ソート・アルゴリズムを思いついた

ソートのアルゴリズムを思いついてしまった。ネット等で確認すれども全く同じ物は見つける事ができない。色々改良案を考える内に、これは現在最速といわれている物を上回るパフォーマンスになるのでわ、そうなると権威のある学術誌に投稿しても載ってしまい、コンピュータアルゴリズムの教科書を書き換える事も間違いないのである。そのつもりになって、少し真面目に研究してみたが、そうそう簡単ではない事に思い至った。
先人が苦労した物を小手先の思いつきでは越えられない。

最速を越えられないとなると仮にこれが新しいアルゴリズムであっても、真面目に研究・論文化しても全くその労力に見合わないであろう。そもそも私は研究者ではなく一介のアプリ制作者でしかないのだ。
得られるメリットとしても、Wikipediaに項目が追加されるかレベルだろうし、スリープソートのようなインパクトもないので作ってるアプリの宣伝にもならない。
また、基本は単純でアホみたいな物なので、多分誰か前に思いついてる方が可能性が高い、ネット上にあらゆるソート・アルゴリズムが網羅されていると考える方がおかしいのである。
みるみるやる気が削がれていく。

しかし新しい物である可能性は消えておらず、ネットのどこかに成果をまとめて揚げておけば、私がこの時点で思いついたよと唾をつけておく事は出来るであろうと思い、まとめたのが以下の文章である。
もう一つひらめきがあればなんとかなるのではという気はまだしており、これを最終結論として放棄は現時点でまだしていませんが。

下記にソートプログラム説明を示すが、まず私はこれをリージョントップ・ソートと名付けた。当初チャンピオン・ソートとしたがあまりにも名前負けしているのでやめ、よりソートの特徴を示しているこの名前にした。
基本版1つ、及び3つの改良版ソートアルゴリズム及びプログラムを作成している。

1. リージョントップ・ソート (基本版)

基本的なソートの方法はトーナメント戦をしてチャンピオンを選ぶという物で、これは従来ソート・アルゴリズムにもある。ただし、トーナメント戦をする従来ソートは元データ以外にトーナメント結果の記憶領域を必要とするが、これは元々のデータ自体にトーナメントの結果を記録し、ソートされる配列以外の余計な記憶領域は基本は要しないというのが違う点である。(引き分けデータの並び順が入れ替わっても問題ない場合)

ソート種類の分類としては、手法は交換になる。また安定ソートではない。

次にアルゴリズムの説明をするが、以下の説明ではチャンピオンをデータ配列の添字から0位、次点を1位で統一する。(この数値が重要である為)

まずデータを横に並べこの上にトーナメント戦の木があると考える。
前から2個ずつの地域に分けられ、最下層である1回戦の木がある。その上に前から4個ずつの地域に分けられ、次の層である2回戦の木がある。最下層の2個の地域が2つ入ることになる。
同様に 8個ずつ, 16個ずつ .. 2^N 個ずつ (2^N < 個数まで) の階層化した地域に分けられ、決勝の木に至るまで続く。
下層から順番にトーナメント戦をする訳であるが、このソートの特徴点としてトーナメント戦でそこまで勝ってきたデータ、そこ以下のトーナメント木に含まれる領域でトップのデータを、その領域の先頭に入れるという処理を行う。上層地域の試合では、2つの下層地域の先頭のデータ同士を戦わせれば良いことになる。
ソート中は各領域の先頭にその領域のトップが入ってるという状態を作って保つ。
(保つにはのちに述べる操作が必要)
トーナメントが一通り終わり全体の先頭データはチャンピオンで最強である。これはトーナメントを卒業し、残りの中で最強を選び、残り部分での先頭に入れる。これを繰り返す。
残りがなくなって全部の試合が終了した時点でソート完了である。

