アルゴリズム 28－8
（1992．7．18）

対称整列法：
攪乱要因dに対しO(n*log(d))時間の整列法

二村良彦　二村夏彦　筧捷彦
早稲田大学　理工学部　情報学科

実際的に高速な整列法を提案する．まず数列がどの程度非整列であるかを明確にするために数列の攪乱要因dと呼ぶ新しい概念を導入する．それは「両隣の要素よりも大きい要素（攪乱要素）の個数」である．次に$0 \leq d \leq (n-1)/2$かつ長さnの全数列についてのdの平均値は$(n-2)/3$であることを示し，対称ヒープを利用した新しい整列法（対称ソート）が与えられた数列の長さnのみならずその攪乱要因dにも明白に依存する平均O(n*log(d))かつ最悪O(n*d)時間の整列法であることを証明する（ただしd=0およびd=1のときは例外的に最悪O(n)時間で整列できる）．最後にQUICKソートおよびリストマージソートとの比較実験および計算量の理論的検討結果を示す．

SYMMETRIC SORT:
AN O(n*log(d)) SORT WHERE d IS A DISORDERING FACTOR

Yoshihiko Futamura　Natuhiko Futamura　Katuhiko Kakehi
Department of Information and Computer Science
School of Science and Engineering, Waseda University
Ookubo 3-4-1, Sinjuku-ku, Tokyo 169, Japan

A new sorting algorithm called symmetric sort and running fast in practical uses is described. The disordering factor d of a sequence is introduced to evaluate the disorderedness of the sequence. The d is the number of elements which are greater than both their left and right elements. Then, $0 \leq d \leq (n-1)$ and the fact that the average value of d is $(n-2)/3$ are shown. It is proved that the symmetric sort runs in O(n*log(d)) average and O(n*d) worst case time on a sequence with length n and disordering factor $d>1$. For d=0 or d=1, the sort runs in O(n) worst case time. Finally, results of theoretical and experimental comparison between the performances of QUICK sort, list-merge sort and symmetric sort are presented.

1．はじめに

全計算機の使用時間中25%がデータの整列(並べ換え)に費やされていることが認識されてから久しい[4]．ワープロおよび住所録等の個人用データーベースが普及した近年においてはこの比率は増加することはあっても減少することはないと考えられる。それに対して整列法は，長さnの数列を$O(n*\log(n))$時間で整列するQUICKソート[2]，ヒープソート[7]，リストマージソート[4]等の出現以来その計算時間を本質的に変える実用的方法は提案されていない．しかし在来の$O(n*\log(n))$時間の整列法には，殆ど整列済みの数列に対しても乱数列に対するのと同程度またはそれ以上の時間を要するという重大な欠点がある．中でもQUICKソートは正順または逆順に近い数列に対して$O(n^2)$時間を要するという不安定さを持つ．

実用的には，整列の対象となるデータの大部分の要素は既に整列済み（正順）である[4]。例えば住所録に新たにデータを追加して，全体を氏名のアルファベット順に整列しなおす場合には，もともとあったデータ（数列の大部分）は整列済みのことが多い。従って殆ど整列済みの数列に対しては格段に早くしかも平均でも$O(n*\log(n))$より良い整列法に対する実用的ニーズは高い。

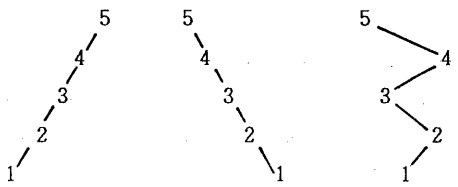
この問題を解決する目的で，本稿ではまず数列がどの程度非整列であるかを明確にするために数列の攪乱要因d（disordering factor）と呼ぶ新しい概念を導入する。それは「両隣の要素よりも大きい要素（攪乱要素）の個数」である。例えば数列が正順（即ち整列済み）または逆順（即ち逆に並べたものが整列済み）ならば$d=0$である。その他でも9,7,5,3,1,2,4,6,8のような数列の攪乱要因は0である。本稿では$0\leq d\leq(n-1)/2$かつ長さnの全数列についてのdの平均値は$(n-2)/3$であることを示す．次に対称ヒープ[2]を利用した整列法，即ち対称ソートを提案しそれが与えられた数列の長さnのみならずその攪乱要因dにも明白に依存する平均$O(n*\log(d))$かつ最悪$O(n*d)$時間の整列法であることを証明する（ただし$d=0$および$d=1$のとき，即ち正順，逆順あるいは9,10,7,5,3,1,2,4,6,8のような数列は例外的に最悪$O(n)$時間で整列できる）。

