アルゴリズム 44－2
（1995．3．17）

葉数最適整列法 LOAS とその実現法

二村良彦† 二村夏彦‡ 遠藤貢一† 平井利治†

†早稲田大学 理工学部
‡School of Computer and Information Science, Syracuse University

数列が整列されている度合を表す事前整列性測度 Leaves(葉数), および事前整列性のよい数列については高速にソートをする適応整列法 LOAS とその実現法について報告する．LOAS はまず与えられた数列を葉数個の区間に O(n)時間で分割し, 次にそれを O(n log 葉数)時間でマージする整列法である．本稿ではまず葉数と LOAS を定義し, 次に LOAS が Leaves に関して最適であることの証明を与える．最後に LOAS の実現法およびその他の適応ソートや実用的な QUICK ソート, MERGE ソート等との性能比較結果を示す．

Leaves-Optimal Adaptive Sort (LOAS) and its Implementations

†Yoshihiko FUTAMURA ‡Natsuhiko FUTAMURA
†Koichi ENDO †Toshiharu HIRAI

†School of Science and Engineering, Waseda University
‡School of Computer and Information Science, Syracuse University

We propose a new presortedness measure *Leaves* and a new sorting algorithm LOAS (Leaves Optimal Adaptive Sort) which is optimal with respect to the measure. LOAS divides a given sequence X into Leaves(X) sorted subsequences in O(n) time. Then it merges the sequences in O(n log Leaves(X)) time. Implementation techniques and proof of the Leaves-Optimality of the algorithm are described. In order to prove that LOAS is an efficient sorting algorithm,we have conducted systematic evaluation of several sorting algorithms including Quicksort, merge sort, Skip sort and MEL sort.

1 はじめに

ソーティングは非常に使用頻度の高いプログラムである[11]ので,その研究は長い間続けられている[10]. 現実の問題として数列のソーティングをあつかう場合には,与えられた数列が殆ど整列済みであることが多く[11],最近は「整列済みに近いことをいかにして計測するか」という事前整列性測度(presortedness measure)および整列済みに近い数列ほど速く整列化するアルゴリズム(適応整列法)に関する研究結果が多数報告されている[1,2,3,5,6,13,15]. 以下では与えられた数列を X,そしてその長さを n で表す. 事前整列性測度およびその測度で計られた X の整列性を各々PM および PM(X)とすると,実用的観点から PM は次の 2 要件を満足することが望ましい:

(1)PM(X)は O(n)で計算できること,

(2)PM(X)は n 以下であること.

本稿では上記 2 要件を満足する事前整列性測度 Leaves(または葉数)およびその測度に関して最適な適応整列法 LOAS(Leaves-Optimal Adaptive Sort)について報告する. LOAS はまず与えられた数列を葉数個の区間に O(n)時間で分割し,次にそれを O(n log 葉数)時間でマージする整列法である. 本稿ではまず葉数と LOAS を定義し,次に LOAS が Leaves に関して最適であることの証明を与える. 最後に LOAS の実現法およびその他の適応ソート(MEL ソート[15], SKIP ソート[3])や実用的な QUICK ソート[9],MERGE ソート[11]等との性能比較結果を示す. 得られた評価結果は下記の 2 点である. (1)キーサイズが小さいときは,葉数が極度に小さい場合を除いて QUICK ソートが一番速い. (2)キーサイズが大きい時は葉数が長さの 1 割程度迄なら LOAS が一番速く,それ以上ならば MERGE ソートが一番速い(ちなみに一様乱数列の葉数は長さの約 3 割であり,葉数の最大値は長さの約 5 割である).

以下では長さ n の順列のみをあつかう. 数列 X の長さおよび集合 S の濃度を各々|X|および‖S‖で表す. 対数の底は 2,即ち log n=$\log_2$ n とする. merge(X,Y)は 2 つの整列済み数列 X と Y をマージする関数である. 例えば X=<1,3,5>かつ Y=<2,4,6>ならば merge(X,Y)=<1,2,3,4,5,6>である. 適応整列法に関する既存の表記法および概念については[3]に従う.