チャンピオンより後の順位を決める戦では「地域のトップが先頭にある」のであるから、以降の未確定部分を最も大きな地域ずつに分けて各地域トップ同士の試合での一番勝者となる。
例えば0位決定後、1位の位置には1個のみ、以降2個の地域で先頭にトップ、4個の地域で先頭にトップ .. と並んでいるので、各地域トップのデータで決定戦をする。
配列添え字数値は 0 からであるが、この2進数を下位から見て最初に1が出てくる位置によって簡単にその位置をトップとする最大となる地域を知る事ができる。例えば 0b???1000 ならば以降 0b1000 (8)個内のトップである。奇数 0x?????1 ならば1個のみでのトップである。(トーナメント木を書いて前から2進数をふればすぐわかる。)

ここで地域のトップが先頭に入るというこのソートの条件を保つ処理について述べる。
前方地域と後方地域のトップが試合をして前方地域のトップが勝った場合、両地域を含む地域のトップが前方地域のトップであり、地域内全域の下層から上層まで「地域のトップが先頭に来る」関係は保たれたままであり、入れ替えもないので何もしなくともよい。
後方地域のトップが勝った場合データを入れ替えるが、前方地域ではトップがそれまでのトップに勝ったデータに入れ替わるだけであるから「地域のトップが先頭に来る」関係は保たれている事は明らかである。後方に降格したデータは後方地域のトップの位置にまず入る訳であるが、このデータは後方地域ではトップから実はボトムである可能性まであり後方地域内のトップ位置を含むトーナメント木は再度下層から上層まで試合を実施し「地域のトップが先頭にある」秩序を作り直す必要が有る。
またこの試合でまたデータの入れ替えが発生したら、再度後方に降格したデータをトップとする下層地域で同じ処理をする、つまり再帰的処理を行う必要がある。
データの入れ替えが無ければ無いほど「地域のトップが先頭にある」秩序を作り直す試合はないので、高速に終了する。

処理をもう少し具体的に述べたのが下記である。プログラムリストを見てもらった方が早いと思うが、理解補助の説明である。
1. データを前から順番に2個ずつの地区に分けていく。地区内で隣同士戦って勝った方を前に、負けた方を後ろにする。
2. データを前から順番に4個ずつの地区に分けていく。先の2個ずつの試合で勝った方が前なので、前側のデータを戦わせ、4個のなかでトップのデータを先頭に入れる。
ここで交換が起こらなければ、1. の関係も保たれたままである。交換があった場合、昇格データは 1. の関係も保たれたままであるが、降格データは壊れる。
よって降格データのみは再度 1. の隣同士戦わせ、 1. の関係を保つようにする。
3. データを前から順番に8個ずつの地区に分けていく。先の4個内の試合で最も勝ったデータが前なので、前側のデータを戦わせ、8個のなかでトップのデータを先頭に入れる。
ここで交換が起こらなければ、2.1. の関係も保たれたままである。交換があった場合、昇格データは 2.1. の関係も保たれているが、降格データは壊れる。
よって降格データのみ再度 1.2. の順で試合を行い、 1.2. の関係を保つようにする。2. で交換があれば後ろはまた 1. の試合を行い 2.1. の関係を保つようにする。
4. 2^N 個ずつの地区まで同様の処理を繰り返すと、これは全データを含む所まで来て、チャンピオンが先頭に入る。また各地区の先頭にその地区のトップが入る形は下の階層まで保たれている。
5. チャンピオンは試合を卒業し、残りの中で再度最強を選ぶ試合を繰り返す。
チャンピオンの後には、1個地区(トップのみ)、2個地区(先頭トップ)、4個地区(先頭トップ)... と並んでいるので、その各トップの中で最強を選べば、次点が選べる。
3. までと同様に入れ替えが起こったときは降格データのみは下層から順にその位置にチャレンジする位置にある各地区のトップと戦わせていき、各地区の先頭にトップが来る形を保つ。
6. 同様にして順番にその位置より後ろにある未確定領域を可能な限り大きな地区になるよう分けていきそのトップ(最も前)で試合をして勝者をその位置に入れるを繰り返すと、ソートが完了する。
7. 引き分けデータがある場合を考える。順番入れ替える時は途中のデータを無視して飛ばすので、飛ばしたデータが飛ばした区間内のどれかのデータと引き分けなら順番は入れ替わる。