上記の議論から明らかなように対称ソートは数列の攪乱要因に依存して自動的に$O(n)$~$O(n*\log(n))$時間を取る。このような性質を持つ$O(n*\log(n))$時間の整列法は筆者等の調査の範囲においては他に例を見ない。例えば，挿入法（insertion sort)は正順の数列に対しては$O(n)$であるが逆順に対しては$O(n^2)$であり，かつ平均$O(n^2)$時間である。従って対称ソートとの比較の対象に成り得ない。

最後に平均的に高速であるとの評価の高いQUICKソートおよび最悪$O(n*\log(n))$と安定しているリストマージソートとの比較実験および計算量の理論的検討結果を示す。以下において使用する対数の底は2である，即ち$\log(n)=\log_2(n)$であるとする。

2．数列の攪乱要因と対称ヒープ

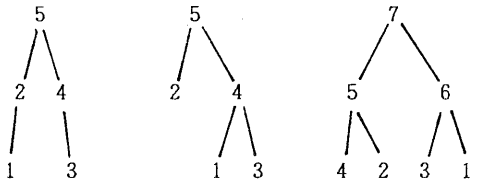
数列に対応する対称ヒープ（symmetric heap)[1]とは，対称（即ち中順）ツリーウォーク[3]によって元の数列を生成することのできる2進増加(または減少)木であり，元の数列より$O(n)$時間(最悪$2n-1$かつ最良$n-1$)で作ることができる(付録１定理１)。この対称ヒープの有する節，B節(枝を２本持つ節，即ちbinary node)，U節（枝を丁度１本持つ節，即ちunary node）および葉（枝を持たない節）の個数を各々n,d,uおよびmとする。この時dを攪乱要因と呼ぶ。即ち数列の攪乱要因とは対応する対称ヒープの持つB節の個数である。B節は攪乱要素（両隣より大きい要素）に対応する。従って数列の攪乱要因は攪乱要素数と一致する。正順，逆順およびその他の攪乱要因0の数列と対応する対称ヒープを図1に示した。



(1 2 3 4 5)　　(5 4 3 2 1)　　(5 3 1 2 4)
の対称ヒープ　　の対称ヒープ　　の対称ヒープ

図1：攪乱要因0の数列に対する対称ヒープ

攪乱要素を1,2および3個持つ数列および対応する対称ヒープの例を図２に示した。



(1 2 5 4 3)　　(2 5 1 4 3)　　(4 5 2 7 3 6 1)
の対称ヒープ　　の対称ヒープ　　の対称ヒープ

図2：攪乱要因1,2および3の数列に対する対称ヒープ

n,d,uおよびmの間には以下の関係が成立する。

(1) n=d+u+m　　（明らか）
(2) d=m-1　　（nに関する数学的帰納法より）
(3) u=n-2m+1　　（上記(1)と(2)より）
(4) m≦[(n+1)/2]　（0≦u≦n-1および(3)より）
(5) 任意のnに対して，mの平均値は(n+1)/3
　　（付録の定理２参照）
(6) 0≦2d≦n-1　　（上記(2)および(4)より）
(7) 任意のnに対して，dの平均値は(n-2)/3
　　（上記(2)および(5)より）

３．対称ソート

対称ヒープの根（root）には最大値があるので，ヒープソートと同様に次の様にすれば与えられた数列を整列することができる。

(1) 与えられた数列を対称ヒープにする。
(2) ヒープの根から最大値を取り出し，所定の場所に置く。
(3) 残りの木構造をヒープにする。ヒープが空になるまで(2)と(3)を繰り返す。

ここでは最大値を取り出すことだけが問題であるから上記(3)においてヒープの対称性（即ち中順ツリーウォークにより元の数列が生成できること）を壊しても構わないことに注意されたい。

