International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 7 | ISSN : 2456-3307

Sorting By Element Position Count

Rayudu Srinivas
Haramaya University, Haramaya Institute of Technology, Harar, Dire Dawa, Ethiopia, India

ABSTRACT

The concept of sorting was introduced in 1880, but the study of research on sorting has started in the year 1950 which is cited as arranging data in a particular format. Rapid growth of information and data in our world leads to augment the development of sort algorithms. The facet which attracted a great deal of research, is developing sort algorithms through improved performance and decreased complexity. Sorting algorithm specifies the way to arrange data in ascending or descending order. Sorting is also used to represent data in more readable formats. Numerous sorting methods were developed but still there is a scope for sorting in research because no one method is suitable for all the applications. In this paper a novel method to sort given list of elements is presented. In this sorting, each element position count is calculated and based on this count value elements are arranged in order. The advantage of this method is easy count the position and swap operation are not required.

Keywords : Bubble Sort, Efficient Sort, Heap Sort, Novel Sort, Quick Sort, Sort, Tree Sort.

I. INTRODUCTION

Sorting can be performed on different types of data like numeric, alphabetic, string, etc. Sorting on numerical values works in such a way that the comparison of elements can be done either based on their actual values or ASCII values and alphabetic data can be sorted using ASCII values hence the strings can be compared and sorted [1-4].

There are several factors that are to be taken into consideration while comparing various sorting algorithms [5]. The first among them is time complexity. It specifies the time taken by the algorithm to run. The time complexity of an algorithm is generally specified using big O notation, where the O represents the complexity of the algorithm. The next factor is the stability which means the algorithm keeps elements with equal values in the same relative order in the output as they were in the input [6-14]. Some sorting algorithms are stable by their nature such as insertion sort, merge sort, bubble sort, while some sorting algorithms are nonstable, such as quick sort. Any nonstable sorting algorithm can be modified to stable sorting algorithm. The third factor is memory space, any algorithm which uses recursion need more copies of sorting data that affect to memory space. There is an extended research on sorting for enhancement of the sorting algorithms to maintain memory space and improve efficiency as new applications demand fast and efficient algorithms. The old basic researches result in the establishment of optimality for new sorting techniques, and new theory challenges for many other classic fundamental algorithms.

Widely Used Sorting Techniques

There are many types of sorting techniques which are differentiated by their time and space requirements. Some of the sorting techniques like bubble, insertion, selection, quick, merge and heap sort are widely used [15-20].

Time Complexity of Algorithms

Time complexity of an algorithm specifies the total time required by a program to run. The time complexity of an algorithms is most commonly expressed using the big O notation. Time Complexity is most commonly estimated by counting the number of elementary functions performed by the algorithm. The algorithm's performance may vary with different types of input data and hence for an algorithm we usually use the worst-case time complexity. Because that is the maximum time taken for any input size. Not only big O notation, Big Omega, Big Theta, Little Oh and Little Omega are also used to specify the time complexity.

II. LITERATURE SURVEY

Radix sort was developed by Herman Hollerith in the year 1887 on Tabulating machines [21]. Here tabulating machine is an electro mechanical machine design to assist in summarizing information and later accounting. Radix sort is also known as “Bin sort”, “Bucket sort” or “Digital sort”. This sorting works on the principal of sorting by Distribution.

John Von Neumann invented Merge Sort in 1945 [22]. Merge sort is implemented using the Divide and Conquer algorithm. Merge sort involves two main phases: First is, dividing the input list into N individual element lists; second is merging the divided list into a sorted one.

Insertion sort was proposed by John Mauchly in the year 1946 [23]. Insertion sort is considered as the best among the simplest comparison based sorting algorithms since it is the most intelligent and efficient algorithm for N number of elements. Counting sort was developed by Harold Seward in the year 1954 [24]. It is the combination of both Ultra Sort and Math sort. Counting sort is an efficient sorting algorithm with asymptotic complexity O(n+k). This sorting is based on the smallest and the largest array which is taken for sorting.

