社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－AL－103 (10)
2005／11／11

区分木ソート

仁尾 都
明星大学経済学部
nio@mi.meisei-u.ac.jp

計算量がO（ｎｌｏｇｎ）で，作業メモリがΘ（ｎ）であり，かつ平均cpu時間がクイックソートと同程度のソートアルゴリズムを提案する．本ソートは逐次添加されるキーを平衡２分探索木の内点が所持する閾値によって定まるキーカバー範囲（区分）を持つ葉に割り当てる．葉に割り当てられたキーの個数が一定数より大きくなると，内点を新しく設け，その閾値を決定し，葉を２分し，２分木を平衡化する．葉のキーは挿入ソートとマージソートでソートされる．

Interval Tree Sort

Misato Nio
Faculty of Economics, Meisei University
nio@mi.meisei-u.ac.jp

Interval Tree Sort Algorithm, whose computing complexity O(nlogn) is superior, and memory size Θ(n) inferior, but average sort time nearly equal to quick sort, is proposed. Every key additively given to the algorithm is assigned to one of leaves of a balanced binary search tree. When the number of keys assigned to a leaf exceeds the fixed number, the leaf is divided into two so that the tree is rebalanced. Keys in a same leaf are sorted by insertion sort and merge sort.

１．はじめに

殆どの計算機アルゴリズムの教科書で最初に出てくるテーマはキー群をその値の大小に従って整列するソートアルゴリズムである．かようにソートは最もポピュラーなアルゴリズム研究分野となっており，その研究開発の歴史は長い．ソートは多くの研究分野だけでなく，会計処理や統計処理の事務処理分野などの日常的業務にも繰り返し活用されており，常により高速なソート技法が望まれている．現在平均的な処理能力が最も高いと一般に認められているソートアルゴリズムはクイックソートである．しかしながら欠点としてその性能がキーの入力順序に大きく依存し，最悪計算量はO（n^2），スタック領域もO（ｎ）となることに起因する不都合な事態が発生するなど，安心した利用ができない．したがって，クイックソートより平均処理速度が高速で，しかも計算量がO（ｎｌｏｇｎ）で，作業メモリも小さく，かつ常に安心して用いることのできるソートアルゴリズムが求められている．本報告では，作業メモリを最大（２ｎ＋６ｎ／ｋ）だけ必要（ｋはｌｏｇｎ程度）

とするものの，計算複雑度がO（ｎｌｏｇｎ）であり，かつ平均 cpu 時間がクイックソートと同程度で，ソートを安全確実に実行できるアルゴリズムを提案する．

２．従来の研究

ソートアルゴリズムの研究分野には今までに非常に多くの努力とアイデアが注ぎ込まれてきた．その結果，多種多様のアルゴリズムが生まれた．しかしながら，他のどのアルゴリズムも，その平均 cpu 時間でクイックソートを陵駕できない．そのため，クイックソートの欠点であるO（n^2）のケースに対処するために，クイックソートの改良が試みられている．主な改良ポイントはピボット選択法の改良であるが，それだけではO（n^2）は改良されない．ピボット選択法として平均的に優れているとされる Median-of-3 も universal な分布に対してO（n^2）となり，その出現頻度も低くないことを指摘した D.R.Musser はクイックソートとヒープソートを使い分ける Introsort〔１〕を提案し，その性能を最悪でもヒープソート並みに抑えることができるとしている．また，平均処理速度がクイックソートよりも優れているとする河村らの多重分割ソート〔２〕があるが，これも内部にはクイックソートが用いられている．多重分割ソートはバケットソートに分類できる．ソートの前処理段階でｎに応じてバケットの数を決定し，バケットの間の閾値をキーからランダムサンプリングすることにより，バケットに振り分けられるキーの平均個数を抑制することを狙ったものであり，平均処理性能をクイックソートより向上させることに成功している．しかしながら，ひとつのバケットに残りのキーが入ってしまうケースがあり，それらのキーをクイックソートでソートしているので依然としてO（n^2）となる．このように，クイックソートを凌駕するソートアルゴリズムの開発もまたクイックソートをベースとすることが行われてきた．

クイックソートをベースにしていないソート法として今回提案する区分木ソートは構造化された挿入ソートであると見做すことができる．キーをひとつずつ取り出してはソート済みのキー列Ａに挿入ソートで挿入を行い，Ａを徐々に長くし長大なキー列を作成する．しかるに挿入ソートはO（n^2）であり，そのままでは使えない．そこでＡを容量が一定ｋの複数区分に分ける．これによって局所的に挿入ソートはO（k^2）に抑制できる．区分を平衡２分探索木の葉に対応させて管理することによって，各キーがどの区分に属するかの処理はO（ｌｏｇ（ｎ／ｋ））になる．葉内のキー数が多くなると葉を２分する．最後に２分木を辿ってＡを取り出す．

