A Sorting Method by Dong-Keun Shin printed on August 9, 1998

Dr. Dong-Keun Shin

Hwa Shin Building, Suite 701
705-22 Yuksam-dong, Kangnam-gu
Seoul 135-080, Republic of Korea

Dong-Keun Shin 8/9/98

Abstract

The author's sorting algorithm creates Shin's tree to store input data. Shin's tree has a binary tree data structure to let Shin sort algorithm have O(N) sorting performance and O(1) searching performance. The author's effort in finding a new sorting algorithm is to design a sorter that sorts data efficiently in parallel. Although Radix sort and hashing have O(N) time complexity, they are not appropriate for parallel processing. Shin's tree can grow both in local or main memory and in auxiliary storage, so Shin's subtrees will be merged to create a whole Shin's tree. Creating local subtrees will help reading data in parallel; thus, each processing element can effectively reads and sorts data in its own memory space. Radix sort does not provide O(1) searching performance and hashing does not have inherent characteristic of parallel processing due to its requirement for a single hash table. Therefore, Shin sort algorithm is strongly recommended.

Introduction

Computer scientists have searched for the best sort method for about a half century. Sorting is a widely and frequently used operation in computer engineering. Sorting coupled with searching is the most fundamental computer operation. In addition, varieties and characteristics have been found in sorting algorithms. There are also major criteria to compare one sorting method to another in selecting the best sorting method. Therefore, finding an efficient sort algorithm has been one of the most interesting problems in computer science.

As there is a rule in any game, there are criteria in choosing the best sorting method. The criteria should be well defined so that the comparative study for choosing the best sorting algorithm has to be a fair game. Two major criteria are time complexity and inherent characteristic of parallel processing. The first criterion, time complexity, draws a very fine line between one group of sorting algorithms and another group. When the number of input item is **N**, the best possible sorting algorithm's processing time is proportional to **N** because key in each input item should be read at least once to be sorted. The time complexity of the best group is O(**N**). There are four major time complexities in currently well-known sorting algorithms. Their worst case time complexities are O(**N**), O(**N**log**N**), O($\mathbf{N}^{3/2}$), and O($\mathbf{N}^2$).

copied by 98.8.10.
82-342-713-4455

Sorting algorithms such as insertion sort and bubble sort are known to have $O(\mathbf{N}^2)$ processing time performance. The selection (or quadratic) sort has $O(\mathbf{N}^{3/2})$ time complexity. There are a few sorting algorithms with $O(\mathbf{N}\log\mathbf{N})$ time complexity such as quick sort (in it's average case), heap sort, binary search tree sort, shell sort, and merge sort. There may be more than sorting algorithms mentioned above. The aforementioned algorithms have the characteristic of key string comparison in common. Some people group them by calling comparative sorting algorithms due to their common characteristic. When one considers sorting as a preprocessing for searching and a sorting operation organize input data for efficient searching, hashing can be considered as a sorting method. Hashing translates a key string to a bucket number in a hash table and stores input item in the hash addressed bucket of a table. If a hash table with a sufficient number of buckets is provided, searching for a key in such a hash table requires O(1) processing time complexity. Therefore, people may consider hashing as a kind of sorting, they may call it distributive sorting. Hashing has been attractive because of it's $O(\mathbf{N})$ distribution process and O(1) searching process. Radix sort attracts people's attentions because of it's $O(\mathbf{N})$ sorting process. It is better than other algorithms' time complexity such as $O(\mathbf{N}\log\mathbf{N})$, $O(\mathbf{N}^{3/2})$, and $O(\mathbf{N}^2)$. However, like other well-known sorting methods, Radix sort lacks competitiveness compared with the hashing because Radix sort's $O(\log\mathbf{N})$ searching process is inferior to hashing's O(1) searching process. Currently well-known sorting methods produce output list, and binary search, which is the best search method that has $O(\log \mathbf{N})$ processing time, may be used on the list. As a result, as far as sorting's and searching's time complexities are concerned, a sorting algorithm which has $O(\mathbf{N})$ sorting process with O(1) searching process is the most ideal one.