Bubble sort was introduced by Inverson in the year 1956[25]. Bubble sort technique works in such a way that it compares two adjacent elements of the input list at a time and swaps them so that they are placed in sorted order. The input data is considered to be sorted completely if there are no swaps occurring in the iteration. Shell sort is also called as Shell’s method [26]. The shell sorting technique is the oldest sorting algorithm in data structures. It was developed by Donald Shell. In 1959 the first version of this sorting algorithm introduced. This algorithm is extension for insertion sort.

Topological concept was first studied in the year 1960 in the content of PERT technique for scheduling in project management. Topological sort is performed through a directed graph in linear order of its vertices. Topological sort is also called as ordering that which is possible if and only if the graph is Directed Acyclic Graph (DAG).

Quick sort was developed by Tony Hoare in 1959 and it is published in the year 1961 [27]. It is commonly used algorithm for sorting; quick sort is the most powerful sorting algorithm. Now a day’s quick sort is used regularly. Quick sort can be worked faster by two or three times when compared to merge and heap sorts. This algorithm works faster and there is no need of temporary memory.

Selection sort algorithm is the simplest sorting algorithm which is based on comparisons done among the elements that are to be sorted [28]. It was introduced in the year 1962. In other way, it is an in-place comparison based algorithm. Tournament sort is a sorting algorithm which is improved version of selection sort by using a priority queue to find the next element in the sort [29]. This was introduced by K. E. Iversion in the year 1962. Selection sort takes

O(n) to select the next element of n elements.

C. L. Mallows introduced Patience sorting in 1962 [30]. The patience sorting algorithm can be applied to the process in order to control them with a series of measurements for the existence of a long increasing subsequence.

J. W. J. Williams invented Heap sort in 1964 [31]. This was also the birth of the heap data structure, presented by Williams and it is useful in its own right. R. W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the tree sort algorithm in the same year.

Bitonic Sort was created by Ken Batcher in 1968 [32]. This algorithm is a comparison-based sorting algorithm that can be run in parallel and therefore it is also known as classic parallel algorithm for sorting sequence of elements. It is the one of the fastest sorting methods and uses less memory size. Odd-even sort algorithm was presented and shown to be efficient by Habermann in 1972 [33].

Prox map sort algorithm was invented by Prof. Thomas A. Standish at the University of California in the year 1980 [34]. Prox map sort algorithm is a sorting algorithm that works by partitioning an array of data items into a sub arrays. The name is sort for computing a "proximity map". Comb sort was discovered by Wlodzimierz Dobosiewicz in 1980 [35]. Later in 1991 it was redesigned by Stephen Lacey and Richard Box.

Smooth sort was invented by EDSGER DIJKSTRAS in 1981[36]. Smooth sort is a sorting algorithm which is similar to heap sort we use this sorting algorithm in order to sort the list of elements which are in unsorted order. Bogo sort also called as permutation sort, stupid sort, slow sort, shotgun sort, monkey sort. Bogo sort was designed in 1984[37].This algorithm was based on monkey theorem. It is a highly ineffective sorting algorithm based on the test paradigm.

Binary tree sort was implemented by P.F.Windley, A.D.Booth, A.J.T.Colin and T.N.Hibbard in 1995. The splay tree was discovered by Daniel Sleator and Robert Tarjan in the year 1985. Some of the experiments on this splay sort by Moffat, Eddy and Peterson was done in the year 1996 [38]. Saikkonen et al., modify splay sort in the year 2012 to be more strongly adaptive to the number of contiguous monotone subsequences in the input, and report on experiments showing that the resulting algorithm is faster on inputs that are nearly presorted according to this measure.

Intro sort was invented by David Musser in Musser in the year 1997, in which he also introduced intro select, a hybrid selection algorithm based on quick select. Flash Sort's work was published in the year 1998 by Karl-Dietrich Neubert. It is a distributing sorting algorithm. It shows the linear computational complexity O(n) for uniformly distributed data sets and it requires a little. Gnome Sort is was introduced in the year 2000 which is based on the technique used by the standard Dutch Garden Gnome [39].

Tim sort is a hybrid stable algorithm uses insertion and merging methods. Tim sorting method was implemented by Tim Peters in 2002 based on the techniques peter mcllroy's "Optimistic Sorting and Information Theoretic Complexity". Bead sort is a natural sorting algorithm and is also known as gravity sort, developed by three mathematicians Joshua J. Arulanandham et al., from the University of Auckland, New Zealand in 2002. Michael A.Bender, Martin Farach-Colton, and Miguel Mosterio proposed Library Sort in 2004 and was published in 2006.