2 数列の葉数

葉数は数列において隣接する要素よりも小さい要素(葉)の個数であり正確には Leaves(X)と書く. X1=<7,1,2,6,5,3,4>については,Leaves(X1)=2. 何故ならば 1 と 3 だけが隣接する要素よりも小さいからである. 例えば 2 は右の隣人より小さいが左の隣人より大きいので葉ではない. これは筆者等が[4]で報告した攪乱要因と本質的に同じである(葉数=攪乱要因+1). 葉数を他の事前整列性測度[3]と比べ,例えば,Runs や Inv+1 以下であり Enc/2 以上であることを後述する. 本節の目的は葉数について最適な整列法の計算量は O(n log Leaves(X))であることを証明することである. 葉の数学的性質を調べるためにここでは対称ヒープ[4]を利用する. これはデカルト木[17]の部分集合であって,順列木[16]とも呼ばれる.

定義 1:数列 X に対応する(減少)対称ヒープ T(X)は下記 2 つの性質を持った 2 分木である:(1)T(X)はヒープである,即ち親ノードの値は子ノードの値より大きい. (2)T(X)を対称(または inorder)トラバーサルすることにより,X を生成することが出来る(図 1 参照).

増加対称ヒープも同様に定義できるが,本稿では特に断らない限り,ヒープにより減少対称ヒープを表すことにする. 数列 X から T(X)を O(n)時間で生成するアルゴリズムは例えば[4]を参照されたい.

対称ヒープにおいて,子供を 2 人持つ節を b 節(binary node),子供を 1 人持つ節を u 節(unary node),そして子供を持たない節を葉と呼ぶ. 以下では与えられた対称ヒープに含まれる b 節,u 節および葉の個数を各々b,u および m で表す(n=m+b+u, m=b+1, u=n-2m+1 に注意).

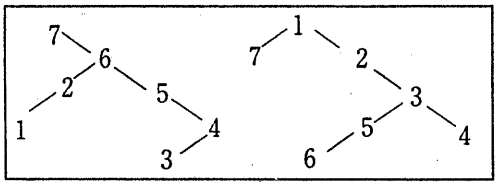


図 1：数列<7,1,2,6,5,3,4>に対する減少(左側)及び増加(右側)対称ヒープ

性質 1：T(X)における b 節,u 節および葉は各々X に於いて両隣より大きな要素,隣の 1 人だけより大きな要素,および葉に対応する．この対応は 1-1 であるので以後は,数列の b 節あるいは u 節と呼ぶ(数列における両端の要素は b 節になれないことに注意されたい)．

性質 2：数列 X の減少対称ヒープの葉数を m,そして増加対称ヒープの葉数を m'とする．この時
(1)X の両端とも減少ヒープにおける葉でなければ m'=m+1,(2)X の両端が減少ヒープにおける葉ならば m'=m-1,(3)X の片端だけが葉であれば m'=m．この性質は,同じ数列の減少ヒープを増加ヒープにすると,両端以外の葉が b 節に変身すると言う事実に基づく．

性質 3：(1) b 節は必ず step down を発生するので m≦Runs, (2) b 節は必ず inversion を発生するので m≦Inv+1, (3) MEL ソート[15]における 2 つ目以降の encroaching list は,b 節および両隣を持つ葉に対応するので Enc≦2m(ただし Enc は常に m より小さいとは限らない．例えば前述の X1 については m=2 に対して Enc =4)．

葉数が上述の PM に関する性質(1)と(2)を満たすことは明らかである．一方 Enc の計算には O(n*log Enc)要す[15]．しかも後述のように Enc 最適な MEL ソートと比べて同等以上の性能を LOAS が既に達成しているので,Enc≦2Leaves であるにもかかわらず我々は Leaves の実用的価値は大きいと判断する．

性質 4：節数 n,葉数 m の順列(または対称ヒープ)の個数を L(n,m)とすれば,木の作り方より明らかに,それは下記の再帰方程式で定義できる：

(1) L(n,m)=0 if m<1 or n+1<2m

(2) $L(n,1)=2^{n-1}$

(3) $L(n,m)=2m*L(n-1,m)+(n-2m+2)*L(n-1,m-1)$ if $1<m\leq(n+1)/2$

補題 1：n が奇数のとき

$L(n,(n+1)/2)>((n-1)/e)^{(n-1)/2}$．ただし e は自然対数の底である．

証明：節数 n のフルの 2 分木(u 節を持たない 2 分木)の個数,即ち L(n,(n+1)/2)を g(n)とする．

$g(n)=L(n-1,(n-1)/2)$ ((3)より)