すでに２分木ソート（Binary tree sort）と呼ばれるソートアルゴリズムが公知である．このアルゴリズムは逐次追加的に与えられたキーを２分探索木に挿入し，最後に２分木を辿ってキーを整列させる．本来，２分木ソートはO（n^2）であるが，キー毎に木を常に平衡に保つ処理を追加すればO（ｎｌｏｇｎ）となる．しかしながら，この処理の負担が大きすぎると考えられてきており，ヒープソートやマージソートなどの他のO（ｎｌｏｇｎ）アルゴリズムと同列に比較されることが殆どなかった．２分木ソートではひとつのキーを

ひとつの内点や葉には一つのキーだけしか保持させないため，木が深くなる．そのため木の平衡化処理の回数がキーの数だけ必要となり，負荷が大きくなる．区分木ソートでは互いに近くの値を持ったキーをひとつの区分として捕らえ，区分単位で葉に記憶し，また区分内のキーの数が区分の容量２ｋ以上になったときに葉を２分し，このときにのみ木の平衡化処理をすることとした．これにより，２分木の深さは浅くなり，平衡化処理回数は削減できる．

区分木ソートを Library sort〔３〕と対比できる．この手法では，挿入ソート列に所々に隙間を持たせ，新しく挿入するキーの格納スペースとする．挿入位置の決定方法は２分法による．これにより，高い確率的でＯ（ｎｌｏｇｎ）を実現する．この点，区分木ソートは葉に固定長２ｋのスペースを設けており，同様に隙間は常に確保される．

区分木ソートの葉をバケットとして解釈すると，上記に述べた多重分割ソート法と比較することができる．多重分割ソート法では，ソート開始時にバケットの数とそれらの閾値を決定し，ソートが完了するまで固定する．区分木ソートではあくまでキーは逐次添加式で受け取り，区分の細分化は最後まで続けるので区分の閾値は常に変動する．

従来から２分木ソートで木を動的に平衡化し，またはバケット法でバケットを動的に平衡的に細分化することは大きな処理負荷になると考えられてきた〔４〕．平衡化の処理負荷を軽減する工夫を施した区分木ソートは，最悪計算量がＯ（ｎｌｏｇｎ）であることと，平均 cpu 性能がクイックソートと同等であるという２つの長所を得た．

２．区分木ソートアルゴリズム

（１）区分木のデータ構造

区分木は平衡２分探索木であり，入力キーの領域とは別の作業メモリの中に作成される．木の根と各内点には２分探索木と同様，閾値を持たせ，これにより葉の担当区分範囲を定める．したがって，内点ｐの左側に位置するすべての葉の担当区分範囲の最大値はｐの所持する閾値よりも小さい．すべての葉は固定の長さ２ｋ＋１のバッファを持っており，更にそれは長さｋの前半バッファと長さｋの後半バッファに分かれている．ソート途中の定常状態では，すべての葉の前半バッファには常にソート済みのｋ個のキーが満杯状態で格納されており，後半バッファには高々ｋ個のソート済みのキーが格納されている状態になっている．中間のｋ＋１番地には後半の格納キー数が記憶される．図１はｋ＝３とした時の昇順ソートのための区分木の状態例で，１１から４２まで連続する整数のキーが処理され，木の内点と葉に蓄えられた有様を示す．各葉の前半と後半のキーは最終段階でにマージされて出力される．キーが逐次添加されるにしたがって次第に葉のキーをカバーする範囲は細分化され，葉の数は増え，区分木の深さも深くなる．例えば１９の閾値をもった内点の右の葉にもう１個のキーが追加されると葉は分割される．

（２）区分木ソートのアルゴリズム

ｋｅｙ（ｉ）（１≦ｉ≦ｎ）を昇順ソートするとする．葉の容量２ｋ＋１のｋは外部指定

である.