In 2008, Pok-Son Kim and Ame Kutzner proposed Block sort. Block sort, or block merge sort, is a sorting algorithm that combining at least two merges

operations with an insertion operation.

Novel sort was proposed by Srinivas et al., in 2013 [40]. In novel sort, the smaller element is moved one or two positions towards left and in bubble sort moves only one position in either direction. Enhanced Novel Sorting algorithm is an extension for novel sorting algorithm [41]. The main drawback of novel sort is that if the smallest element in the array is present at last location then n/2 iterations are required to move that element to the first location. In order to overcome this drawback R. Srinivas, the author of novel sorting algorithm proposed a modification in the same year.

Schematize sorting was developed by R. Srinivas et al., in the year 2016[42]. Schematize sorting uses only assignments and comparison operations to sort the given elements. This sorting algorithm is efficient when compared with bubble, novel, insertion and selection sorting methods. In 2017 Min-Max Sorting is proposed by Srinivas et al. This algorithm is based on linear search [43].

III. SORTING BY ELEMENT POSITION COUNT ALGORITHM

As the name indicates, this algorithm calculate position count for each element in phase 1 and in phase 2 all the elements are arrange based on the position count.

Algorithm:

Phase 1:

- ✓ Starting from the first element, calculate the position count of each element by comparing with other elements in the list accordingly increment the count value of a position counter corresponding to the element.

Phase 2:

- ✓ After counting position count value for each element arrange elements according to its count value.

Example: Consider 7 5 6 2 1 9 3 8 4

In this example, consider PCount to count position count Value that is based on number elements greater than selected element.

Phase 1: Starting from the first element calculate the count values for every element,

Step 0:

Given elements and corresponding index and PCount values

Elements: 7 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : 0 0 0 0 0 0 0 0 0

Step 1:

Select first element from the list i.e. 7 compare it with other elements and count number of elements greater than that element.

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2** 0 0 0 0 0 0 0 0

Step 2:

Select second element from the list i.e. 5 compare it with other elements and count number of elements greater than that element.

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4** 0 0 0 0 0 0 0

Step 3:

Select third element from the list i.e. 6 compare it with other elements and count number of elements greater than that element.

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3** 0 0 0 0 0 0

Repeat above procedure for other elements 2 to 4. In final step 9 the count will be

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**

Phase 2: Arrange elements according to their PCount value. This arrangement can be done in two ways, first method is by using additional memory. This approach is very simple to arrange elements. If we want to arrange elements in descending order PCount gives index value. i.e. first element count is 2 so it is placed in third position, second element count is 4 so it is placed in fifth location and element 9 count is 0 so that it will be placed in first position.

This procedure is repeated for all other elements.

Step 1:
Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : **7**

Step 2:
Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : **7 5**
….
Step 9:
Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 9 8 7 6 5 4 3 2 1

To arrange the elements in ascending order, elements are placed in their positions by evaluating their index as **number of elements – Pcount - 1**. i.e. **N-PCount-1**, where N is number of elements.

First element PCount is 2 and its index value is 9-2-1= 6. So the elements is placed in position 6

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 **6** 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 7

Second elements count is 4 and its index is 9-4-1=4.
Elements: **7 5** 6 2 1 9 3 8 4
Index : 0 1 2 3 **4** 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 5 7

The same procedure is repeated for all other elements.
Elements:**7 5** 6 2 1 9 3 8 4
Index : 0 1 2 3 **4** 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 1 2 3 4 5 6 7 8 9

In second method no additional memory is required to arrange the elements. To arrange the elements in order, the largest element is considered first and placed in first position but before placing it in first position it is temporarily stored in memory. In second step first element position value is used to place it in correct order and the element which is there is stored in temporary memory before placing it in order. The element stored in temporary memory is considered next. This procedure is considered for N times.