$=(n-1)*L(n-2,(n-1)/2)$

$+(n-1-(n-1)+2)*L(n-2,(n-3)/2)$

$>(n-1)*L(n-2,(n-1)/2)=(n-1)g(n-2)$．

g(1)=1 であるので $g(n)=(n-1)(n-3)(n-5)\ldots 2*1$

$=2^{(n-1)/2}((n-1)/2)!\geq((n-1)/e)^{(n-1)/2}$ (QED)．

定理 1：m≧1 かつ n+1≧2m ならば,

$L(n,m)\geq((2m-2)/e)^{(n-1)/2}$

証明：n=2m-1 の時は,補題 1 より

$L(n,m)\geq((n-1)/e)^{(n-1)/2}=((2m-2)/e)^{(n-1)/2}$．

n+1>2m のときは，n に関する数学的帰納法により証明する．

(1) Base: n=1 とき,仮定より m=1.

従って,左辺=$2^{n-1}\geq 0$=右辺.

(2) Induction Step: N>0 のとき,m=1 なら上記 2 と同様．m>1 のとき,

$L(n,m)$

$=2m*L(n-1,m)+(n-2m+2)*L(n-1,m-1)$

$\geq 2m*L(n-1,m)$ (n+1>2m より)

$\geq 2(m-1)((2m-2)/e)^{(n-2)/2}$ (n+1>2m より,n≧2m. 従って帰納法の仮定を適用)

$=(2m-2)^{n/2}/e^{(n-2)/2}>((2m-2)/e)^{(n-1)/2}$ (QED)．

性質 5：節数 n の順列が持つ平均葉数は(n+1)/3 である[16]．また 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．ただし等しい値が並んだ時は右側が大きいものとする(証明は付録 1)．

定義 2[12]：ある非負の整数値関数 M (事前整列性測度と呼ぶ) に対して,整列法が長さ n の数列 X を O(max|n, log‖below(X,M,M(X))‖|)のキー比較で整列できる時に限り,その整列法は M 最適であると言

う．ただし，below(X,M,z)={Y|M(Y)≦z かつ|Y|=|X|}．

定理2： O(|X| log Leaves(X))の整列法は葉数最適である．

証明： M が Leaves のときは

$$\| \text{below}(X,M,M(X)) \| = \sum_{i=1}^{m} L(n,i)$$

定理1より $L(n,m) \geq ((2m-2)/e)^{(n-1)/2}$ であるので

$\log \| \text{below}(X,M,M(X)) \| \geq O(n \log m)$ は明らかである(QED)．

定理1より $\log L(n,m) = O(n \log m)$ であるので，LOAS は真の意味で最適であることに注意されたい．

3 葉数最適整列法 LOAS

葉数最適整列法は，筆者などが提案した対称整列法[5]が最初であったと思われる．しかしそれは実用を意識しないで開発したもので，メモリー所要量および実行時間において実用的な整列法(MERGE ソート，QUICK ソート等)より劣っていた．本稿で提案する整列法 LOAS は基本的には[2,3]のマージに基づく整列法の特殊な場合である．それは前述の数列 X1=<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 X を左からスキャンし下降列と上昇列のペア列(<7,1>,<2,6>),(<5,3>,<4>)に分ける．次に各ペアを(単純)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： 上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1つの上昇列を得る．

これがO(n log 葉数)のアルゴリズム従って葉数最適であることは明らかである．またメモリー所要量は高々葉数個の部分列を管理するために[(n+1)/2]，そしてマージのためのコピー領域に[n/2]の合計 n である．LOAS は，数列に関する下記の2つの性質に基づき基本アルゴリズムにおける分割法とマージ法を改良したものである．