これ等の操作の詳細を説明するために以下の用語を用いる。

Ｂヒープ：根がＢ節であるヒープ
Ｕヒープ：根がＵ節であるヒープ
勝ちヒープ：Ｂ節の大きい方の子供

上記(1)を実現するO(n)時間アルゴリズムは文献[2]に与えられている。Ｕヒープまたは葉に対しては(2)および(3)は常数時間で実現できることは明らかである。Ｂヒープに対しては(3)において図３の抜き出し(extraction)操作（Ｅ操作）を行い，Ｕヒープに変換する。即ち(3)から(2)に戻る際にはヒープは常にＵヒープになるようにする。Ｅ操作の要点は，根の勝ちヒープが葉またはＵヒープになるまで再帰的にＢ節の子供の比較を繰り返すことである。Ｅ操作の例を図４に示した。

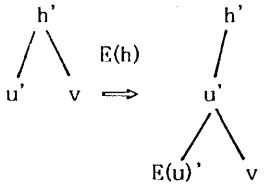


図3：Ｂヒープhに対する抜きだし操作（Ｅ操作）
v'≦u'のときuに再帰的にＥ操作を施し，その結果（Ｕヒープ）の右の子供としてvを持たせる。uがＢヒープでなければE(u)=u。u'<v'ならばuとvを入れ替えた後，同上の操作をする。但しヒープh,u,vおよびE(u)の根の値を各々h',u',v'およびE(u)'で表す。

上記の整列法においてキー，即ち数値の比較が行われるのはＥ操作においてのみである。またＥ操作が行われるのはＢ節に対してのみであるので，数値の比較回数を評価するためにはヒープの節数が２になるまで(2)と(3)を繰り返せばよい（節数２以下のヒープはＢ節を持ち得ないからである）。

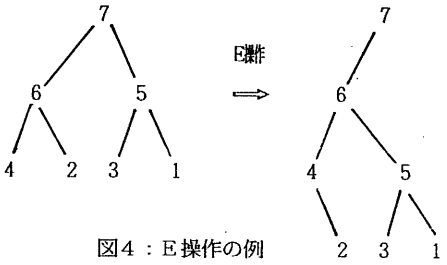


図４：Ｅ操作の例

上記の整列法において要素数nかつ攪乱要因dの数列に対する数値の比較回数は$O(n*[\log(d)])$であることを以下に示す（[]はガウスの記号を意味する）。但し次の２点を仮定する。

H1：節数tかつ葉数sのＢヒープに対するＥ操作において，勝ちヒープの節数は１とt-2の間を等確率で取る。

H2：１回のＥ操作によりＢ節（攪乱要素）は平均d/(n-2)個減少する。

Ｅ操作をn-2回行うことによりd個のＢ節を取り除くことができ，従ってその後はＥ操作が不要になることがH2の根拠である。この妥当性を実験的に示したグラフが図５である。そこでは長さ1000の乱数列に対応する対称ヒープから出発し，最大値を次々に除去した後に残っている節数（横軸）と攪乱要因（縦軸）との関係を示している。

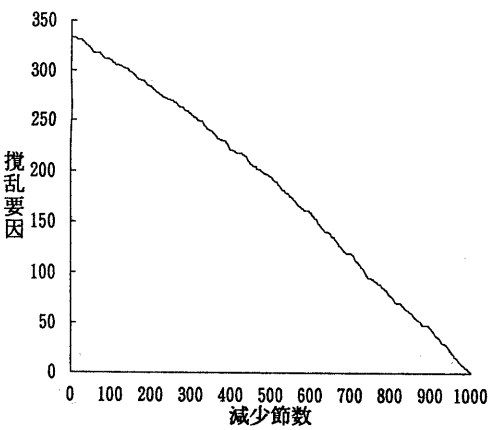


図5:攪乱要因と節数の減少関係（n=1000）

与えられたＢヒープが持つ節および葉の個数（Ｂ節の個数＋１）を各々tおよびsとする（$s \leq (t+1)/2$）。この時与えられたＢヒープに対するＥ操作の平均計算量（数値の比較回数）をC(s,t)で表す。これは上記(3)に関する議論に基づき次のように定義できる。