引き分けデータを元々入っていた順番に並ぶようにしたい場合は、データと同じ大きさのインデックス配列を用意し、順番を入れておく。交換があった時、この配列も同じ位置を交換する。データ比較で引き分けの時、配列の数を比較し、元々前にあったデータを勝者とすれば良い。(従来他ソートと同じ)

性質について幾つか述べると、まず実行時間はソートデータの乱雑さによって大きく異なる。
元々ソート順に並んでいた場合の動作が最速となり交換回数0となる。正反対に並んでいる場合が最遅となり、バブルと同じ比較・交換回数となる。
下表に算出したものを示すが、比較回数の平均は理論的算出をどうするか、いまいちわからないので空きである。のちに実際の計測値を示す。

データ個数		4		8		16		32		64
	min	max	min	max	min	max	min	max	min	max
比較回数	4	6	12	28	32	120	80	496	192	2016
交換回数	0	6	0	28	0	120	0	496	0	2016

比較回数	min = (N log2 N) / 2	max = (N - 1) * N / 2	
交換回数	min = 0	max = (N - 1) * N / 2	average = (N - 1) * N / 4

またこれの利点としては、並列処理や、ディスク、ネット上のデータをソートする場合の読み込みキャッシュなど、高速化対応との相性が良いというのが挙げられる。
並列処理に強いのは高校野球等の実際のトーナメント戦から明らかである。また、領域のトップデータ入れ替えが生じない限り同じデータが何度も比較に使用される事になるので、キャッシュされていれば高速化される。

2. リージョントップ・ソート (摩擦減少変更版)

基本版を高速化する事を考える。上記で最も時間のロスであるのはデータの降格が発生する度に「地域のトップが先頭に来る」関係を作り直さねばならないところである。
よってデータ降格回数、つまり移動が少なくできればできるほど、試合の数、データの入れ替えは減らせ処理時間が速くなる。
下記の2つの所では、試合毎にいちいち場所を入れ替えず、勝ったデータを覚えておき、最後に最強データとトップの席のデータのみを入れ替えても問題ない。
1. 前の地区から降格したデータがあり、その位置に入るデータを再度トップ候補の中から決めるとき
2. チャンピオンより後の各候補の中から最強データを決めていくとき
これによりソート方法の種別は交換と選択のハイブリッドに変わる。
動作速度は後の実測データで示すように最大・平均回数は改善される。最小回数は変わらない。
(交換の最小回数は元々0のため)

ここで「地域のトップが先頭に来る」関係の保たれた地域に一つデータ入ってきた場合の位置をつきとめる処理は、ソートされたデータに2分検索を持ちいて1件新たなデータを追加する場合と基本的に同じ動作になる。これは2分検索をソートにまで拡張したものと考える事もできる。ただしデータ挿入以降の処理は、当然のごとく本当にデータシフト挿入するだけの2分検索の方が速い。
基本的にはデータをシフトすると「地域のトップが先頭に来る」関係は壊れてしまうが、例外として既にソートされたデータの場合のみ、データをシフトして間に1個データを挿入しても「地域のトップが先頭に来る」関係の保たれた状態になる。

3. リージョントップ・ソート (全地域交換変更版)

降格データに対して「各地域のトップが先頭に来る」秩序を保つ別の方法として、先頭データのみでなく地域全体毎入れ替える方法も考えられる。
これにより、下層地域内「地域のトップが先頭に来る」関係を作り直す処理は不要になる。
ただし、地域のサイズが合う必要があるので 1th 以降の順位を決めていく次点ではこの変更は使用することができない。
動作速度は後の実測データで示すように最大・平均回数は改善される。最小回数は変わらない。
(交換の最小回数は元々0のため)

この方法の持つ大きなデメリットとしてデータ数が 2^N 個でなければ、2^N 個までどのデータにも負けるデータを詰める必要が有り、外部記憶にある非常に大きなデータ数のソートでは余計な領域を多量に必要とする点がある。ある範囲のメモリあるいはディスク範囲の交換をCPUを介さずにできるハードがある、データ性質から2^N 個まで余計なデータを詰めねばならないデメリットは無視できる等あれば、有効性は高くなる。