性質6： 下り列1，上り列1，下り列2，上り列2，...のように数列を分割した時 merge(下り列 i，上り列 i)の最小値は下り列 i の最後の要素である．

性質7： merge(下り列 i，上り列 i)の最後の要素は merge(下り列 i+1，上り列 i+1)の先頭の要素よりも大きい．またこのようなマージの過程で作られる隣接する2つの列に於いても，左列の最後の要素は右列の先頭の要素よりも大きい．

LOAS 以外の葉数最適整列法としては下記の2つを我々は知っている．(1)Enc≦2Leaves であるので，MEL ソートは葉数最適である．ただし後述の通りキーの比較回数に関しては LOAS より劣っている．(2)対称整列法[5]は最悪O(n*Leaves)のアルゴリズムであるので，Leaves≦log n のとき対称整列法は葉数最適である．対称整列法の抜き出し操作において，勝ちヒープが子供を2人持つ際に，その2人の子供と，負けヒープの3者の中の大小関係をヒープ中に残すことにより，対称整列法の高性能化が図れることを我々は確認している(抜き出し方式については付録2参照)．

4 LOAS の実現法

LOAS は基本的にマージに基づいた整列法であるので，その実現に際しマージ戦略を適切に選ぶことが大切である．マージ戦略の詳細を説明する為にまず2つの数列の重複と適応マージ[3,10]について述べる．以下では z の b 番地から始まり e 番地で終わる上昇列を(b,e)で表す．

数列の重複： 隣接する2つの上昇列(b1,e1)および(b2,e2)の重複とは次の2つの上昇部分列(ob,e1)および(b2,oe)である(b2=e1+1 に注意)．ただし z(ob-1) ≦ z(b2)<z(ob) かつ z(oe)<z(e1) ≦ z(oe+1)．例えば<1,2,3,4,6,7,9>と<5,8,10>の重複は<6,7,9>と<5,8>である．

適応マージ： 2つの上昇列をマージする際に重複以外の要素を移動させないマージ法を適応マージと呼ぶ．配列を用いるマージ法ではデータのコピーが大きなオーバーヘッドである．適応マージはそれを減らす為の方法である．

LOAS のマージは常に重複を持つ数列に対して行われる(性質6および7)ので重複の有無の判定が不要である．これにより整列に必要なマージの回数(約

葉数回)分のキー比較が節約できる．そして次のようにすれば効率良く適応マージをすることができる．

LOAS における適応マージ：マージすべき 2 つの上昇列を(b1,e1)および(b2,e2)とする．ここでまず 2 者のうちで短い方,例えば(b2,e2),の中の重複の端点 oe を見つける．この時(oe+1,e2)は移動不要であることが分かる．従って(b2,oe)のみをコピー領域に移動する．また z(e1)を oe 番地に移動する．次に(b1,e1-1)と(b2,oe)を後尾から b1 番地から oe-1 番地にマージする．この方法により(b1,ob-1)も移動されないことに注意されたい．(b1,e1)の方が短い場合はマージを数列の先頭から行う．短い方を移動させるので,コピー領域は n/2 あれば十分である．

なお,複数の数列をマージする際に数列が奇数個ある場合には,1 番目または 3 番目の数列で長い方をそのまま残しておき,残りの偶数個の数列をマージする．これは一番最後の数列が短いままで残るのを避けるための方策である．

5 マージ戦略の比較

上述の基本マージ戦略に対して下記の 2 点を変更する方式を各種開発し,基本戦略と性能比較した．

(1)基本戦略では数列の重複の一端,例えば oe,を順次探索で求めるのに対し,i)2 分探索法で求める方式．および ii)マージすべき数列の一方が長さ 1 のとき,他方への挿入個所を 2 分探索法で求める方式．

(2)基本戦略では適応マージを行うのに対し,2 分法の威力を期待して下記の 2 方式を開発した．

i)2 分マージ[11]を行う方式(基本戦略との比較を図 2 中の LABM により示した．図中では LOAS が基本方式の性能を表す．参考のために普通の MERGE ソート[11]の評価データも図に含めた)．および

ii)適応マージを行うが,重複の両端 ob と oe を見つけてから,(ob,e1-1)と(b2+1,oe)を 2 分マージする方式(基本戦略では重複の一端のみ見つければよい)．