$C(2,t)=1$

$C(s,t)=1+(1/(t-2))*S(s-1,t-2)$　（$s>2$かつ$t>3$の時）

但し

$S(1,n)=0$

$$S(m,n)=\sum_{t=1}^{n}(1/\alpha(m,t))*\sum_{s=1}^{\alpha(m,t)}C(s,t)$$

但し$\alpha(m,t)=\min([(t+1)]/2,m)$。従って

$$S(m,n)=2*\sum_{t=3}^{m-1}(1/t)*\sum_{s=2}^{t}C(s,t)$$

$$+\sum_{t=2m-1}^{n}(1/m)*\sum_{s=2}^{m}C(s,t)\quad(2\leq m\text{かつ}2m-1\leq n\text{の時})$$

この時下記の定理３が成立する（証明は付録参照）。

定理３：$\sum_{s=2}^{m}S(s,t)\leq t*\sum_{s=2}^{m}[\log(s)]$

前述の仮定H2と定理３より下記の定理４が得られる（証明は付録参照）。

定理４：要素数nかつ攪乱要因dの数列に対する対称ソートの平均計算量は$O(n*[\log(d)])$である。

定理５：要素数nかつ攪乱要因dの数列に対する対称ソートの最悪計算量は$O(n*d)$である。

証明：攪乱要因がdであれば数列に対応するヒープの含むＢ節の個数もdである。従ってＥ操作はd回以内の比較後，必ずＵ節または葉に至り終了する。これより$C(s,t)\leq d$。従ってn-2回のＥ操作には高々$d(n-2)$回の比較が必要である(QED)。

例えば次のような数列95,96,5,97,98,3,99,2,100,1 (n=11,d=5)に対して比較回数は$O(n*d)=25$回となる。

定理４により計算量としては$O(n*\log(d))$であることは証明できたが，その上限を示すより精密な式はまだ求まっていない。しかし筆者等はそれを$n*\log(2d)+2n$，即ち$n*\log(8d)$以下であると予想している(図6)。

４．整列法の性能比較

配列ではなく，リスト構造として与えられた数列を小さい順に（即ち昇冪に）並べ変える場合について，QUICKソート（但しリストの先頭の要素を軸とする素朴な方式），リストマージソートおよび対称ソートの３者を比較する。一般に整列法の性能を評価する際の基準としては次の２つがある[4]。

(1) 数の大小関係の比較回数

(2) 数の入れ替え（即ち数列における場所の交換）回数

しかしここで取り上げた３つの整列法はリスト構造を利用するが故に上記(2)の入れ替えに要する時間は(1)と比べれば殆ど無視できるので，(1)だけを問題にする。200桁の十進数のような大きな数を整列する場合には，それ等の数の比較回数が少ないことが特に重要である。３者に関する比較回数の理論値を表１に示した。

表１：整列に要する計算量の比較（理論値）

	方式	QUICK	リストマージ	対称
計算量	平均	n*log(n)	n*log(n)	n*log(d)
	最悪	n^2	n*log(n)	n*d
	最良	n*log(n)	n*log(n)	n

比較実験のためのデーターは下記のような攪乱要因dを含む長さ2000の数列である（乱数は0と10000との間の数値）。

(1)d≒666(=2000/3)，d≒500,d≒250,d≒125,d≒62,d≒31,d≒15,d≒7

(2) 正順，逆順（d=0）

指定された個数の攪乱要因を含む数列を生成するために下記の２つの方法G1,G2を用いた。

G1:長さｎの乱数列の含む攪乱要因が約n/3である事実を利用する。即ち長さ3dの乱数列の後に長さn-3dの整列済みの数列を繋げる（例えばn=10,d=2のとき4,1,9,2,6,5,1,2,3,4)。

G2:長さn-dの攪乱要因0の数列に対して，両隣よりも大きな要素をd個等間隔に追加する（例えば下記のS1およびS2)。

これ等のうち比較的小さなdに対するG1は現実的整列問題に現われる典型である。しかしG1によっては，指定された個数の攪乱要素を正確に含むことができないことに注意されたい。

