Is it new sort algorithm?

I bethought of one sort algorithm. I couldn't find the same one on the internet.
I improved it with some idea and I thought its performance may overcome a present fastest sort.
If it becomes so, thesis submitted to an authoritative academic journal shall pass and it probably rewrites textbook of computer algorithms. I studied the algorithm seriously, but only found that it is not so easy. Only the idea of cheap tricks can't exceed predecessors effort.
Even if it is new algorithm, labor of study and paper writing doesn't balances with the value I will get. To begin with, I am not a scholar, only an application maker. The gain is, even if maximum level, adding its article to Wikipedia. But algorithm itself has no impact like sleep sort. It will be not a worth for the advertisement of my application. And I think somebody already thought out it before, so much simple and silly. All kind sort algorithm regulated on the internet is not the right idea. My motivation is deflated quickly.
But the possibility it is new one still remains. I think I can put a mark "I found it this point of time" on internet by displaying something outcome document. I closed materials until now and wrote below.
Still, I have hope that other more flash can go through the tunnel to its ideal shape. I have not given up and don't think it is the final conclusion.

Sort algorithm explanation is from here.
I name it "Region top sort". At first, I named “Champion, sort” but changed my mind. It can’t live up to its name. Present name fits more to sort method character. There are basic version and three alternate versions of sort algorithm and program from here.

1. Region top sort (Basic)

Cardinal sort method is a tournament match to choose a champion. Already there has been past sort algorithm using this, I know. However, past sort using tournament demands the memory area for tournament result besides sort data. This sort memorizes tournament result in sort data area itself. Extra memory besides sorting arrays is not necessary. (If draw data sort order is not care) It is one of this sort advantage.

Classification: Sort method is exchange. It is not stable sort.

Algorithm explanation:
In this explanation, I unify data array index start number. Champion is 0th, runner-up is 1st. (This index number is important.)
At first, arrange all data horizontally and imagine the tournament tree above it.
From the front, divide the data into sections including 2 items each. Undermost layer first round trees stand on it.
From the front, divide the data into sections including 4 items each. Next layer second round trees stand on first round trees. The section consists of 2 sections of undermost layer.
Similarly, divide into section 8 items each, 16 items each... 2^N items each (while 2^N < amount). And these form into stratification. The tree continues up to the final match.
Play tournament match from lower layer and do this sorting characteristic process "Set the top items to the foremost of the section”. Top item means survivor of a tournament match included in the section. The high layer match is the match between tops, foremost data of two low layer sections. While sorting, the rule "each section's top stays foremost" must be kept. (To keep it, the process described later is necessary)
After the final match of the tournament, the foremost item of all data is champion, strongest data. It graduates from the matches for sorting. Choose strongest item of rest and set it foremost of rest data.
This next or lower ranking decision after champion is the final winner of matches among the tops of each section. Because "section top stays foremost", the best time is the case dividing the rest as large sections as possible.
For example, after the champion(0th) decision, 1st location is section including one item only, and section including 2 items and top foremost, section including 4 items and top foremost... continue.
Set 1st location the winner of ranking decision matches among section top items.
The maximum section size that the location is top comes out from its array index number. The number starts from 0. Convert index number to binary and search first location of 1 from lowest digit. Binary by this 1 and lower 0 is the size. For example, if index is 0b???1000, it is the top of 0b1000(8) items among it and after. If index is odd number 0x?????1, it is the top of itself only. (It can easily see when you write tournament tree and set binaries to its roots from the front.)

Repeat this process. When no pending data remains and all match ends, this sort finishes.

The process how to keep "the top of the section stays foremost of it" rule of this sort is shown below.
Condition branches by the result which wins in the match between the front section top and the back section top.
In the case front section top wins, the front section top location is equal to the top of the high level section including both sections. The rule in high section is kept. Nothing is necessary.
In the case back section top wins, in the front section, the rule is kept. Because the top item only becomes stronger one. In the back section, the rule is broken. The demotion item to back section comes top location at first, but it has possibility from top to bottom of the section. In the back section, the matches of the tournament tree, including the top location from low to high layer, are necessary for regaining the rule.
If this match accrues data exchange again, the lower layer section top demotion item needs the same process. That is, recursive process is necessary.
The less data exchanges are, the less matches for keeping the rule become, and the faster sort finishes.