我々の実験結果では(1),(2)共に基本戦略の改善にはならなかった．特に(2)では大幅に性能が低下した(図 2 の LABM 参照)．実験の際に使用されたテストデータは文献[7]の方式 2 によって生成した．今後より精密なテストをし,その詳細について報告する予定である．

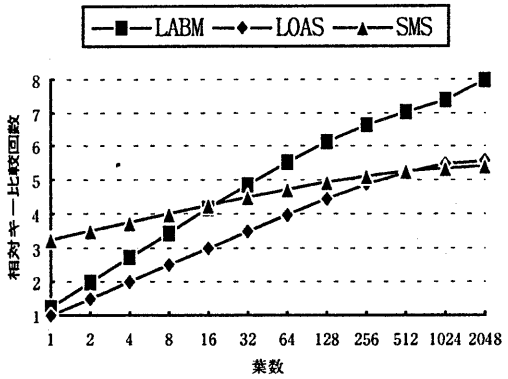


図 2:長さ 4095 の数列の整列に要するキー比較回数

LOAS は基本戦略,LABM はマージに 2 分マージを利用したもの,そして SMS は普通の MERGE ソートを表す．葉数 1 に対する LOAS の比較回数(約 8190)により,全ての観測データを正規化して示した．

6 他方式との性能比較評価

数列の葉数を制御した乱数列を用いた整列法の評価は[1]にもあるが,ここではそれよりもさらに系統的な評価を行う．各種整列法の性能評価を行なう際の大きな問題は,アルゴリズムによる差とプログラミング技術による差が混じりあった結果が得られることである．従って正確の最を論じることは困難であるので,ここではおおざっぱに各方式の傾向を調べた．

我々はさきに,自然 MERGE ソート[11]およびその改造型の RUNS 最適整列法に関して LOAS との性能比較を行い,それ等が実用的観点から LOAS を凌ぐものでないことを示した[6]．ここで新たに LOAS との性能比較するのは下記 2 つの整列法である：

(1)MEL ソート:マージを基本とした整列法で,入力列を Encroaching List(以下 Enc.List)と呼ばれる昇順列に分割し,マージする．Enc.List の数を Enc(X)とおけば O(n log Enc(X))で分割可能である．従っ

て整列もO(n log Enc(X))となり,Enc 最適である.
(2)SKIP ソート[3]:挿入整列法において,要素の挿入場所の探索に2分法を適用でき,かつ平均O(n log n)で整列可能にするために,skip list[14]を用いたもの.これはたとえばRuns 最適である.

これ等の新しい適応整列法に加え,実用的な MERGE ソートと QUICK ソートも一緒に評価した.QUICK ソートは軸選択をランダムに行うものとした.

各種整列法の性能評価をするために,2種類の方法で順列の列を生成した.1つの方法は理論的に一様に順列を生成するO(順列の長さ*葉数)のもの([7]の方式 2)であり,他方は一様性の保証のない高速簡便法でO(数列の長さ)のもの([7]の方式 3)である.両方式により葉数を 1 から 2048 まで変えながら長さ 4095 の順列を,葉数あたり 50 個ずつ生成した.そして各順列の整列に要するキーの比較回数と実行時間を計測した.そして葉数1の順列に対するLOASの計測結果によって他の葉数の計測結果を正規化した後図示した(図 3,4,5).各図における濃線が方式2[7]によって生成された順列に対する結果を示し,淡線は簡便法によるものを示す(両者の最は非常に小さく,小さなグラフ上で差違を認識することは困難である).