上記(1)で指定された各dに対してG1により生成された８つの数列の整列に要したキー（数値）の比較回数の平均およびn*log(8d)の値の比較を図6に示した。

またG2により生成された下記２種類の数列に対する同様の比較結果を図7に示した。

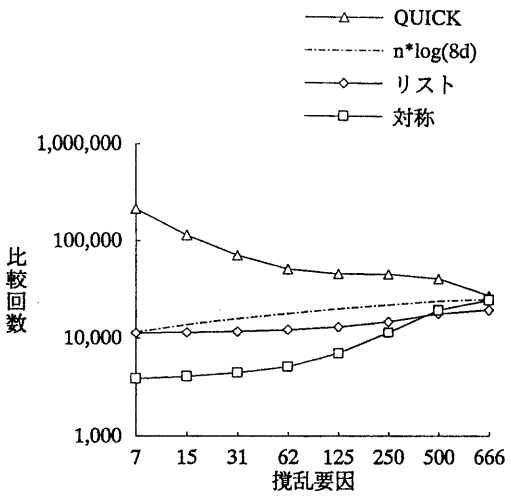


図6:規則G1により生成された8数列に対する比較結果

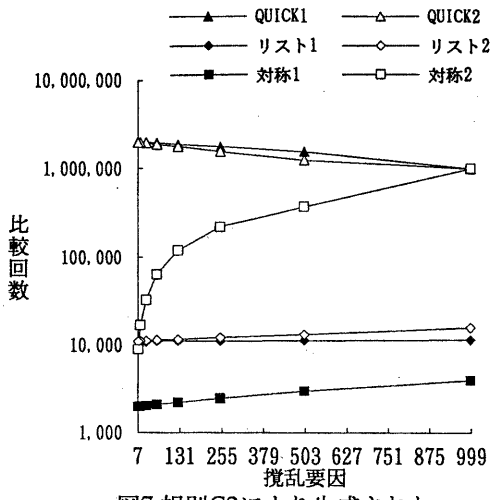


図7:規則G2により生成された数列S1,S2に対する比較結果

S1：5から5おきの上昇列に右隣+2の攪乱要素を等間隔でd個埋め込んだ長さ2000の数列。

例：n=10,d=3のとき5,10,17,15,20,27,25,30,37,35

S2：5から5おきの上昇列に-(右隣)の要素を等間隔でd個埋め込んだ長さ2000の数列。

例：n=11,d=5のとき-5,5,-10,10,-15,15,-20,20,
-25,25,30

対称ソートは対称ヒープを利用するが故に，他の２者よりも３倍ポインターメモリー（数値の所在場所の番地を格納するメモリー）を必要とする。因みに１ポインターメモリーは約４バイトであるので8nバイトのメモリーが余分に必要である。従って現実的にはメモリー管理の時間も考慮しなければならない。しかし図6及び7に対応する実験においては使用システム（muLISP86[5]）のメモリーに余裕があった故に，数値の比較回数が最小の方式の所要時間が最小であった。但しこの所要時間には比較回数のカウンターの更新等，テストデーター計測時間およびゴミ集め時間も含む。現時点では対称ソートのプログラムは非常に素朴であり，ほとんど最適化されていない。従って比較回数に関する改良の余地は多大である。

上記(2)の正順および逆順の数列に対してQUICKソートは1,999,000回，リストマージソートは11,085回そして対称ソートでは1,999回の比較が必要である。

攪乱要因が整列法の性能に与える影響が大きいことおよび素朴なQUICKソートが不安定であることが以上のデータより結論できる。また，攪乱要因が同じであっても攪乱要素の現れ方によっても整列法の性能が影響されることも図7より理解できる。

5．おわりに

攪乱要因dという数列の非整列度を表す概念を導入し，それが現在普及している実用的整列法の性能に影響を与えることを示した。また平均O(n*log(d))，最良O(n)かつ最悪O(n*d)時間の対称ソートを開発し，dが小さい(d≦n/8)ときには，既存の整列法よりも少ないキーの比較で整列が可能なことを示した。実際にデーターベース処理等で整列が行われる場合，dは小さいことが多い。従って，対称ソートは理論的のみならず実用的にも興味深いと期待される。

理論的には，計算量に関するより精細な上界を求めることが今後の課題である。実用的には下記の３点が当面の課題である。

(1) 本稿で述べた抜き出し操作(E操作)は，計算量の評価をできるだけ簡単にするための最も単純なものである。より込み入った抜き出し戦略を取ることにより比較回数を大幅に減らすことが可能である。現在いくつかの抜きだし戦略の比較評価を行っている。

