2012 4th International Conference on Computer Engineering and Technology (ICCET 2012)
IPCSIT vol.40 (2012) © (2012) IACSIT Press, Singapore

Relative Split and Concatenate Sort V2 (RSCS-V2)

Abdul Wahab Muzaffar[1], Hina Anwar[2], Farooque Azam[3]

[1, 2,3]National University of Science and Technology, H-12, Islamabad, Pakistan

Abstract. After the emergence of computer systems; the number and objects needs to be arranged in a particular order either ascending or descending orders. The ordering of these numbers is generally referred to as sorting. Studies showed that more than 50% of computing is based on sorting. Sorting has many applications in computer systems, file management, memory management. Sorting algorithm is an algorithm by which elements are arranged in a particular order following some characteristic or law. Sorting is very important from the dawn of the computing till now and many sorting algorithms have been proposed with different time and space complexities. In this research authors develop a new sorting technique to keep in view the existing techniques. Authors also proposed the algorithm i.e. Relative Split and Concatenate Sort V2, implements the algorithm and then compared results with some of the existing sorting algorithms. It is discovered that the algorithms proposed in this research is relatively simpler and efficient than some of the existing well known sorting algorithms i.e. bubble sort, insertion sort and selection sort. A simulator is designed to compare the Relative Split and Concatenate Sort V2 algorithm with other sorting algorithms. Simulation results have been summarized as graphs with number of elements on x-axis and time taken in milliseconds on the y-axis. Relative Split and Concatenate Sort V2 (RSCS-V2) algorithm is a successor to Relative Split and Concatenate Sort V1 (RSCS-V1) that is already presented in a conference. RSCS-V2 has much better results than RSCS-V1.

Keywords: New Sorting, Time Complexity, RSCS, Sorting Algorithm.

1. Introduction

Sorting is defined in English Language Dictionary [1] as “Sorting is a process by which the sedimentary particles become separated by some particular characteristic”. In computer science sorting is mainly used to arrange the numbers or objects in a specific order which follows some characteristics or law. The ordering is either ascending or descending as per the requirement of the problem. There is no one sorting algorithm that is best for each and every situation. Donald Knuth in [2], reports that “computer manufacturers of the 1960s estimated that more than 25 percent of the running time on their computers was spend on sorting, when all their customers were taken into account".

Generally sorting falls into two categories [3]: first category is ordering which places elements of same kind in particular sequence based on their properties and the second category is categorizing that places elements in same group or under same label based on their properties.

Sorting algorithms fall into two classes with respect to time complexity i.e. $O(n^2)$ and O(nlogn). $O(n^2)$ algorithms work in iterations, where as those with O(nlogn) time complexity are more effective and their efficiency is far better than $O(n^2)$ algorithms. O(nlogn) algorithms are divide-and-conquer in nature while works recursively. O(nlogn) sorting algorithms are: merge sort proposed by Von Neumann in 1945, Shell’s sort in 1959, and quick sort by Hoare in 1962 [4]. The ultimate intention of so much sorting techniques is the cost and complexity reduction of the algorithms [5]

2. Literature review

In this section a review of existing sorting techniques, history of formation methodologies as well as algorithms are presented. The chapter also discusses the applications and limitations of sorting algorithms.

2.1. Categories of Sorting

Sorting is broadly categorized into two major categories: Internal sorting and external sorting [6]

Internal Sorting: This sorting category is called internal as the whole sorting process takes place in the main memory, as data to be sorted is small enough to be fit into the main memory. Bubble Sort, Cocktail Sort, Insertion Sort, Shell Sort, Selection Sort, and Quick Sort are some well known sorting algorithms lie under this category [7].

External Sorting: This sorting category is used when the data being sorted is in large amount and doesn't fit into the main memory. Merge sort and Heap Sort come under this category of sorting [8].