4. ツートップソート

ここまで「各地域のトップが先頭に来る」規則を定めそれを利用してソートする事を考えた。
そこにもう一つ規則を増やしたリージョントップ・ボトム・ソート(あるいは "Both sides now" ソート)というのを考えたが、後に簡単に述べるがプログラム作成で止まってしまった。
そこで規則を一つ追加する事が本当にソート性質の改善に有効か見る為に、もっと簡単なツートップソートというのを考えた。
「各地域のトップが先頭にきて、地域内の次点者は地域の真ん中(下層の2つの地域に割った時に後方地域の先頭)に来る」ように規則を定めるのである。2個を考えた時前にトップ、次に次点が入る。4個でも一番前にトップ、2個目に次点、2^N個なら一番前にトップ 2^(N-1)個目に次点を入れるというようになる。
データ入れ替えがあった時に、後方地域ではトップと次点の関係を保つ必要があり処理がもう少し複雑になる。
もうプログラムを見た方が速いと思うので詳細説明は省略する。
動作速度は後の実測データで示すように全種類データで最大・平均回数は改善されるが、最小回数は逆に増えてしまう。(交換の最小回数元々0であったのが次点者のデータを真ん中に動かす分が増えた為)
またいくつかのデータをランダムに作ってのソートでは逆に演算回数増えている。
規則を追加する事で、全体としては演算回数減っていくが、中間程度に乱雑なデータのソートでは逆に演算回数が増えている模様である。

またこの方法の持つ大きなデメリットとしてデータ数が 2^N 個でなければ、2^N 個までどのデータにも負けるデータを詰める必要が有る。次点者の位置がデータ個数外だと正しく動作しない為である。

＊ここで他の改造アイデアについても、簡単に述べておく。(プログラムはない)

区域の割り方として、ここでは添え字下から見て連続するデータを同じ地域に割っているが、添字の上の方から見て割る事もできる。例えば8個のデータで2個ずつの地域に割った時 0b000 と 0b100 が同じ地域になる。
理論的な平均時間等の性質は全く変わらないが、時間のかかる並び方の分布を変える事ができる。
ソートデータは完全に一様に乱雑ではなく、例えば強いデータはこの地方に固まっている、弱いデータはこの地方に固まっている等の偏りはあるものであるが、区域の割り方を変えて調整する事で現実のソート時間を短くできる事が考えられる。

トーナメントでデータ末尾側のテイルエンダーを選んでいく方法も考えられる。これは昇順か降順と同じ違いで、方向性を変えるだけでソートの性質は全く変わらない。

ただしチャンピオンを選ぶ方ならトーナメント木を決勝までいった時点で、必ずデータの先頭にチャンピオンが入っているが、テイルエンダーを選ぶ方ならデータ個数が 2^N 個でなければテイルエンダーが末尾に入っていない。(範囲外としてスキップされている。)よってデータ個数が 2^N個かの判定が必要であるので少しだけプログラムが複雑になる。

5. リージョントップ・ボトム・ソート

　未完成のリージョントップ・ボトム・ソートについて簡単に述べておく。これも当初チャンピオン・テイルエンダー・ソートと名前をつけていたが名前負けするのでやめた。
ソートの方法から "Both sides now" ソートという通称も考えたが、この通称通りの結果となってしまっている。
ソートの方法としては、トップと同時にボトムも選ぶというものである。つまり地域先頭にトップのデータを入れるのと同様に地域最後にもボトムのデータを入れるのである。
ここで便利な点は、データ入れ替えが発生しない限り関係は保たれたままなので、トップ側処理とボトム側処理は並列処理が可能だという点である。現在のCPUはスマホですらマルチコアが当たり前であり、並列処理による最速ソートの地位に躍り出る事は十分有り得ると考えたのであった。
データを4個で紙の上でソートしてみると、このアルゴリズムでは最大交換回数5回になるが、他のアルゴリズムは6回か数がわからないかであり、最速になる気はしているのである。