方式 2 の生成法は理論的には一様性が保証されているが,実際に大きな n に対する計算においては,桁落ちによる計算誤差が発生し,正しく順列が生成される保証はない.その場合に一様性を確認するためには,例えば n=4095,葉数 2048 のとき,約 1516^{2047} 以上の自由度を持つ乱数列のランダムネスを調べる必要があり,我々はその方法を知らない.実際の評価結果(図 3,4,5)を調べてみると,両方式による結果に大差はない.従って我々は次のような予想を立てている.「整列法の性能評価の目的で長い順列を生成するには,[7]の簡便法で十分である」.図 3 は,必要なキーの比較回数を示している.キーの比較回数は,整列法の細かい実現法にあまり依存しないものである.従って大体の傾向として,SKIP ソートや QUICK ソートが無駄な比較を多く行っていることを示す(従来も性能比較では一様乱数列が用いられることが多く,その葉数は常に約 n/3 であり,QUICK に有利な偏った結論しか得ることができなかった).図4は,キーの長さが小さいとき(2バイト程度)の実行時間の比較であり,キーの比較以外の操作に要するいわゆるオーバーヘッドの大きさを示す.QUICK ソートのオーバーヘッドの小ささは,キーの比較回数を犠牲にして達成されている.また MEL ソートのオーバーヘッドが小さいのは Enc.list の実現にリンクドリストを用いたからである.リンクドリストを用いれば,他のマージベースの整列法の時間に関するオーバーヘッドを減らせることが予想できる.図5は,キーの長さを 50 バイト程度に長くしたときの実行時間の比較である(ただしキーの移動時間は無視した).キーの比較に時間が長くかかると,本質的に比較回数の少ない LOAS が有利になる.特に葉数が少ない場合における LOAS の高速性は顕著である.

7 おわりに

上述のような各種実験の結果,LOAS は今後の改良により実用的高速整列法になるという感触を我々は得た.特に,数列の性質 5 で示したように,一様乱数列の葉数は平均約 n/3 であるので,m=n/3 の近辺でも他の整列法より速くすることを目標に,LOAS を改造中である.一方,現実の世界で大量にソートされているデータの葉数を調べることも今後の課題である.

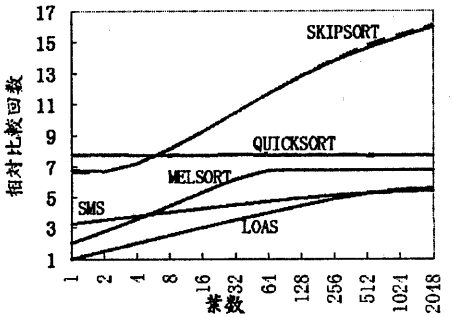


図3:長さ4095の順列の整列に要する比較回数
葉数 1 に対する LOAS の比較回数(約 8190)で正規化して示した.ただし SMS は MERGE ソートを表す.

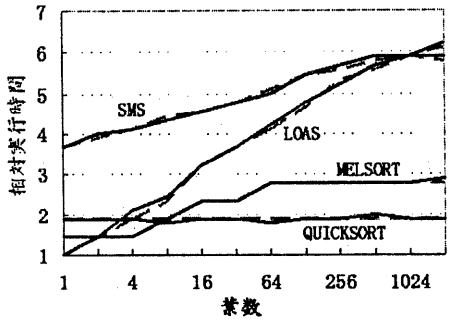


図4:長さ4095の2バイト長順列の整列時間

葉数1に対するLOASの実行時間で正規化して示した.

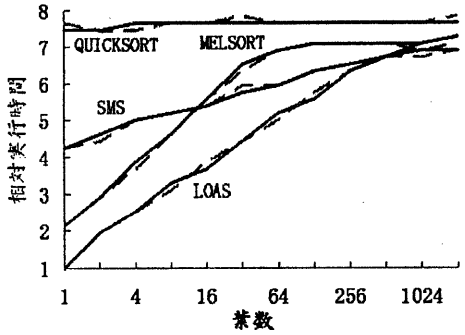


図5:長さ4095の50バイト長順列の整列時間

葉数1に対するLOASの実行時間で正規化して示した.

参考文献

[1]浅野：各種ソーティグアルゴリズムの実際的評価, 情報処理学会アルゴリズム研究会30-7、92年11月.

[2]CARSSON AND CHEN：On partitions and presortedness of sequences. Acta Inf.29,267-280,1992.

[3] ESTIVILL-CASTRO AND WOOD：A survey of adaptive sorting algorithms,ACM Computing Surveys,Vol.24,No.4,December,1992,441-476.

[4]二村,筧,二村：対称ヒープの実現とその応用,情報処理学会アルゴリズム研究会28-7、92年7月.

[5]二村,二村,筧：対称整列法,情報処理学会アルゴリズム研究会28-8、92年7月

[6]二村,二村,遠藤,平井：葉数最適LOAS:葉数に関して最適な適応整列法,日本ソフトウェア科学会,94年11月.