Computer scientists including the author have been eager to search not only for a sorting algorithm that has O(N) sorting process and O(1) searching process but also for a sorting algorithm which has inherent characteristic of parallel processing. Because hardware affordability has been increased enormously, people may provide not only more processors but also special hardware architecture to accelerate the sorting operations. Thus, within a computer system, more processing elements will be provided in the future. They may be either hardware processing elements or software processing elements. In most cases, their structures in a system will be identical. In case of the most ideal sorting algorithm, if $\boldsymbol{K}$ processing elements are provided, the processing speed may be increased about $\boldsymbol{K}$ times. Not every sorting algorithm does have such inherent characteristic of parallel processing. Some sorting algorithms has to perform series of operations in serial. In future, more processing capabilities will be provided to a computer system; therefore, a sorting algorithm with highly inherent characteristic of parallel processing will have more potential.

In this paper, the author will introduce his sorting algorithm which has O(N) sorting process with O(1) searching process and which also has inherent characteristic of parallel processing. Dong-Keun Shin, the author, discovered a new sorting algorithm on July 3rd, 1998 and wrote a handwritten manuscript on July 4th, 1998. If the author's sorting algorithm is a new one, the new sorting algorithm is termed "Shin's sort algorithm" or "Shin sort algorithm". This paper explains Shin sort algorithm, illustrating two

examples to show how the algorithm sorts input data. Several issues in sorting methods are discussed. And the paper finally provides its conclusion.

Shin Sort Algorithm

Shin sort algorithm reads and stores character strings or integers numbers into a binary tree character by character or digit by digit respectively. Thus, each node in the tree primarily represents a character in an input key string or a digit in an input integer number. It stores each character or each digit in an input key creating a node, and it links the created nodes together by each node's pointer for its left child. If a part of the current input string has been already stored in the created tree, it uses the relevant portion of the tree and it uses the lowest character or digit node of the portion for different portion of the current input string. The lowest node's right child pointer in the relevant portion will point to the first character node or the first digit node in the other portion of the string, which will create another branch in the Shin tree. This algorithm may read an input string which has been read and stored already, or it may read an input string that is identical to a portion of stored input string in the tree. In both cases, the algorithm increments the counter in the lowest node of the input string or the portion of the stored input string. Each node in a tree has data structure for a character, digit position (for integer numbers input case only), counter, and pointers to left and right child nodes.

If there is a special node for associated information for a key, each node in Shin's tree should include a pointer for the associated information for any key. When associated information node is pointed by a node in Shin's tree, the node in Shin's tree must be the last character or last digit in a key. This node is termed "terminal node for a key" or "terminal node", and the node is at the last character or last digit of a key. Thus, in Shin's tree only a terminal node points to an associated information node.

Examples for Shin Sort Algorithm

The First example will show how the Shin sort algorithm sorts input: 316, 273, 4329, 9837, 16, 274, 107, 273, 5, and 272 creating a Shin tree.

(1) The first input 316 is read, and three nodes, i.e., 3, 1, 6, are made and linked together by node 3's and node 1's left child pointers. (Figures 1-9 will be inserted later.)

(2) The second input 273 is read and inserted into the tree. the node "2" in the input 273 is smaller than "3" in the first input 316, so the node "2" will have node "3" as right child of the node "2".

(3) The third input 4329 is read and found to have different digit position. Thus, zero node becomes the root node and the Shin tree is changed to include the input 4329.

(4) An input 9837 is read and inserted into the tree.

(5) An input 16 is read. Because 16 is a two digit integer, zero node is added.