　問題となっている点をいかに述べておく。
・複雑すぎる。プログラムの仕様書を大体書いて、プログラムをある程度書いた所で非常に膨大になる事に気がついた。しかも再帰を含むのだ。プログラムを書き終える事ができても、デバッグは難航するであろう。金にもならないものに要する時間が大きすぎる。
・現在のソートに速度で勝つまでには上記でやってきたような最適化が必要になるであろうが、仮に作成まで完了しても既に複雑すぎるのであるから、これを改良する事は至難の技になるであろう。
　基本となる所はシンプルに実現できる必要があるのである。

　トップ戦で降格したデータの位置決めとボトム戦で昇格してきたデータの位置決めの両方を行わねばならないのであるが、これが思いの外複雑である。領域でトップデータが先頭に、ボトムデータが末尾に入ってる事を保証する為の操作が複雑になるのである。
簡単にできそうだと思われる方もいるかもしれないが、これを読んでいるあなたがソートの専門家である研究者であれば研究の良いネタであると思われれるが、それ以外の人間は時間の無駄になるのでやめた方が良いであろうと言っておく。
ただし、複雑すぎるのは私が「この条件により、この処理は省略できる」というような重要なポイントを見逃してるだけである可能性はある。
私もまた何か思いついて、本ソートプログラムを完成させてみようと思う事があるかもしれない。

6. 計測データ

　比較と交換について別々に回数を計測している。データが多く外部記憶にデータがある場合、交換回数は速度に直結していると考えるためである。
下記数値はプログラムを実行すれば出てくるものであって、ポイントとなる所は既に述べているので特に説明は記さない。

6-1. Champion Sort [Basic]
Data amount:1 Total count:1
Compare [minimum:0 maximum:0 average:0.000000]
Exchange [minimum:0 maximum:0 average:0.000000]
Data amount:2 Total count:2
Compare [minimum:1 maximum:1 average:1.000000]
Exchange [minimum:0 maximum:1 average:0.500000]
Data amount:3 Total count:6
Compare [minimum:3 maximum:3 average:3.000000]
Exchange [minimum:0 maximum:3 average:1.500000]
Data amount:4 Total count:24
Compare [minimum:4 maximum:6 average:5.333333]
Exchange [minimum:0 maximum:6 average:3.000000]
Data amount:5 Total count:120
Compare [minimum:8 maximum:10 average:9.333333]
Exchange [minimum:0 maximum:10 average:5.000000]
Data amount:6 Total count:720
Compare [minimum:9 maximum:15 average:13.000000]
Exchange [minimum:0 maximum:15 average:7.500000]
Data amount:7 Total count:5040
Compare [minimum:11 maximum:21 average:18.333334]
Exchange [minimum:0 maximum:21 average:10.500000]
Data amount:8 Total count:40320
Compare [minimum:12 maximum:28 average:23.333334]
Exchange [minimum:0 maximum:28 average:14.000000]
Data amount:9 Total count:362880
Compare [minimum:20 maximum:36 average:31.333334]
Exchange [minimum:0 maximum:36 average:18.000000]
Data amount:10 Total count:3628800
Compare [minimum:21 maximum:45 average:37.666668]
Exchange [minimum:0 maximum:45 average:22.500000]
Data amount:11 Total count:39916800
Compare [minimum:23 maximum:55 average:46.333333]
Exchange [minimum:0 maximum:55 average:27.500000]
Data amount:12 Total count:479001600
Compare [minimum:24 maximum:66 average:54.000000]
Exchange [minimum:0 maximum:66 average:33.000000]
Data amount:100 Total count:100
Compare [minimum:3451 maximum:4263 average:3901.920000]
Exchange [minimum:2040 maximum:2893 average:2482.380000]
Data amount:512 Total count:512
Compare [minimum:65535 maximum:107675 average:102484.134766]
Exchange [minimum:60513 maximum:70675 average:65489.023438]
Data amount:1000 Total count:1000
Compare [minimum:65535 maximum:405052 average:391165.900000]
Exchange [minimum:65535 maximum:267105 average:250076.896000]