(2) 最悪O(n*log(d))時間の対称ソートを開発し，その性能を評価中である。実用的にこれが最悪O(n*d)の対称ソートより優れているか否かということはちょうどヒープソートとQUICKソートのような関係にある。即ち与えられた数列の性質に依存しており，決定的な結論はまだ得られていない。

(3) 攪乱要因が小さい(d≦n/8)とき，200桁の整数あるいは長大な文字列等のように複雑なデータを整列する場合に対称ソートは特に有効である（対称ソートはキーの比較回数が少ないから）。また対称ヒープは分割統治法を適用し易い[1]という特性を有する。この２つの特徴を生かせる現実問題を調査中である。

6．参考文献

1) 二村夏彦，筧捷彦，二村良彦：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資料 1992年7月．

2) Hoare, C.A.R.: Quicksort, Computer J., Vol.1, No.1, 10-15, (1962).

3) Knuth, D.E.: The Art of Computer Programming, Vol.1 :Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, U.S.A., (1968).

4) Knuth, D.E.: The Art of Computer programming, Vol.3: Sorting and Searching, Addison-Wesley, Reading, Massachusetts, U.S.A., (1973).

5) muLISP-86 Reference Manual, LISP Language Programming Environment Soft Warehouse inc., (1986).

6) Stanley, R.T.: Enumerative Combinatorics,

Vol.1, Wadsworth & Books/Cole Advanced Books, Monterey, California, U.S.A., (1986).

7) Williams, J.W.J.: Algorithm 232: Heapsort, Comm. ACM, Vol.7, No.6, 347-348, (1964).

付録1　対称ヒープ

数列に対応する減少対称ヒープとは次の３つの性質を有する木構造である：

(1) 一つの節から出る枝の数は２本以下である。

(2) 親の値は子供の値以上である。

(3) 対称（即ち中順）ツリーウォーク[3]によって元の数列を生成することができる。

上記(1)および(2)は，所謂ヒープ（または優先木）の特質である[3,4]。性質(3)を有する故に本稿で用いるヒープを対称ヒープと呼ぶ。これは元の数列の全ての要素の位置関係がヒープの中で保存されることを意味する。但し対称ツリーウォーク(symmetric または in-order tree traversal)とは，木構造をその根から出発し縦優先（depth-first）に辿り，葉に到達したときまたは節に２度目に到達したときにその値を取り出す方法である。

例：数列を数のリスト（LISPのＳ式）で表現したとき数列(10 1 5 3 6 2 7 3 15 4 6)に対応する対称ヒープのＳ式は(15(10()(7(6(5(1()())(3()()))(2()()))(3()()))(6(4()())()))であり，その図表現は図8の通りである。