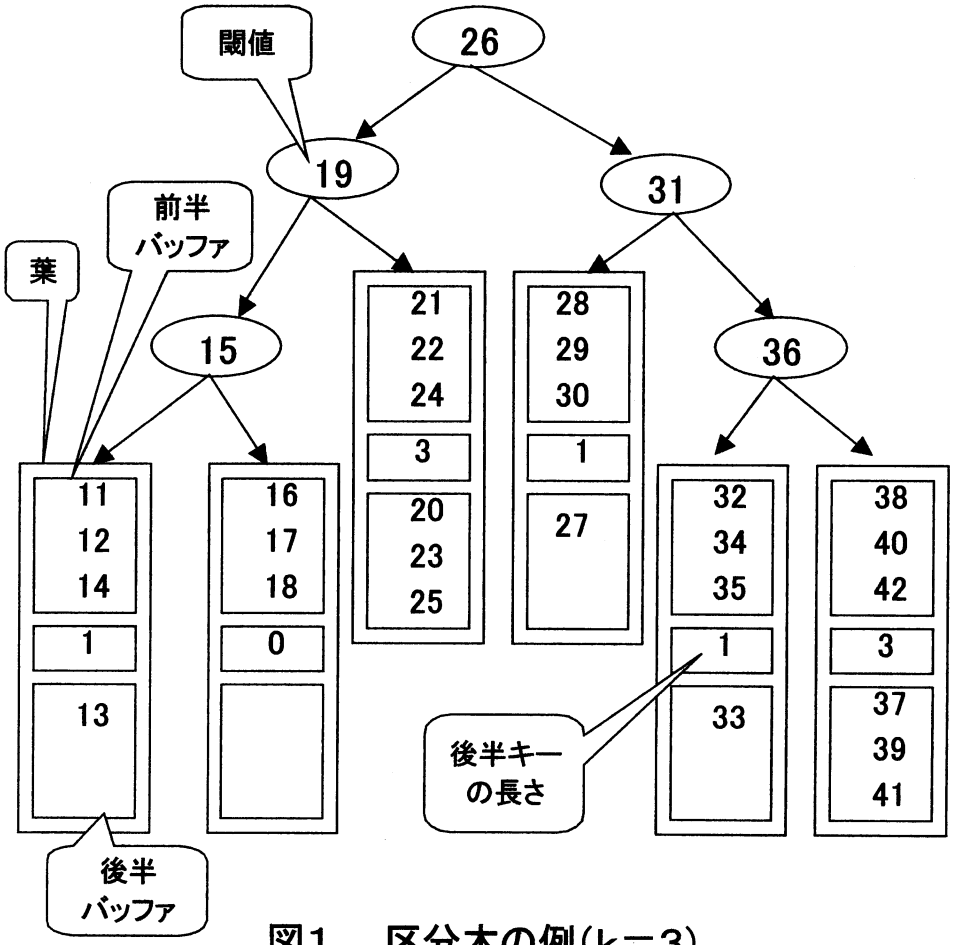

図1. 区分木の例(k＝3)

〔手順１〕最初の２ｋ＋１個のキーを何らかの方法（例：挿入ソート）でソートする．もしｎ＞２ｋ＋１なら，根を作成し，前半ｋ個を左の葉の前半に，ｋ＋１番目を根の閾値にし，最後のｋ個を右の葉の前半に格納する． 他の場合は終了.

〔手順２〕for i=２ｋ＋２ to n

〔手順２．１〕２分探索木を辿りキー（ｋｅｙ（ｉ））が所属する葉（ｉ）を決定.

〔手順２．２〕

if（葉（ｉ）の後半の格納キー数＜ｋ）（〔手順ｉ〕キーを後半に挿入ソート）

elseif（葉（ｉ）の後半の格納キー数＝ｋ） ‘葉（ｉ）の後半が満杯のとき’

（〔手順ii〕内点ｔとひとつの葉ａを新しく作成する．キーを葉（ｉ）の後半に挿入ソートし，結果を葉（ｉ）の前半とマージし，その結果の前半ｋ個を葉（ｉ）の前半に，ｋ＋１番目をｔの閾値にし，最後のｋ個

をａの前半に格納する．ｔを葉（ｉ）の親の子とし，ｔの左の葉を葉（ｉ）とし，ｔの右の葉をａとする．

〔手順iii〕２分木を平衡化する）

〔手順３〕２分探索木に沿って順次葉のキーと内点の閾値を取り出し，ｋｅｙ（ｉ）の格納エリアにソート結果を上書きする．この際，各葉の前半のキーと後半のキーはマージして出力する．

３．計算複雑度

以下，ひとつのキー当たりの最悪計算量とその繰り返し回数を述べる．

〔手順１〕＝Ｏ（（２ｋ＋１）^{2}）．理由：２ｋ＋１個のソートは挿入ソートと仮定．

———繰り返し回数は全部で１回．

〔手順２．１〕＝Ｏ（ｌｏｇ（ｎ／ｋ））　理由：葉の数はＯ（ｎ／ｋ）．平衡化は２色木を採用し，その深さはＯ（２ｌｏｇ（ｎ／ｋ））．———全部でｎ－２ｋ－１回．

〔手順ｉ〕＝Ｏ（ｋ）．———全部で（ｎ－２ｋ－１－（（ｎ－２ｋ－１）／ｋ））回．

〔手順ii〕＝Ｏ（ｋ）．　理由：後半はｋ個のキーに対し１個のキーを挿入ソート（Ｏ（ｋ））．その結果を前半とマージソートＯ（２ｋ＋１）．その結果を葉（ｉ）の前半とａの前半に格納（Ｏ（２ｋ＋１））．