I describe the process again more concretely for assisting understanding, though I think seeing program is better than reading this.
1) Divide the data into sections including 2 items from front to back. Play matches between next items in sections. Set winner front, loser back in these sections.
2) Divide the data into sections including 4 items [including two 1)-sections] from front to back. Play matches between top items of 1)-sections. Because winner was set front by 1)-matches, play matches between front items of 1)-sections. Set winner data of 4 items foremost.
If no exchange accrues, rule-1) is kept. If an exchange accrues, the promotion item keeps rule-1), but the demotion item is possible to break rule-1). In 1)-section of demotion item, play 1)-match between next items to keep the rule-1).
3) Divide data into sections including 8 items [including two 2)-sections] from front to back. Play matches between top items of 2)-sections. Because winner was set foremost by 2)-matches, play matches between front items of 2)-sections. Set top data of 8 items foremost.
If no exchange accrues, rule-2)1) is kept. If an exchange accrues, the promotion item keeps rule-2)1), but the demotion item is possible to break rule-2)1). In 1)-section of the demotion item, play 1)-match, and in 2)-section of the demotion item, play 2)-match, to keep the rule-1)2).
If an exchange accrues again by 2)-match, play 1)-match in 1)-section of the demotion item, to keep the rule-1).
4) Repeat the same process until 2^N size sections, those covers whole data. Champion comes foremost of whole data. And in each section of every layer, the top data are set foremost.
5) Champion graduates the matches. The matches continue to choose the strongest item from the rest.
After champion item, section including 1 item (top itself only), section including 2 items (foremost is top), section including 4 items (foremost is top)... follow. Runner-up item is the strongest item of these rest tops.
Like 1)2)3), if an exchange accrues, the tops these can challenge to the demotion item location play matches with the location item, to keep the rule "section top comes foremost".
6) Repeat the same way as previous and decide following raking items. Divide the unsettled part after the location as large as possible and set the winner of top (section foremost) to the foremost location.
Repeat until sort finishes.
7) Thinking the case draw data exist. In the exchange process, items between are no regard. If there is draw item with the jump item between, these order becomes upside down.
If draw items must follow the original order, prepare the same size index array as data, and set order in it. Whenever an exchange accrues, exchange the same location of this index array. When items comparison is draw, compare the number in this index array and judge the original front side item as the winner. (Same as past sort)

Performance: The execution time differs by sort data randomness.
The fastest case is originally following sort order. Exchange time is 0.
The slowest case is just the opposite order, just the same comparison/exchange time as bubble sort.
Below table is theoretical calculate time. Comparison average time is blank because it is not sure. Actual measurement value table exists the last chapter of this document.

Data amount(N)	4		8		16		32		64	
	min	max	min	max	min	max	min	max	min	max
Comparison time	4	6	12	28	32	120	80	496	192	2016
Exchange time	0	6	0	28	0	120	0	496	0	2016

Comparison time	min = (N log2 N) / 2	max = (N - 1) * N / 2	
Exchange time	min = 0	max = (N - 1) * N / 2	average = (N - 1) * N / 4

As this advantage, speed up technology like parallel computing and read cache (of sort data on the disk/internet) can improve it. Parallel computing is clear from fact tournament match. Unless otherwise accrued section top data exchange, the same data is used comparison many times. Speed up by data cache can be known from it.

2. Region top sort (Less conflict remodeling)

Trying speed up of the basic version.
The main time loss is when item demotion accrues. "Set the top of the section to foremost of it" rule re-order process needs time. As few as possible this item demotion time, the match and the exchange become fewer and the execution time becomes faster.
At below two situations, don't exchange item locations after the match. Memorize only the strongest item location. Exchange is only one time at last between the strongest item and the top location.
1. When demotion item comes from the front section. (The process to decide the top location item from its candidates)
2. Process to decide the each ranking item after champion from its candidates.

Classification: Sort method becomes hybrid of exchange and selection.

Performance: Like actual measurement after, the maximum/average time improves. Minimum time is same. (Minimum exchange time is originally 0)

This demotion location searching process resembles Binary Search (one item addition to the sorted data). Item move process is different and Binary Search is faster naturally. Binary Search does the data shift only between two locations because the other data are already sorted. The data shift between doesn't fit non sorted data.
It may be thought enhanced Binary Search version. But it can't improve any more by the Binary Search way.

3. Region top sort (Full region exchange remodeling)

There is another way to keep "Set the top of the section to foremost of it" rule. Exchange the section whole data, not only the foremost data. By that, the re-order process of lower layer in the section is not necessary.
However, it can apply only the case that the match is between front section and back section and both sections size is same.