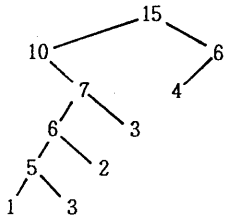


図8：数列(10 1 5 3 6 2 7 3 15 4 6)
に対する対称ヒープの図表現

定理１：与えられた長さnの数列から$O(n)$時間(最良n-1かつ最悪2n-1)で対応する対称ヒープを作成することができる(アルゴリズムと証明は文献[1])。

定理２：任意の自然数nに対し，節数nの対称ヒープは平均(n+1)/3枚の葉を持つ。

この定理を証明するために下記の３つの補題を用いる（補題の証明は省略）。

補題１：$L(n,m)$は節数nかつ葉数mの対称ヒープの総数を表すものとする（但し$1 \leq m \leq [(n+1)/2]$）。このとき

$L(0,m)=0$

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

補題２：長さnの数列に対応する対称ヒープの総数はn!である，即ち

$$\sum_{m=1}^{[(n+1)/2]} L(n,m)=n!$$

補題３：$ls(n)$は節数nの全てのヒープが持つ葉の総数を表すものとする。即ち

$$ls(n)=\sum_{m=1}^{[(n+1)/2]} m*L(n,m)$$

このとき$ls(n)=n!+(n-2)*ls(n-1)$。

定理２の証明：平均葉数を$fut(n)$と置く。補題２と３より，$fut(n)$=葉の総数/ヒープの総数=$ls(n)/n!$。

従って

$$\begin{aligned} fut(n) &= 1+((n-2)*ls(n-1))/n! \\ &= 1+((n-2)/n)*ls(n-1)/(n-1)! \\ &= 1+((n-2)/n)*fut(n-1) \\ &= 1+((n-2)/n)(1+((n-3)/(n-1))*fut(n-2)) \\ &= \ldots \\ &= (\sum_{i=1}^{n-1} i(i+1))/(n(n-1))=(n+1)/3 \quad \text{(QED)} \end{aligned}$$

付録2　対称ソートの計算量の評価

まず$S(m,n)$の第１項および第２項を各々$S1(m,n)$および$S2(m,n)$と置く。即ち

$$S1(m,n)=2*\sum_{t=3}^{m-1} (1/t)*\sum_{s=2}^{t} C(s,t)$$

$$S2(m,n)=\sum_{t=2m-1}^{n}(1/m)*\sum_{s=2}^{m}C(s,t)$$

補題４：$\sum_{s=2}^{m}[\log(m)]=m*([\log(m)]-1)+R(m)$

但し$R(m)=m+[\log(m)]+2-2^{[\log(m)]+1}$

（証明は文献[4]43頁問題42(b)参照）

補題５：$Q(s,t)=(t-2s)(s+3-2^{[\log(s)]+1})/(s+1)$とするとき，$S(s,t)\leqq t*[\log(s-1)]+Q(s-1,t)$ならば，

$$\sum_{s=2}^{m}S(s,t)\leqq t*\sum_{s=2}^{m}[\log(s)]$$

（証明は省略）

補題６：$3\leqq n$なるnに対して，$2\leqq m\leqq(n+1)/2$ならば

$S(m,n)\leqq n*[\log(m-1)]+Q(m-1,n)$

ただし，$Q(s,t)=(t-2s)(s+3-2^{[\log(s)]+1})/(s+1)$

証明：nに関する数学的帰納法で行う。

(1) Base: $n=3$の時，$m=2$。従って

$$左辺=S(2,n)=\sum_{t=3}^{n}(1/2)*\sum_{s=2}^{2}C(s,t)=(n-2)/2$$

$$右辺=Q(1,n)=(n-2)(4-2)/2=n-2$$

従って左辺≦右辺。

(2) Induction step:

$$S1(m,n)=2*\sum_{t=3}^{m-1}(1/t)*\sum_{s=2}^{t}C(s,t)$$

$$=2*\sum_{t=3}^{m-1}(1/t)*\sum_{s=2}^{t}(1+(1/(t-2))*S(s-1,t-2))$$

$$=2(m-3)+2*\sum_{t=3}^{m-1}(1/t)(1/(t-2))*\sum_{s=2}^{t}S(s-1,t-2)$$

$$=2(m-3)+2*\sum_{t=3}^{m-1}(1/t)(1/(t-2))*\sum_{s=2}^{t-1}S(s,t-2)$$

$$\leqq 2(m-3)+2*\sum_{t=3}^{m-1}(1/t)*\sum_{s=2}^{t-1}[\log(s)]$$

（帰納法の仮定と補題５より）

$$=2(m-3)+2\sum_{t=3}^{m-1}(1/t)((t-1)([\log(t-1)]-1)+R(t-1))$$

（補題４より）

$$=2(m-3)+2\sum_{t=3}^{m-1}((1-(1/t))([\log(t-1)]-1)+(R(t-1)/t))$$

$$=2(m-3)+2*\sum_{t=3}^{m-1}([\log(t-1)]-1+(R(t-1)/t))-2*\sum_{t=3}^{m-1}([\log(t-1)]-1)/t$$