2.2. Taxonomy of Sorting Algorithms

"Taxonomy is the practice and science of classification. The word finds its roots in the Greek taxis (meaning 'order', 'arrangement') and νόμος, nomos ('law' or 'science')" [9]. There are multiple taxonomies of sorting algorithms. Knuth proposed a sorting taxonomy by dividing the sorting algorithms under three categories [10]: 1) Insertion, 2) Selection and 3) Exchange.

Their well known examples are simple Insertion Sort, Selection Sort and exchange sort respectively. In Knuth's introduction to sorting he describes: 1) Insertion sort, which takes the item one at a time, and each new item, is inserted into its proper position by comparing it with the previously sorted items. 2) Exchange sort, in which if two elements are found to be out of order, they are interchanged. This process is repeated until no more exchanges are needed. 3) Selection sort, in which the smallest item is located and somehow separated from the rest: the next smallest is then selected, and so on. Taxonomy presented by Green and Barstow in [11]. This taxonomy suggested by SUSAN M. MERRITT [12] divides the sort into two categories: 1) easysplit/hardjoin and 2) hardsplit/easyjoin.

2.3. Existing Sorting Algorithms

A number of sorting algorithms are currently used in the field of computer science. This section will briefly discuss some of the trendy sorting techniques among them. These are following:

2.3.1 Bubble Sort:

Bubble sort is said to be first sorting algorithm and so pioneer in sorting. Bubble sort is a sequential sorting algorithm, it sort the items in passes. In each pass one value is moved to the left and this will be the least value during the pass [13]. Main disadvantage of bubble sort is that it takes n^2 comparisons then the length of the list Complexity of bubble sort for average case and worst case is $O(n^2)$. When we have a sorted list and apply bubble sort it shows a behavior of O(n), showing its best case complexity [5].

3.3.2 Cocktail Sort:

Cocktail sort is also based on the same methodology as bubble sort, it is a variation of bubble sort that is both a stable sorting algorithm and a comparison sort [14]. The average and the worst case complexity of cocktail sort is equal to bubble sort i.e. $O(n^2)$[5].

2.3.3 Comb Sort:

Comb sort is a relatively simplistic sorting algorithm originally designed by Wlodek Dobosiewicz in 1980 and later rediscovered and popularized by Stephen Lacey and Richard Box, who described it in Byte Magazine in April 1991 [15]. It also belongs to the family 'Exchange Sort". Combo sort improves on bubble sort, and rivals in speed more complex algorithms like Quicksort

2.3.4 Heap Sort:

Heap sort is also part of the selection sort family. Although it is somewhat slower in practice on most machines than a good execution of quick sort but it has the advantage of a worst-case Θ (n log n) runtime [16]. Basic implementations require two arrays - one to hold the heap and the other to hold the sorted elements.

2.3.5 Selection sort

Selection sort is another renowned sorting algorithm. Its main disadvantage is that it is inefficient for large lists, its performance is worst than insertion sort for large number of items. It takes 'n' number of passes for a list of length 'n' [17]. Complexity of selection sort for average case and worst case is $O(n^2)$. Selection sort is more advantages in terms of memory as it takes less memory.

2.3.6 Insertion Sort:

Insertion sort is another sorting that is very simple, efficient and well known technique. Its disadvantage is utilization of more memory as compared to bubble sort and selection sort; also it becomes very slow while list gets larger [18]. Complexity of selection sort for average case and worst case is $O(n^2)$, while for the best case it shows a behavior of O(n). The insertion sort algorithm is a very slow algorithm when list is very large [18].

2.3.7 Merge Sort:

Another algorithm, based on O(nlogn) category or divide-and-conquer principle, is merge sort. It was proposed by John von Neumann in 1945 [19]. Its average case and worst case complexity is O(nlogn), and shows a behavior of O(logn) in the best case [19].

2.3.8 Quick Sort:

Quick sort is fastest among the sorting algorithms proposed by Von Neumann in 1962 [5]. Despite its slow worst case, quick sort is best practical choice. Complexity of quick sort for worst case is $O(n^2)$, for the best case it shows a behavior of O(log n) and average case is Θ(nlogn) [20].