Performance: Calculating time maximum/average is improved like actual measurement data later. Minimum time is same. (Minimum exchange time is originally 0)

This method has one not negligible demerit. When data amount is not 2^N, it must fill weakest items until 2^N in data rear side. If sort data are very large in external storage, these waste extra storage.
I think effective cases of it exist. The case the hardware without using CPU can exchange selection range of memory or disk. Or the case data nature can be without regard to 2^N size.

4. Two top sort

Until here, sort uses "Set the top of the section to foremost of it" rule. I add one more rule and make next version named "region top bottom sort". But I stopped programming in the middle because it's too complex.
I think out more easy this "two top sort" instead of it. I try to check that rule addition is effective to sort performance improvement or not.
Add "Set runner-up of the section to the center (foremost of the lower layer back section) of it" rule.
In a section of 2 items, front is top and back is runner-up. In a section of 4 items, foremost is top, and 2 items after is runner-up. In a section of 2^N items, foremost is top, and one 2^(N-1) items after is runner-up.
As program modification, at data exchange, in the back section, the process for keeping both top and runner-up rules is necessary. I omitted a detailed explanation because seeing program is faster.

Performance: Calculating time maximum/average of all permutations case is improved like actual measurement data later. But the minimum time of it increases conversely. (Minimum exchange time is originally 0.
(This method always needs the steps those runner-up item goes to the rule location and goes to the order location.)
The sort time of the some random data increases. By adding this rule, normal case middle degree disorder data sort time increases.

This method has a demerit. When data amount is not 2^N, it must fill weakest items until 2^N in data rear side.

*** I state here other alteration idea simply. (No program)**

(a) Until now, all items included one section is continuous. Index number binary is scanned from downside. There is another way, scanning index number binary from upside. For example, thinking 8 data sort. Set 0b000 item and 0b100 item to a same 2 items section.
Theoretical performance like average time is just the same if data are ideal disorder. But the time distribution is not same. Real sort data has something deviation, like strong items gather in one district, weak items gather in another district. By adjustment of the way how to interpret index number binary to section, actual sort time can become faster in some cases.

(b) Choose tailender instead of champion, very back side of data by tournament match.
It changes only sort direction, same as switching ascending or descending order. Sort performance is just same.
It has demerit. Champion always stays foremost of data after the final match of tournament tree. Unless the data amount is 2^N, tailender doesn't stay very back after the final tournament match. The match is skipped because of out of range. The program is a little complex for the judgement data amount is 2^N or not.

5. Region top bottom sort

This region top bottom sort is unfinished. But I describe about it simply.
At first, I named "Champion tailender sort" but changed by the reason the name is too much to it.
Next, I gave it the name "Both sides now" sort. Present situation follows this name song lyric.
Sort method is, at the same time choosing the top of the section, choosing bottom. That is, set top item to foremost, similarly set bottom item very back of the section.
Next shown is remarkable point. So long as no data exchange accrues, the rules are kept, both top side and bottom side processes can be executed parallelly. Even in smart-phone, multi-code CPU is common at the present time. I think this sort may fully jump to the fastest position by parallel computing.

Problems presently I face are below.
* Too complex. I wrote almost program spec sheet, and started to program. The program becomes very large before making all. And it consists of recursive program. Even if I can finish writing program, debug would meet the difficulty. In spite of it can't convert into money, the time necessary is too large for me.
* Even if I can make it work, it is already too complex to improve it. The improvement would be necessary to speed up. The base version must be incarnated from simple structure.
Both demotion item by top match and promotion item by bottom match, cause new match for both "top" and "bottom" rules. It is unexpectedly complex. This sort process needs that the rules are kept. But in some case, it is unwarranted. Before the match for demotion item in the section, the rule may be broken by promotion item. And reverse case, too.

You who read this paper until here may think it is easy. If you are a computer science scholar, especially professor of sort, it may good material for your work. But for other person, you shouldn't try to take over this. it is only waste of time, I warn.
However, still I remain the hope this method would be fastest one if it will complete. The reason is my sort examination on paper of 4 items case. This algorithm exchange maximum time is 5 times. Other algorithm is 6 times or hard to calculate. Too complex reason may be I overlook some important condition that can omit and make simple some process. I may find out something new idea to finish this sort program by.

6. Time measurement

I measured comparison time and exchange time independently.
Exchange time largely affects to calculating speed, when sort data is too large and exists in external storage.
Below value comes from the program execution result.
I already mentioned essential point previously. So I don't add explanation here.

6-1. Region top 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. Region top 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 remodeling]

```
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]
```