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**

In step 1 element 9 is considered and before placing it in index 0 corresponding element 7 is stored in temporary memory and element 9 stored in its position. Next element to be considered is 7 and its index is 2. The element stored in index position 2 is 6 and it is stored in temporary location before placing element 7. The same procedure is repeated for other elements.

Step1:

Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 9
Temp1= 7

Step2:
Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist : 9 7
Temp2: 6

Step 3:
Elements: **7** 5 6 2 1 9 3 8 4
Index : 0 1 2 3 4 5 6 7 8
PCount : **2 4 3 7 8 0 6 1 5**
Newlist :9 7 6
Temp1=2
After step 9, the elements will be
Newlist : 9 8 7 6 5 4 3 2 1

In the above example, algorithm evaluates only the number of element greater than the selected element. The same procedure can also be repeated by counting number of elements less than the selected element. If there are duplicate elements in the list then separate counter is maintained for counting the number of occurrences of that element.

IV. RESULTS

Time Complexity

The time complexity of this algorithm is $O(n^2)$. Where n is size of the list. The Best, Worst and Average cases have the same time complexity.

This method requires maximum n * (n-1) comparisons i.e. there are n number of iterations and in each iteration there are n-1 comparisons.

In each iterations there maybe maximum of n-1 increment operations. Therefore total number of increments maximum of n*(n-1).

Space Complexity

The space complexity is O(n). The memory required is based on method adapted to place elements in proper order.

V. CONCLUSION

The sorting algorithm proposed in this paper is very simple to understand and implement. The execution efficiency of this algorithm is relatively good compared to bogo sort, permutation sort and bubble sort.

VI. REFERENCES

[1]. Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani & Nabeel Imhammed Zanoon, "Review on Sorting Algorithms A Comparative Study", International Journal of Computer Science and Security (IJCSS), Volume (7) : Issue(3) : 2013

[2]. T. Cormen, C.Leiserson, R. Rivest and C.Stein, "Introduction To Algorithms", McGraw-Hill, Third Edition, 2009,pp. 15-17.

[3]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to Algorithms", Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7

[4]. M. Goodrich and R. Tamassia, "Data Structures and Algorithms in Java", John wiley & sons 4th edition, 2010, pp.241-243.

[5]. Neubert, Karl-Dietrich. "The Flash sort Algorithm". Dr. Dobb's Journal: 123. Retrieved 2007-11-06

[6]. Aditya Dev Mishra and Deepak Garg. "Selection of Best Sorting Algorithm ", International journal of intelligent information Processing. II (II).pp. 363-368,DEC, 2008.

[7]. Hoare, C. A. R. "Algorithm 64: Quicksort". Comm. ACM. 4 (7): 321. doi:10.1145/366622.366644, 1961

[8]. Donald Knuth. "The Art of Computer Programming, Volume 3: Sorting and Searching", Third Edition. Addison–Wesley, 1997. ISBN 0-201-89685-0. Pages 138–141 of Section 5.2.3

[9]. Simon Singh "Flipping pancakes with mathematics". The Guardian. Retrieved March 25, 2014.

[10]. Arulanandham, J. J.; Calude, C. S.; and Dinneen, M. J. "Bead-Sort: a Natural Sorting Algorithm." Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, No. 76, 153-162, 2002.

[11]. Shell, D. L. (1959). "A High-Speed Sorting Procedure" (PDF). Communications of the ACM. 2 (7): 30– 32. doi:10.1145/368370.368387

[12]. Phillips, Malcolm. "Array Sorting". Retrieved 3 August 2011.

[13]. Brejova, B. "Analyzing variants of Shell sort". Inform. Process. Lett. 79 (5): 223–227. doi:10.1016/ S0020-0190(00) 00223-4 , Sep, 2001.

[14]. Gonnet, Gaston. "Optimal Binary Search Trees". Scientific Computation. ETH Zürich. Retrieved 1 December 2013.

[15]. Paul E. Black. "gnome sort". Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology. Retrieved 2011-08-20.

[16]. Bender, M. A.; Farach-Colton, M.; Mosteiro M.. "Insertion Sort is O(n log n)". Theory of Computing Systems. 39 (3):391. doi:10.1007/s00224-005-1237-z,2006.

[17]. Kutzner, Arne; Kim, Pok-Son, " Ratio Based Stable In-Place Merging Lecture Notes in Computer Science", Springer Berlin Heidelberg. pp. 246–257, 2008.