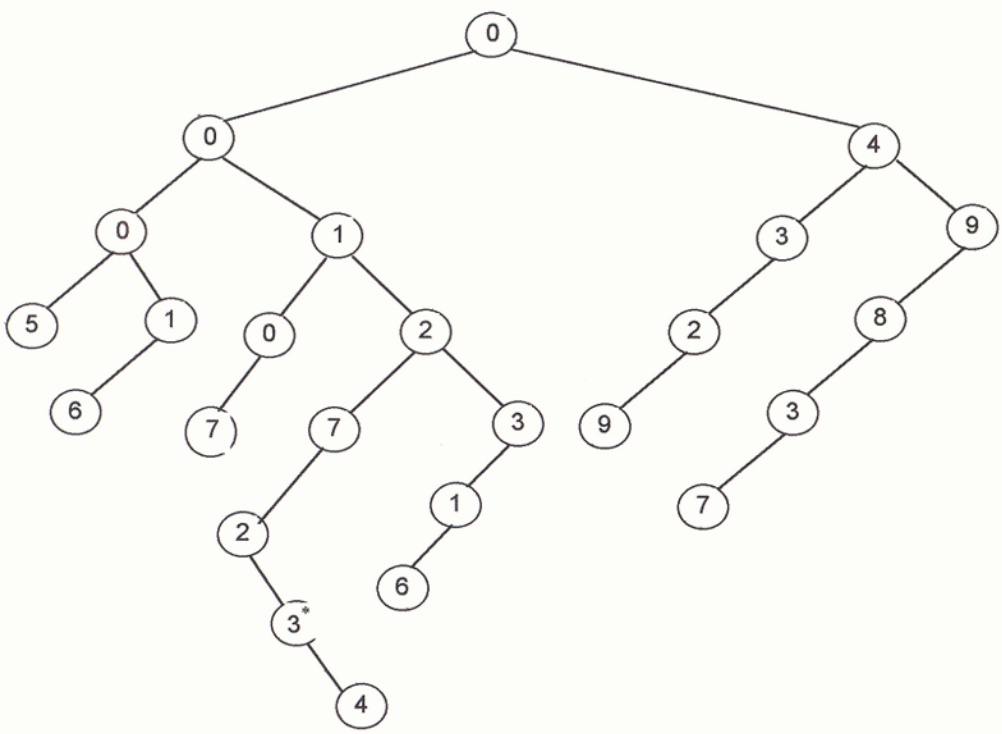
(6) An input 274 is read. The algorithm recognizes '27' portion in the previous input 273, so the algorithm puts 274's node "4" in the right child of node "3" in the input 273.

(7) Then an input 107 is read and inserted between 16 and 273.

(8) Then the algorithm reads another input 273 and stores in the Shin tree.

(9) Next Shin sort algorithm will read an input 5 which is a single digit number.

(10) Next it reads 272 as a final input. It has to insert it right before 273.



Print out numbers stored in the tree.

Output List: 0005, 0016, 0107, 0272, 0273, 0273*, 0274, 0316, 4329, 9837.

* Since counter in node 3 of 273 has been incremented and the value is set to one. Thus, the input number 273 is printed one more time in the output list.

The second example will show how the Shin sort algorithm sorts input: LEE, KIM, KING, KITE, KENT, KIN, KANT, KILE, KIMMY, KIMBERLY, KITE, KID, KINS, JACOB, KANG, KITE.

< Figures and explanations will be inserted later.>

Properties of Shin Tree

Shin's sorting sorts input data, creating Shin tree in reverse order. Reverse Shin tree is also a binary tree that has key string nodes for each character or digit in a string of a key are connected by their right child pointer and that has nodes' left child pointer connects greater character or digit in internal expression number (e.g., ASCII, EBCDIC). In this case, keys shown in the Shin's tree become more recognizable because letter nodes are placed left to right as letters in a key are placed. The tree in it's reverse order will be traversed to print out keys in the tree. Preorder tree traversal on this tree will generate output list in reverse order. When a key word is a portion of another key word as maked in a node of Shin's tree, tree traversal has to produce the shorter keyword prior to the longer keyword for the output list. At this point, one may assume there is another tree traversal, so called suorder which traverses tree in the order of right child first, the node itself second, and left child third. Then the suorder will printed keys in reverse Shin tree in right order. Therefore, the author wants to include suorder in tree traversals with preorder, inorder, and postorder.

Examples of Reverse Shin Tree

<Two examples of Reverse Shin Tree will be included here later.>

The node for the associated data can be expressed as shown in Figure. Each character node in each tree has a pointer to the node.

<Figure for the data structure for the associated data will be given later.>

Discussion

Everyone knows that no sorting method can have better than O(**N**) time complexity because the method has to read every key at least once. It is true that literal sorting methods(termed because of their characteristic: character-based or letter-for-letter) have an advantage in acquiring O(**N**) processing time for sorting because they do not compare keys but they find key position from letters of the key. Comparative sorting methods deal with number of input keys due to subsequent key comparisons in their sorting processes. Because of the above reason, it is impossible to acquire O(**N**) time complexity for any comparative sorting. Is O(log**N**) time complexity necessary for searching in a sorted input item? The answer is yes, there is no better search method than the binary search which requires O(log**N**) processing time. One has to compare the searched key with a key in the list to eliminate keys do not have any potential to be the searched key. If there is a plain sorted list, key comparison is mandatory. Then the