[7]二村，青木，大谷，白井：整列法評価のためのランダム順列生成法，情報処理学会アルゴリズム研究会44-2，95年3月.

[8]浜村,若林,宮尾,吉田：入力列のプリソーテッドネスを考慮した最適並列ソーティングアルゴリズム,COMP-90-37.

[9]HOARE：Quicksort Compt.J.,1,1,1962,10-15.

[10]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.

[11]KNUTH：The Art of Computer Programming. Vol.3. Addison-Wesley,Reading,Mass,1973.

[12]MANNILA： Measures of presortedness and optimal sorting algorithms. IEEE Trans.Compt.C-34,1985,318-325.

[13]MOFFAT AND PETER SSON： Historical searching and sorting,2nd AISA,1991.

[14]PUGH：Skip lists:A probabilistic alternative to balanced trees, Commun. ACM,33,1990,668-676.

[15]SKIENA：Encroaching lists as a measure of presortedness,BIT 28,1992,755-784.

[16]SPRUGNORI：Properties of binary trees related to position,Comput. J.,Vol.35,No.4,1992, 395-404.

[17]VUILLEMIN： A unifying look at data structures,CACM,Vol.23,No.4 (April),1980.

付録1　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．ただ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で取る．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

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

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，

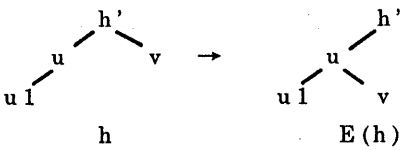
avel(x,1) = avel(x,2) = 1 より，avel(x,n)
= 1+(n-2)p(x) = (n+1)/3-(n-2)/(3x²)　(QED).

付録 2:対称整列法(対称ソート)における抜出し操作

与えられた数列に対する(減少)対称ヒープでは，最大値がヒープの根にあるので，ヒープの性質を保ちながら(対称性は保つ必要はない)根から次々と要素を取り出せば，数列をソートすることができる．この時に，ヒープから根を除去する操作(いわゆる DELETE-MAX)を抜出し操作と呼ぶ．この操作の基本は[5]に示したように，根の子供 2 名を比べ，大きいほうを次の根にし，小さい方をその根の一方の子供とするものである．この方法の欠点は，全体の中でも小さな値が一旦根につくと，いつまでも根の子としてい続け，そのために比較の結果である大小関係を構造として残すというヒープの特徴が利用できない点である．何故ならば，小さい値と比べられた最大値はヒープから除去されてしまうからである．ここでは，我々の実験でより高速で対称ソートを行える抜出し法について述べる．以下では根を h，その子供を u と v (ただし u の方が v よりも大きいとする)，そして u の子供を u1 と u2 とする．そして各木の根の値を h',u',v',u1',u2'のように木の名前の右肩にダッシュを付けて表す．ここでの方法は，小さい方の子供 v および大きい方の子供 u が持つ 2 名の子供 u1 と u2 の 3 者の最大の者を u の唯一の子供とする操作 SM(u,v)に基づく．SM は 3 者の比較結果を構造としてヒープに残しておく．抜出し操作を E(h)で表すと，その概略は次の通りである．

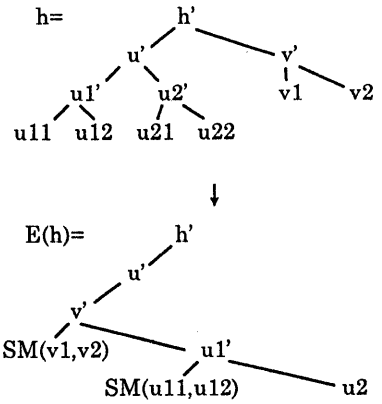
(1)h が葉又は子供を一人しか持たなければそのものを返す．

(2) h が子供を二人持つ時，その大きい方の子 u が子を一人しか持たなければ，小さい方の子 v を u の子供としたヒープを返す(下図参照)．



(3)その他の場合，E は下図のような変換をヒープに施す．

例えば u'≧v'≧u1'≧u2'のとき E は下図のような変換をヒープに施す．



また例えば u'≧u1'≧v'≧u2'のとき E(h)は下図のようになる．