[18]. Mallows, C. L. "Patience sorting". Bull. Inst. Math. Appl. 9: 216–224,1973

[19]. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford, "8.2 Counting Sort", Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, pp. 168–170, ISBN 0-262-03293-7,2001.

[20]. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford "Introduction to Algorithms ", (3rd ed.). MIT Press and McGraw-Hill. pp. 655–657. ISBN 0-262-03384-4.

[21]. Andersson, Arne, and Stefan Nilsson. "A new efficient radix sort." Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on. IEEE, 1994.

[22]. Kumar, Akash, Akshay Dutt, and Gautam Saini. "Merge Sort Algorithm." International Journal of Research 1.11 (2014): 16-21.

[23]. Grabowski, Franciszek, and Dominik Strzalka. "Dynamic behavior of simple insertion sort algorithm." Fundamenta Informaticae 72.1-3 (2006): 155-165.

[24]. Bowers, K. J. "Accelerating a particle-in-cell simulation using a hybrid counting sort." Journal of Computational Physics 173.2 (2001): 393-411.

[25]. Astrachan, Owen. "Bubble sort: an archaeological algorithmic analysis." ACM SIGCSE Bulletin. Vol. 35. No. 1. ACM, 2003.

[26]. Goodrich, Michael T. "Randomized shellsort: A simple oblivious sorting algorithm." Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010.

[27]. Sedgewick, Robert. "Implementing quicksort programs." Communications of the ACM 21.10 (1978): 847-857.

[28]. Jadoon, Sultanullah, Salman Faiz Solehria, and Mubashir Qayum. "Optimized selection sort algorithm is faster than insertion sort algorithm: a comparative study." International

Journal of Electrical & Computer Sciences IJECS-IJENS 11.02 (2011): 19-24.

[29]. Yap, Chee K. "New upper bounds for selection." Communications of the ACM 19.9 (1976): 501-508.

[30]. Ebell, Mark H., et al. "Strength of recommendation taxonomy (SORT): a patient-centered approach to grading evidence in the medical literature." The Journal of the American Board of Family Practice 17.1 (2004): 59-67.

[31]. Sharma, Vandana, Satwinder Singh, and K. S. Kahlon. "Performance Study of Improved Heap Sort Algorithm and Other Sorting Algorithms on Different Platforms." IJCSNS 8.4 (2008): 101.

[32]. Nassimi, David, and Sartaj Sahni. "Bitonic sort on a mesh-connected parallel computer." IEEE Transactions on Computers 1 (1979): 2-7.

[33]. Zhang, Wang, et al. "Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip." Global Congress on Intelligent Systems. IEEE, 2009.

[34]. Mackaness, William A. "An algorithm for conflict identification and feature displacement in automated map generalization." Cartography and Geographic Information Systems 21.4 (1994): 219-232.

[35]. Pu, Yuliang, et al. "An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl." Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on. IEEE, 2015.

[36]. Kelley, C. T. "Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition." SIAM journal on optimization 10.1 (1999): 43-55.

[37]. Taherkhani, Ahmad, Ari Korhonen, and Lauri Malmi. "Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course." Proceedings of the 12th Koli Calling International Conference on Computing Education Research. ACM, 2012.

[38]. Weiss, Mark Allen, and Susan Hartman. Data structures and problem solving using Java. Vol. 204. Boston, MA: Addison-Wesley, 1998.

[39]. Meolic, R. . Demonstration of Sorting Algorithms on Mobile Platforms. In CSEDU ,2013, pp. 136-141.

[40]. Srinivas, R., and A. Raga Deepthi. "Novel Sorting Algorithm", International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397, Vol. 5 No. , 01 Jan 2013

[41]. Srinivas, R. "Enhanced Novel Sorting Algorithm." Global Journal of Computer Science and Technology, 2013.

[42]. Srinivas, R., Shaik Vahid, and K. A. Sireesha. "Schematize Sorting." I J C T A, 9(6), 2016, pp. 3353-335.

[43]. Srinivas, R., K. A. Sireesha and Shaik Vahid,. "Min-Max Sorting" , 2017.