divide and conquer strategy in the binary search provides the best way to find the searched key because the key itself does not provide any information for its location. Number of comparisons are required in searching a key in a plain sorted list will be proportional to log**N**. Therefore, when a searching is performed on a plain sorted list, there is no search method better than the binary search. Because finding the position of a key in a plain sorted list requires O(log**N**) for all N input keys, comparative sorting algorithms has at best O(**N**log**N**) time complexity. Because of the fact, a new sorting method should avoid producing a list of plain sorted output that every current sorting method produces. Associated new search method has to find a key immediately from a data structure using it's key information like hashing does in searching. Consequently, in order to achieve O(N) sorting processing time and O(1) searching process time, one has to store a data item using it's key without any key comparison and retrieve the item using the information in the key. Radix sort has mixed approach. In other words, it has a literal (or character-based) approach like Shin sort, but it inserts keys into queues with other keys. Inserting a key into a series of queues loses the information about the key's unique location in an output list. As a consequence, it's search operation still requires O(log **N**) processing time which is no better than search operation require in other sorting methods. Radix sort cannot be recommended for a highly parallel sorter because it lacks inherent characteristic of parallel processing. Radix sort serially moves data item from one queue to another based on counted position's character value or digit value in the item's key. From the first queue to the last queue, a data item should wait in a queue to be moved in it's turn. The data items are moved one by one in serial. Because of the nature of this process, these movements cannot be processed in parallel. Therefore, the sorting speed of a Radix sorter cannot be accelerated in proportional to the number of processing elements are provided.

One may point out that Shin's tree requires some more memory space for keys. Because a key in a data item has associated data which is usually much larger than the key itself, memory space requirement for keys cannot be a considerable problem. It is acknowledged that the author made a mistake in analyzing memory space requirement for Radix sort in his July 4, 1998's handwritten manuscript. The Radix sort does not require more memory space than Shin sort does.

Conclusion

The author's Shin sort algorithm is presented in this paper. While reading input data, Shin's sorting method creates Shin's tree that has a binary tree data structure like hashing creates a hash table. The Shin sort algorithm is superior to currently existing sort algorithms in terms of two major criteria: sorting and searching time complexities and inherent characteristic of parallel processing. The Shin sort algorithm has O(N) time complexity and O(1) time complexity for sorting and searching respectively. No currently existing sorting algorithms has as good time complexities as Shin sort has. Parallel readings are feasible in Shin's sorting algorithm because it can create subtrees locally and merge them together later. Although hashing has the same time complexities that Shin sort algorithm has, it cannot create many partial hash tables while reading input data. Thus, hashing lacks inherent characteristic of parallel processing in

its nature. Shin sort's inherent characteristic of parallel processing shows a promising sign for future parallel sorter. The author recommends his proposed Shin sort algorithm as the best sorting algorithm.

Bibliography

[1] Aho, A. V., Hopcroft J. E., Ullman, D. U. *Data Structures and Algorithms*, Reading: Addison-Wesley, 1985.

[2] Culler, D. E. <To be filled later.> *Communications of the ACM* <To be filled later.>.

[3] Shin, D. K. "The Theory of Massive Cross-Referencing." *The Proceedings of the Eighth International Conference on Software Engineering and Knowledge Engineering*, Jun. 1996: 545-52.

98.8.10

98.8.10

98.8.10

121 General Hospital
Box 314
APO AP 96205-0017

August 9, 1998

Dean Paul R. Gray
College of Engineering
320 McLaughlin Hall
The University of California at Berkeley
Berkeley, CA 94720

Dear Dean Gray:

I am enclosing my paper that I have just printed out. Although the paper has not been completed yet, the paper will show how my paper about a new sorting algorithm will be. I hope you to distribute copy of this paper to some UCB faculty who may be interested in this work including Professor Alan J. Smith and Professor David E. Culler. Thank you.

Sincerely,

Dr. Dong-Keun Shin

cc: UCB faculty

121 General Hospital
Box 314
APO, AP 96205-0017

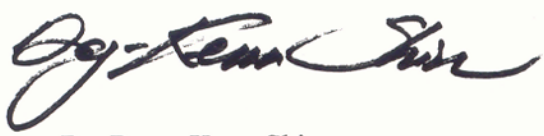
August 9, 1998

Professor Mona Zaghloul, Chair
EECS Department
School of Engineering and Applied Science
Phillips Hall
The George Washington University
Washington, DC 20052

Dear Chair Zaghloul:

I am enclosing my paper that I have just printed out. Although the paper has not been completed yet, the paper will show how my paper about a new sorting algorithm will be. I hope you to distribute copy of this paper to some GWU faculty who may be interested in this work including Professor Arnold C. Meltzer. Thank you.

Sincerely,



Dr. Dong-Keun Shin