6-2. Champion Sort [Less conflict alternation]
Data amount:1 Total count:1
Compare [minimum:0 maximum:0 average:0.000000]
Exchange [minimum:0 maximum:0 average:0.000000]
Data amount:2 Total count:2
Compare [minimum:1 maximum:1 average:1.000000]
Exchange [minimum:0 maximum:1 average:0.500000]
Data amount:3 Total count:6
Compare [minimum:3 maximum:3 average:3.000000]
Exchange [minimum:0 maximum:3 average:1.500000]
Data amount:4 Total count:24
Compare [minimum:4 maximum:6 average:5.333333]
Exchange [minimum:0 maximum:6 average:3.000000]
Data amount:5 Total count:120
Compare [minimum:8 maximum:10 average:9.000000]
Exchange [minimum:0 maximum:8 average:3.933333]

Data amount:6 Total count:720
Compare [minimum:9 maximum:14 average:12.033334]
Exchange [minimum:0 maximum:11 average:5.900000]
Data amount:7 Total count:5040
Compare [minimum:11 maximum:20 average:16.995237]
Exchange [minimum:0 maximum:15 average:7.677778]
Data amount:8 Total count:40320
Compare [minimum:12 maximum:27 average:21.298412]
Exchange [minimum:0 maximum:20 average:10.465873]
Data amount:9 Total count:362880
Compare [minimum:20 maximum:35 average:28.061905]
Exchange [minimum:0 maximum:22 average:11.408995]
Data amount:10 Total count:3628800
Compare [minimum:21 maximum:39 average:31.463148]
Exchange [minimum:0 maximum:27 average:13.755274]
Data amount:11 Total count:39916800
Compare [minimum:23 maximum:46 average:37.599262]
Exchange [minimum:0 maximum:30 average:15.753829]
Data amount:12 Total count:479001600
Compare [minimum:24 maximum:54 average:42.750179]
Exchange [minimum:0 maximum:36 average:19.101740]
Data amount:100 Total count:100
Compare [minimum:1074 maximum:1249 average:1161.850000]
Exchange [minimum:302 maximum:377 average:336.980000]
Data amount:512 Total count:512
Compare [minimum:10430 maximum:11294 average:10900.849609]
Exchange [minimum:2364 maximum:2584 average:2477.164062]
Data amount:1000 Total count:1000
Compare [minimum:24502 maximum:26346 average:25445.341000]
Exchange [minimum:5061 maximum:5421 average:5245.574000]

6-3. Champion Sort [Full region exchange Altenation]
Data amount:4 Total count:24
Compare [minimum:4 maximum:5 average:4.666667]
Exchange [minimum:0 maximum:6 average:3.000000]
Data amount:8 Total count:40320
Compare [minimum:12 maximum:22 average:18.380952]
Exchange [minimum:0 maximum:23 average:11.771429]
Data amount:512 Total count:512
Compare [minimum:9899 maximum:10830 average:10446.103516]
Exchange [minimum:2845 maximum:3737 average:3253.757812]

6-4. Two top sort
Data amount:4 Total count:24
Compare [minimum:5 maximum:5 average:5.000000]
Exchange [minimum:2 maximum:7 average:4.333333]
Data amount:8 Total count:40320
Compare [minimum:14 maximum:24 average:19.842857]
Exchange [minimum:3 maximum:23 average:12.766667]
Data amount:512 Total count:512
Compare [minimum:29664 maximum:36859 average:33055.343750]
Exchange [minimum:17361 maximum:22830 average:19948.535156]

***gcc q-sort**
Data amount:8 Total count:40320
Compare [minimum:14 maximum:29 average:19.171354]
Data amount:11 Total count:39916800
Compare [minimum:20 maximum:63 average:31.485238]
Data amount:512 Total count:512
Compare [minimum:4427 maximum:5081 average:4598.078125]
Data amount:1000 Total count:1000
Compare [minimum:9660 maximum:10725 average:10018.899000]