$$=2*\sum_{t=2}^{m-2}[\log(t)]+2*\sum_{t=3}^{m-1}(R(t-1)-[\log(t-1)]+1)/t$$

$$=2*((m-2)([\log(m-2)]-1)+R(m-2))+2*\sum_{t=3}^{m-1}(R(t-1)-[\log(t-1)]+1)/t$$

$$=2(m-2)[\log(m-2)]-2(m-2)+R(m-2))+2*\sum_{t=3}^{m-1}(R(t-1)-[\log(t-1)]+1)/t$$

$$\leqq 2(m-2)[\log(m-2)]$$

$$S2(m,n)=\sum_{t=2m-1}^{n}(1/m)*\sum_{s=2}^{m}C(s,t)$$

$$=\sum_{t=2m-1}^{n}(1/m)*\sum_{s=2}^{m}(1+(1/t)*S(s-1,t-2))$$

$$=n-2m+2+\sum_{t=2m-1}^{n}(1/m)(1/t)*\sum_{s=2}^{m}S(s-1,t-2)$$

$$=n-2m+2+\sum_{t=2m-1}^{n}(1/m)(1/t)*\sum_{s=2}^{m-1}S(s,t-2)$$

$$=n-2m+2+\sum_{t=2m-1}^{n}(1/m)*\sum_{s=2}^{m-1}[\log(s)]$$

（帰納法の仮定と補題５より）

$$=(n-2m+2)(1+(1/m)*\sum_{s=2}^{m-1}[\log(s)])$$

$$=(n-2m+2)(1+(1/m)((m-1)([\log(m-1)]-1)+R(m-1)))$$

（補題４より）

$$=(n-2m+2)(1+((1-(1/m))([\log(m-1)]-1)+R(m-1)/m)$$

$$=(n-2m+2)([\log(m-1)]+(R(m-1)/m)$$
$$-([\log(m-1)]-1)/m)$$

$$=n*[\log(m-1)]+(n-2(m-1))(R(m-1)-[\log(m-1)]+1)$$
$$/m-2(m-1)*[\log(m-1)]$$

$$=n*[\log(m-1)]+Q(m-1,n)-2(m-1)*[\log(m-1)]$$

従って$S(m,n)=S1(m,n)+S2(m,n)$

$$\leqq 2(m-2)[\log(m-2)]+n*[\log(m-1)]$$
$$+Q(m-1,n)-2(m-1)*[\log(m-1)]$$

$$\leqq n*[\log(m-1)]+Q(m-1,n) \qquad \text{(QED)}$$

補題５と６より下記の定理３が得られる。

定理３： $\sum_{s=2}^{m} S(s,t)\leqq t*\sum_{s=2}^{m}[\log(s)]$

補題７： $p=d/(n-2)$かつ$k=1/p$と置く。この時kが自然数数ならば $\sum_{i=0}^{n-3} C([ip]+2,i+2)\leqq(n+k-2)*[\log(d)]+k$

証明： $\sum_{i=0}^{n-3} C([ip]+2,i+2)=\sum_{i=0}^{k-1}\sum_{j=2}^{d+1} C(j,(j-2)k+i+3)$

（kが自然数より）

$$\leqq\sum_{i=0}^{k-1}\sum_{j=2}^{d+1} C(j,n)=k\sum_{j=2}^{d+1} C(j,n)$$

$$=k(d)+(k/(n-2))\sum_{j=2}^{d+1} S(j-1,n-2)$$

$$=n-2+(k/(n-2))\sum_{j=2}^{d} S(j,n-2)$$

$$\leqq n-2+k\sum_{j=2}^{d}[\log(j)] \qquad \text{（定理３より）}$$

$$=n-2+k(d)*([\log(d)]-1)+k*R(d)$$

（補題４より）

$$\leqq(n-2)*[\log(d)]+k([\log(d)]+1)$$

$$=(n+k-2)*[\log(d)]+k \qquad \text{(QED)}$$

定理４： 要素数nかつ攪乱要因dの数列に対する対称整列法の平均計算量は$O(n*[\log(d)])$である

証明： 与えられた数列をヒープ化するための計算量定理１より$O(n)$であるので$k=(n-2)/d$が自然数ならば補題７より明らか。kが自然数でなければ，ある自然数$d'<d$に対して$k'=(n+d'-2)/d$が自然数となる。与えられた数列のどの要素よりも大きなd'個の要素を追した新しいヒープに補題７を適用すれば，それは$(n+k-2)*[\log(d)]+k'$以内で整列できる。従ってその計算時間は$O(n*[\log(d)])$である(QED)。