高速LOAS（LOAS4CA）の実現法ではメモリ消費の面においてQuicksortに劣っている（Quicksortの$O(\log N)$に対してLOAS4CAは$O(N)$）．LOAS4CAの更なる高速化およびメモリ消費量の低減を実現するマージアルゴリズムを目下開発中である．完成次第，追って報告する予定である．

参考文献

[1] 河村，江口，小笠原，重村：多重分割ソートアルゴリズム，情報処理学会論文誌，Vol. 35，No.12，1994年.

[2] 二村，二村，遠藤，平井：葉数最適整列法LOASとその実現法，情報処理学会アルゴリズム研究会44-2，95年3月.

[3] 二村，青木，大谷，二村：単純指標を持つ乱順列の高速生成法，日本ソフトウェア科学会誌 第14巻6号，1997年.

[4] Bentley, J.L. and McIlroy, M.D. : Engineering a Sort Function. Software Practice and Experience, 23, 11, 1993, 1249-1265.

[5] Dvorak, S. and Durian, B., Stable linear time sublinear space merging. Comput. J. 30, 4, 1987, 372-375.

[6] Estivill-Castro,V. and Wood, D. : A Survey of Adaptive Sorting Algorithms. Computing Surveys, 24, 4, 1992, 441-476.

[7] Knuth, D.E. : The Art of Computer Programming . Vol.3. Addison-Wesley, Reading, Mass, 1973.

[8] Knuth, D.E. : Structured Programming with goto Statements. Computing Surveys, 6, 4, 1974, 261-301.

[9] Sedgewick, R.: Implementing Quicksort Programs. CACM, 21, 10, 1978, 847-857.

付録　一様乱数列における平均葉数

ここでは１とxの間の一様乱数列で長さがｎのものにおける葉数は$(n+1)/3-(n-2)/(3x^2)$であることを証明する(この証明は[3]からの引用である). 但し等しい値が並んだ時は右側が大きいものとする．まず「既に生成された一様乱数列（長さ２以上）に新たに要素（ｃとする）を追加した時に増える葉数の平均個数」を$p(x)$と置く．この計算には次の性質を利用する．

（１） 直前の要素（bとする）は葉かu節のどちらかに限られる．

（２） 直前の要素 bが cより大きなu節の時だけ葉数が１増加する．

これらの性質から$p(x)$は「直前の要素 bが新たな要素 cより大きい葉である確率」に等しいことがわかる． bの直前の要素をaとすると，仮定よりa,b,c共に１とxとの間の値を等確率を取る．また bがu節であるためにはa以上でなければならない．従って

$$p(x)=\left(\sum_{a=1}^{x}\sum_{b=a}^{x}\sum_{c=1}^{b-1}1\right)/x^3=(1-1/x^2)/3$$

ここで1とxの間の一様乱数列で長さがnのものにおける平均葉数をavel(x,n)とおけば$n>2$の時，

$\mathrm{avel}(x,1)=\mathrm{avel}(x,2)=1$より，

$\mathrm{avel}(x,n)=1+(n-2)p(x)$

$=(n+1)/3-(n-2)/(3x^2)$

(QED)