2.3.9 Shell Sort:

It is also known as the generalized form of the insertion sort as the elements by this sort takes longer jumps to get their original positions. The worst case complexity of the algorithm is O(n2) [21,22]. It got its name after its presenter, Donald Shell.

3.4 Research methodology

Authors in this research paper propose a new sorting algorithm and implement it in a high level language and compared with some of the well known existing sorting techniques i.e. bubble sort, cocktail sort, insertion sort, selection sort, and quick sort. The final analysis of the paper is in the form of graphs showing the running time comparison of Relative Split and Concatenate sort and existing sorting algorithms.

3. Proposed algorithm

3.1. Algorithm1*(RSCS-V2)-Steps:*

The Steps of the proposed algorithm are as follows:

1. Divide the list into 3 sub-lists.
2. Take average of each of the sub-list.
3. Sort the three averages and named as large, medium and small.
4. Compare the 1st element of the list with each average.
5. If the element is less than (<) small average, put it in a new list of smaller items, after finding the exact location of the item using binary search.
6. Else if the element is greater than (>) large average, put it in a new list of larger elements, after finding the exact location of the item using binary search.
7. Else put it in a new list of medium elements, after finding the exact location of the item using binary search.
8. Take next element, if it is smaller than the smaller average, compare it with the elements already presented in the smaller array, and put it at its exact location.
9. Else if it is larger than the larger average, compare it with the elements already presented in the larger array, and put it at its exact location.
10. Else compare it with the elements in the medium array and put it at its exact location.
11. Repeat these steps for the whole list.

3.2. Algorithm1*(RSCS-V2):Running Cost Analysis*

The main structure of the algorithm depicts that there is an outer main loop within which there lies another loop. The outer loop will run n number of times, of the elements of the list and inner loop will make its way n times in worst case analysis, i.e. if the whole list to be sorted is in reverse order (descending while ascending is needed or ascending when descending is needed).

The length of the array: n, Outer Loop runs: n, Inner Loop runs: n

Comparison Statements: c

So, Total Time: n*n-1*c, Total Time: n*n-1*c, Ignoring Constants we will get; Total Time: $n*n=n^2$

By keen observing it the worst case running cost of algorithm is calculated to be O(n2). The behavior of the algorithm in the best case will be O(n), depicting that the elements in the list are in sorted form (descending or ascending whatever needed). Similarly the average case of the running cost will be O(n2) depending upon the elements in the list.

4. Comparison with existing sorting algorithms

4.1. Proposed Algorithm: Relative Split and Concatenate Sort (RSCS V-2)

Relative Split and Concatenate sort is implemented in C# .NET and compared with the algorithms lie under the category of $O(n^2)$ running time complexity i.e. bubble sort, cocktail sort, insertion sort, selection sort. For each comparison, lists of different sizes were generated and sorted. The sizes were 5000, 10000, 20000, 50000, 80000 and 100000. Minimum number was kept zero and the maximum was kept 10000 always. Following are the results of these experiments. Graphical as well as textual description of the results is presented for convenience. Input list is generated randomly and the experiments were performed on a system with following specifications:

Processor 2.0 GHz RAM 256MB

RSCS-V2 was proposed after some changes in the RSCS-V1, a new algorithm was proposed which shows some better results. Following are the results:

1. **Comparison with Bubble Sort**

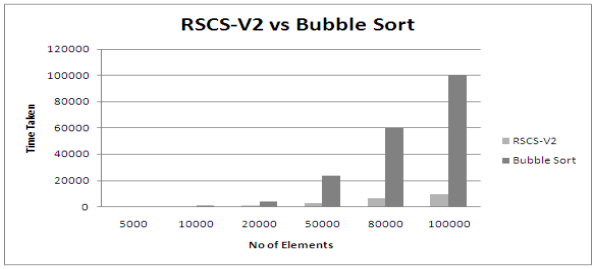

Figure 1: RSCS-V2 vs Bubble Sort

In the above graph, at x-axis we have placed number of elements in the list to be sorted and at y-axis we have placed the time taken by program for execution in milliseconds. It can be seen clearly that Relative Split and Concatenate sort (RSCS-V2) shows much better performance than Bubble sort that is obvious from the graph.