———全部で（ｎ－２ｋ－１）／ｋ回．

〔手順iii〕＝Ｏ（ｌｏｇ（ｎ／ｋ））　理由：木は２色木で平衡化すると仮定．

〔手順３〕＝Ｏ（（ｎ／ｋ）＋ｎ）　理由：すべての内点（Ｏ（ｎ／ｋ）個）は３回スキャンされる．各葉はその後半がすべて空である場合が最悪なので，手順３での前半と後半のバッファ間のマージのための計算量は考慮する必要はない．すべての葉のＯ（ｎ）のキーが出力バッファに移動する．———全部で１回．

したがってすべての合計のオーダーはＯ（ｎｌｏｇｎ＋ｎｋ）となり，ｋの値をｌｏｇｎと同程度にすることにより，Ｏ（ｎｌｏｇｎ）とすることができる．

作業メモリは，１つの内点のために長さ５の管理メモリが必要である（Ｏ（５ｎ／ｋ））．またすべての葉のバッファの合計がＯ（２ｎ＋ｎ／ｋ）であるので，作業メモリはＯ（２ｎ＋６ｎ／ｋ）となる．

４．計算機実験

区分木ソートの性能評価のために，区分木ソート，クイックソート及びヒープソートを計算機に実装した．区分木ソートのインターフェースを **its(key,n)**とし，出力先も **key** とし，作業メモリはルーチン **its** の中でｎに応じて動的に確保する．クイックソートは **quicksort(key,from,to)**とし，コーディングを付録に掲載する．ピボットは３つのキーの平均とし，再帰実行時にソート対象数が１２個以下になった場合は挿入ソートを行う．１２

とした理由はその場合の処理速度が最も高速であったためである．ヒープソートはquicksort(key,n)としたが，そのコーディングは紙面数の都合により掲載できない．共に，使用した言語はＥＸＣＥＬＶＢＡで，計算機のｃｐｕは PentiumⅣ（２ＧＨｚ）である．表１に各種条件下での，区分木ソートの cpu 時間の対クイックソート比と対ヒープソート比を掲載した．ただし，ｋ＝1.2とした．従って比が１より小さい場合，２分木ソートのほうが優れていることを示す．キーはすべて実数である．一様乱数（０＜乱数＜１）はＶＢＡの乱数発生関数Ｒｎｄを用いた．標準偏差が与えられた正規分布の乱数はボックスミューラー法で作成した．

区分木ソートにとって都合の最も悪いケースは，キー出現順序が殆ど降順か昇順である場合である．なぜなら，そのような時は最近に作られた葉にキーが連続的に配分され，その葉が更に分割され，木の平衡化作業が連続的に繰り返され，他の葉の後半は空白のまま残されてしまうからである．またヒープソートにとってもキーが昇順や降順の場合はやはり最悪のケースになる．２分木に対しキーを追加し，あるいは削除する場合，常に根から葉に至るまで，すなわち木の深さと同じ回数だけ木を上下するという探索処理を強いられるためである．そこで，昇順と降順のケースを取り上げた．昇順のキーはまず指定された分布のキーを乱数で作成し，次にそれをソートすることによって得た．降順のキーも同様である．

表１．区分木ソートの cpu 時間比（対クイックソート，対ヒープソート）(N(m,σ):正規分布)

n	キーの分布	一様分布	$N(0,10^{-3})$	$N(0,10^{3})$	$N(0,10^{-3})$
	キーの入力順序	ランダム	ランダム	昇順	降順
10^3	対クイックソート比	0.95	0.98	1.73	1.75
	対ヒープソート比	0.49	0.52	0.36	0.56
10^4	対クイックソート比	0.90	0.90	1.80	1.82
	対ヒープソート比	0.47	0.46	0.34	0.55
10^5	対クイックソート比	0.85	0.85	1.85	1.86
	対ヒープソート比	0.43	0.43	0.33	0.55
10^6	対クイックソート比	0.86	0.85	1.91	1.91
	対ヒープソート比	0.41	0.41	0.32	0.54