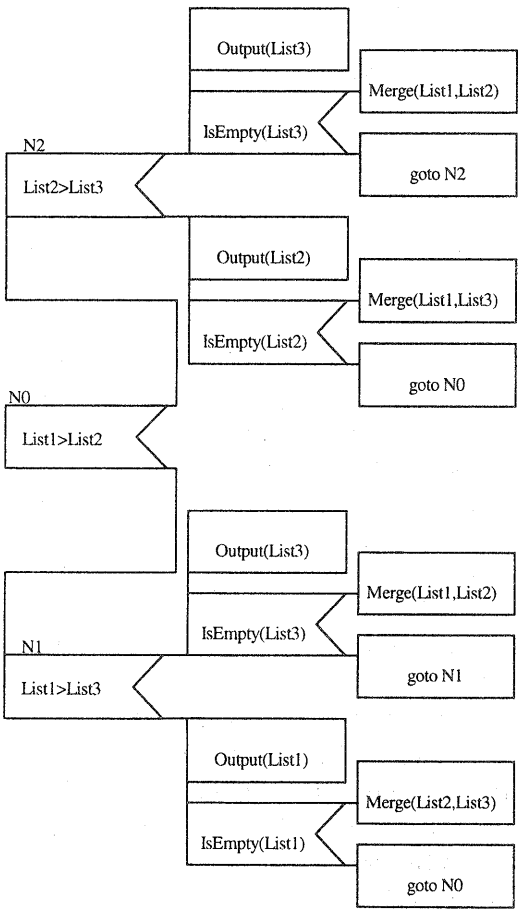

図３：図１に対応したDecision Treeによる3-ウェイマージのPAD

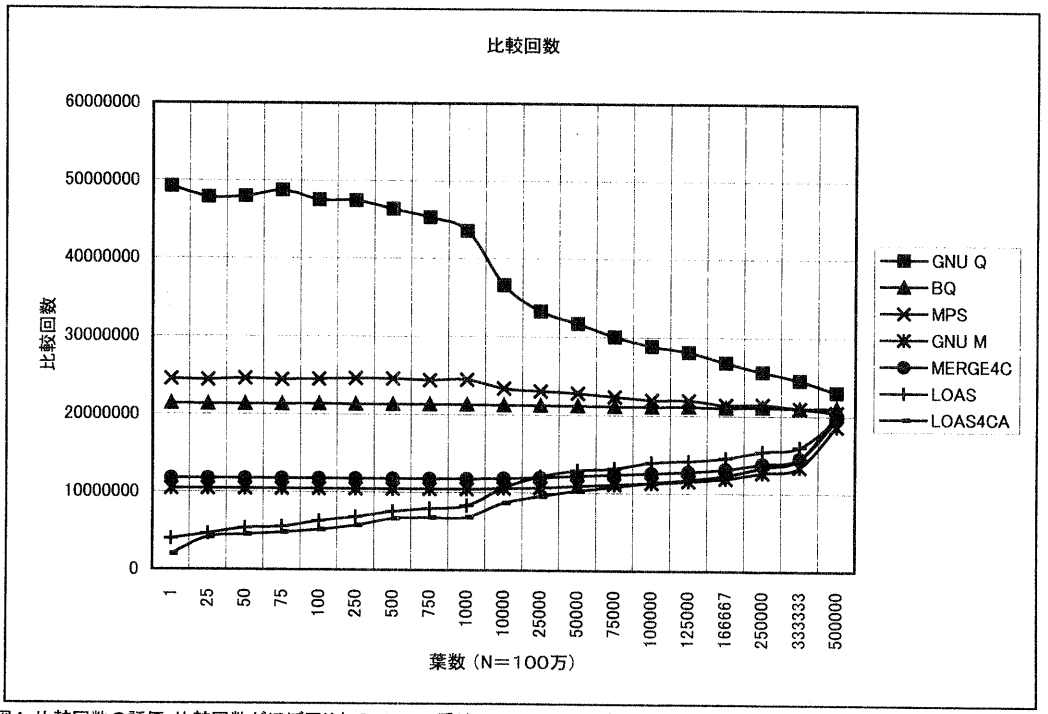

図4：比較回数の評価：比較回数がほぼ同じなのでLOAS系はLOAS4CA,マージソート系はMERGE4Cのみ示されている．（LOAS4CAMはLOAS4CAとほぼ同じため図から除いた）

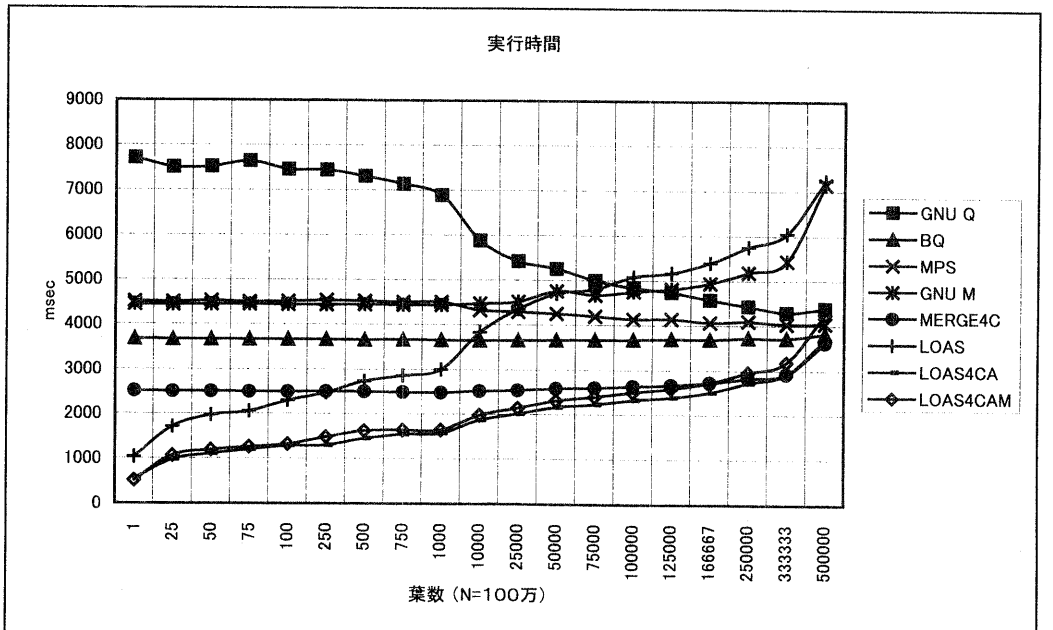

図5：実行時間の評価：LOAS系の5方式中では，LOAS4CAが全領域において最速であるので，LOAS4,LOAS4Cは図から除いた．LOAS4CAは他方式よりも全領域において高速であることが理解できよう．これに一番近い性能を示すものは，LOAS4Cと同様のマージ方式を取るMERGE4Cである．これにより我々のマージ方式が優れていることが理解できよう．