2. **Comparison with Cocktail Sort**

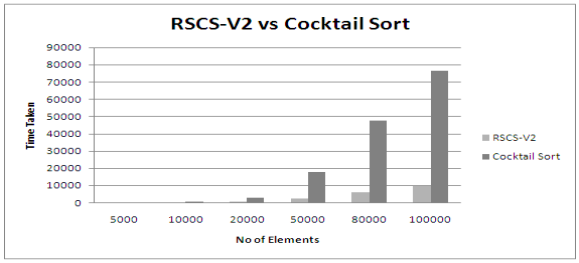

Figure 2: RSCS-V2 vs Cocktail Sort

The above graph is same as that of bubble sort that at x-axis are the numbers of items and along y-axis are the execution times in milliseconds. From the above graph it is depicted that Relative Split and Concatenate sort (RSCS-V2) shows clearly efficiency than Cocktail sort.

3. **Comparison with Selection Sort**

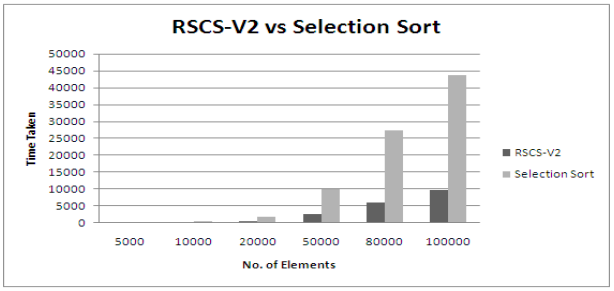

Figure 3: RSCS-V2 vs Selection Sort

Above graph depicts the performance difference of Selection and the Relative Split and Concatenate sort (RSCS-V2). Along x-axis is the number of items in the input list while along y-axis execution times. Relative Split and Concatenate sort (RSCS-V2) shows a clear domination over Selection sort.

4. Comparison with Insertion Sort

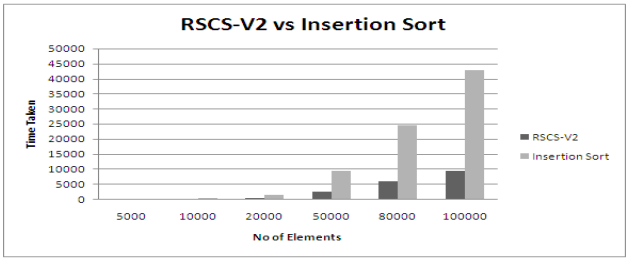

Figure 4: RSCS-V2 vs Insertion Sort

Above graph depicts the performance difference of Insertion and the Relative Split and Concatenate sort (RSCS-V2). Along x-axis is the number of items in the input list while along y-axis is the execution times. Relative Split and Concatenate sort (RSCS-V2) is clearly efficient than Insertion sort.

5. Conclusion

Proposed Relative Split and Concatenate Algorithms (RSCS-V2) is an $O(n^2)$ algorithm with efficient results than previously proposed RSCS-V1. Simulation studies were carried out to test the usability of the new developed algorithms. Results of implementation graphs of the proposed algorithm shows that it is efficient, in running time, over other $O(n^2)$ category algorithms i.e. bubble sort, cock tail sort, insertion sort, selection sort, and RSCS-V1. This algorithm takes more memory and trade of the time and space general rule. Summarizing the whole discussion, results clear that the proposed algorithm above than the middle orders algorithms. Comparison of proposed algorithms with n^2 algorithms shows that it is almost among the best n^2 algorithms.