５．結果の検討

表１を概観すると，キーの入力順序がランダムの場合，区分木ソートの計算機による実際の平均 cpu 時間はクイックソートより短く，また，区分木ソートはヒープソートに比べ０．５６以下となっている．ｎが増加するにしたがってその比が向上する理由は，ｎが小さい間はクイックソートの再帰的 Call の回数が少なくその処理負荷の割合が軽いためであろう．

キー出現順序が降順か昇順であるような区分木ソートにとって最悪のケースでは，クイックソート比で最大１．９倍程度となりかなり劣る．しかし表には記載していないが，区分木ソートでは昇順や降順のキーの場合と一様分布の場合の cpu 時間の比は１．２４倍に収まっており，本手法がＯ（ｎｌｏｇｎ）であることの利点が現れている．クイックソート比では最大１．９倍にもなる理由は，クイックソートで採用したピボットを最初と中間と最後のキーの平均する方法（Median-of-3）を採用したことにより，昇順・降順キーに対する性能はランダム順のキーの場合に比べ，逆に大幅に性能が向上しているためである．しかしながらピボットの決定方法が如何に定められようとも，その決定方法にもっとも都合の悪い入力データが存在する．Median-of-3法に対してＯ(n^2)となるようなMedian-of-3 killer〔１〕のケースを今回は取り上げていない．

平均cpu時間の実験結果は使用言語や計算機環境，双方のコーディング方法に依存し，特に今回クイックソートの再帰的 call の負荷軽減対策として挿入ソートを併用するだけでは不十分の可能性があり，区分木ソートの平均 cpu 時間がクイックソートより優れているとは断定できない．この程度の差であれば区分木ソートの平均 cpu 時間は現時点ではクイックソートと同程度であるとせざるを得ない．実験結果の検証を補強するために，区分木ソートとヒープソートの cpu 時間比を表に掲載した．表１から，クイックソートのヒープソートに対する cpu 時間比が約１．９から５．９になることが読み取れる．D.R.Musser はクイックソートのヒープソートに対する cpu 時間比が２から５になると指摘している〔１〕．表から読み取れるように，ヒープソートのクイックソートに対するcpu時間比は１．９から５．９の範囲に入っており，本実験に用いたクイックソートやヒープソートの計算機実装方法，ならびに本実験結果はある程度の妥当性があると考えている．

実験により，ｋは約１２とした時に cpu 時間がもっとも少なかったが，その近辺のｋに対しても性能の大きな劣化は見られなかった．

機能的な観点から見ると，バケット法や多重分割ソート，さらにはクイックソートなど多くのソートアルゴリズムがソート開始時点ですべてのキーが与えられていなくてはならないのに対し，区分木ソートではキーを逐次添加することができるという点と，stable であるという点を長所として持っている．この点はヒープソートも同様である．

５．終わりに

作業メモリをΘ（ｎ）だけ必要とはするものの，cpu 時間がＯ（ｎｌｏｇｎ）である２分木ソートを提案した．ソートの開始時期には２つの葉だけを持つ平衡２分木であったものが，キーが次々と添加されるにしたがって葉が２分割されて大規模な平衡２分探索木に生長するという特長を持つ．計算機実験により，平均 cpu 時間がクイックソートと匹敵することを確認した．

〔1〕D.R.Musser, Introspective Sorting and Selection Algorithm, Software: Practice and Experiences,27(8),pp.987-pp.993,August,1997
〔2〕川村畑行，江口賢和，小笠原基泰，重村哲史，多重分割ソートアルゴリズム，情報処理学会，Ｖｏ．１３５，Ｎｏ．１２，１９９４
〔３〕M.A.Bender, M.Farach-Colton, M.Mosteiro, Insertion Sort is O(nlogn), http://xxx.lanl.gov/PS_cache/cs/pdf/0407/0407003.pdf
〔4〕S.Basse，アルゴリズム入門－設計と解析－，星雲社，１９９８

〔付録〕比較に用いたクイックソートのアルゴリズム

```
    Sub quicksort(key, a, z)
    If  z - a <= 12  Then
      For  i = a + 1 To  z
        w=key(i)
        For j=i-1 to a step -1
           If key(j)>w then (key(j+1)=key(j)
                       else (key(j+1)=w: goto nexti )
        next j
        key(a)=w
nexti:   next  i
    Else
        pivot = (key(a) + key((a + z) / 2) + key(z)) / 3
        left = a : right = z
        Do While left <= right
            Do While key(left) < pivot     (left = left + 1)
            Do While key(right) > pivot     (right = right - 1)
            If  left <= right then   (w = key(left): key(left) = key(right): key(right) = w:
                                      left = left + 1 : right = right - 1)
        Call quicksort (key, a, right)
        Call quicksort (key, left, z)
    EndIf
```