6. References

[1] Sorting (2009), Sorting. http://dictionary.reference.com/browse/sorting, Accessed October 25, 2009.

[2] Philippas T., Yi Z. (2003), "A Simple, Fast Parallel Implementation of Quicksort and its Performance, Evaluation on *SUN* Enterprise 10000". IEEE- Euro micro Conference on Parallel, Distributed and Network-Based Processing (Euro-PDP'03). ISBN: 0-7695-1875-3/03

[3] Knuth D. (1997) "The Art of Computer Programming, Volume 3: Sorting and Searching'', Third Edition. *Addison*-Wesley, 1997. ISBN 0-201-89685-0. pp. 138–141, of Section 5.2.3: Sorting by Selection.

[4] Alexandros Agapitos and Simon M. Lucas, "Evolving Efficient Recursive Sorting Algorithms", 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, 2006.

[5] Iqbal Z., Gull H. and Muzaffar A. W. (2009) "A New Friends Sort Algorithm". 2nd IEEE International Conference on Software Engineering and Information Technology, ISBN 978-1-4244-4520-2, pp 326-329.

[6] Knuth D. (1997) "The Art of Computer Programming, Volume 3: Sorting and Searching'', Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0.

[7] Internal_sort, (2009), Internal Sort. http://en.wikipedia.org/wiki/Internal_sort, Accessed October 25, 2009.

[8] External_sorting, (2009), External Sorting http://en.wikipedia.org/wiki/External_sorting, Accessed October 25, 2009

[9] Taxonomy, (2010), Taxonomy. http://en.wikipedia.org/wiki/Taxonomy, Accessed January 10, 2010.

[10] Knuth. D.E. The Art of Computer Programming. Vol. 3, Sorting and Searching. Addison-Wesley, Reading, Mass., 1973.

[11] Green, C., Barstow. D. On program synthesis knowledge. Artif. Infd. 10 (1978). pp. 241-279.

[12] MERRITT S. M. (1985), "An inverted taxonomy of Sorting Algorithms. Programming Techniques and Data Structures", Communications of ACM, Vol. 28, Number 1, ACM.

[13] James D. Fix and Richard E. Ladner, "Sorting by Parallel Insertion on a One-Dimensional Subbus Array", IEEE TRANSACTIONS ON COMPUTERS, VOL. 47, NO. 11, NOVEMBER 1998.

[14] Cocktail_Sort(2009), Cocktail Sort, http://en.wikipedia.org/wiki/Cocktail_sort Accessed December 25, 2009.

[15] Comb_sort (2009), Comb Sort. http://en.wikipedia.org/wiki/Comb_sort, Accessed October 25, 2009.

[16] Heap_sort (2009), HeapSort. http://en.wikipedia.org/wiki/Heap_sort, Accessed October 25, 2009.

[17] Seymour Lipschutz. Theory and Problems of Data Structures,Schaum's Outline Series: International Edition, McGraw-Hill, 1986. ISBN 0-07-099130-8., pp. 324–325, of Section 9.4:Selection Sort..

[18] Seymour Lipschutz. Theory and Problems of Data Structures,Schaum's Outline Series: International Edition, McGraw-Hill, 1986. ISBN 0-07-099130-8., pp. 322–323, of Section 9.3:Insertion Sort.

[19] Cormen T. H, Leiserson C. E., Rivest R. L. and Stein C. [1990] (2001). "Introduction to Algorithms", 2nd edition, MIT Press and McGraw-Hill, ISBN 0-262-03293-7, pp. 27–37. Section 2.3: Designing algorithms.

[20] 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. Section 7.1:Quicksort, pp.145–149.

[21] Shahzad B., Afzal M. T (2007) "Enhanced Shell Sorting Algorithm", World Academy of Sciences, Journal, Volume 27.

[22] Shell _sort, (2009), Shell Sort. http://en.wikipedia.org/wiki/Shell_sort, Accessed January 12, 2010.