International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

Enhanced Shell Sorting Algorithm

Basit Shahzad, and Muhammad Tanvir Afzal

***Abstract*—**Many algorithms are available for sorting the unordered elements. Most important of them are Bubble sort, Heap sort, Insertion sort and Shell sort. These algorithms have their own pros and cons. Shell Sort which is an enhanced version of insertion sort, reduces the number of swaps of the elements being sorted to minimize the complexity and time as compared to insertion sort. Shell sort improves the efficiency of insertion sort by *quickly* shifting values to their destination. Average sort time is $O(n^{1.25})$, while worst-case time is $O(n^{1.5})$. It performs certain iterations. In each iteration it swaps some elements of the array in such a way that in last iteration when the value of h is one, the number of swaps will be reduced. Donald L. Shell invented a formula to calculate the value of 'h'. this work focuses to identify some improvement in the conventional Shell sort algorithm. "Enhanced Shell Sort algorithm" is an improvement in the algorithm to calculate the value of 'h'. It has been observed that by applying this algorithm, number of swaps can be reduced up to 60 percent as compared to the existing algorithm. In some other cases this enhancement was found faster than the existing algorithms available.

***Keywords*—**Algorithm, Computation, Shell, Sorting.

I. Introduction

SORTING has been a profound area for the algorithmic researchers. And many resources are invested to suggest a more working sorting algorithm. For this purpose many existing sorting algorithms were observed in terms of the efficiency of the algorithmic complexity. Shell sort and insertion sort algorithms were observed to be both economical and efficient [1,2]

The comparison of "Enhanced Shell sort" with Insertion sort and Shell sort is made. In Shell sort the numbers of swaps are reduced as compared to Insertion sort and in "Enhanced Shell Sort" the numbers of swaps are further reduced as compared to Shell sort.

A. Insertion Sort

The insertion sort, as its name suggests, inserts each item into its proper place in the final list. The simplest implementation of this requires two list structures: the source list and the list into which sorted items are inserted. To save memory, most implementations use an in-place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place.

Like the bubble sort, the insertion sort has a complexity of $O(n^2)$. Although it has the same complexity, the insertion sort is a little over twice as efficient as the bubble sort.

In the following example, it is calculated that how many swaps are required in Insertion sort. Following is the list of 38 elements the Insertion sort algorithm is applied, in order to see that how much swaps are required for this algorithm to bring the elements in order.

20,10,51,92,25,57,48,37,12,86,33,1,113,1,2,228,27,82,60, 100,12,52,3,1,85,65,14,41,71,17,25,62,14,2,0,83,49,32

When Insertion Sort was applied on the given array then number of swaps calculated for it are 367.

B. Shell Sort

The first diminishing increment sort. On each pass, 'i' sets of n/i items are sorted, typically with insertion sort. On each succeeding pass, i is reduced until it is 1 for the last pass. A good series of i values is important to efficiency [1].

Invented by Donald Shell in 1959, the shell sort is the most efficient of the $O(n^2)$ class of sorting algorithms [4]. Of course, the shell sort is also the most complex of the $O(n^2)$ algorithms.

The shell sort is a "diminishing increment sort", better known as a "comb sort" to the unwashed programming masses. The algorithm makes multiple passes through the list, and each time sorts a number of equally sized sets using the insertion sort [5]. The size of the set to be sorted gets larger with each pass through the list, until the set consists of the entire list. (Note that as the size of the set increases, the number of sets to be sorted decreases.) This sets the insertion sort up for an almost-best case run each iteration with a complexity that approaches O(n) [9,10].

The items contained in each set are not contiguous, rather , if there are i sets then a set is composed of every i-th element. For example, if there are 3 sets then the first set would contain the elements located at positions 1, 4, 7 and so on. The second set would contain the elements located at positions 2, 5, 8, and so on; while the third set would contain the items located at positions 3, 6, 9, and so on.[6]

The size of the sets used for each iteration has a major impact on the efficiency of the sort. The algorithm provides efficient execution for medium-size lists. Along with the benefit of being robust, the algorithm is considered to be somewhat complex and not nearly as

Manuscript received October 9, 2001.

Basit Shahzad is senior Lecturer at the COMSATS Institute of Information Technology, Islamabad-Pakistan where he is working the software and algorithms research group. He has published several research papers and has keen interest in the area of algorithms (e-mail: Basit_shahzad@comsats.edu.pk).

Tanvir Afzal is undertaking his PhD research in the Institute of Information Systems and Computer Media at the Graz University of Technology, Austria (e-mail: mafzal@iicm.edu).

efficient as the merge, heap, and quick sorts are [2].

It has been observed that Shell sort is a non-stable in-place sort. Shell sort improves on the efficiency of insertion sort by quickly shifting values to their destination. Average sort time is $O(n^{1.25})$, while worst-case time is $O(n^{1.5})$.

Various spacing may be used to implement the shell sort. Typically, the array is sorted with large spacing, then the spacing is reduced, and the array is sorted again. On the final sort, spacing is one. Although the shell sort is easy to comprehend, formal analysis is difficult. In particular, optimal spacing values elude theoreticians. Knuth has experimented with several values and recommends that spacing 'h' for an array of size N be based on the following formula:

Let h1 = 1, hs+1 = 3hs + 1, and stop with ht when ht+2 ≥ N. Thus, values of h are computed as follows:

h1 = 1
h2 = (3 x 1) + 1 = 4
h3 = (3 x 4) + 1 = 13
h4 = (3 x 13) + 1 = 40
h5 = (3 x 40) + 1 = 121

To sort 100 items we first find an 'hs' such that hs ≥ 100. For 100 items, h5 is selected. The final value (ht) is two steps lower, or h3. Therefore sequence for the values of 'h' will be 13-4-1. Once the initial 'h' value has been determined, subsequent values may be calculated using the formula

hs-1 = floor(hs / 3).

Let's calculate the number of swaps for the same problem by using Shell sort as discussed in Insertion sort.

20,10,51,92,25,57,48,37,12,86,33,1,113,1,2,228,27,82,60,
100,12,52,3,1,85,65,14,41,71,17, 25,62,14,2,0,83,49,32

The algorithmic implementation showed that the series of 'h' is {4,1}, and the swaps required for the above values under Shell sort algorithm are 170.

C. Enhanced Shell Sort Algorithm

Enhanced Shell Sort algorithm works in the same way as existing Shell Sort algorithm. Calculating the value of 'h' is a key step in the execution of shell sort. The value of 'h' in conventional shell sort is determined by the formula:

Let h1 = 1, hs+1 = 3hs + 1, and stop with ht when ht+2 ≥ N.

By using this existing formula the shell sort algorithm reduces the number of swaps up to 50 % as compared to that of Insertion Sort.

Enhanced Shell Sort algorithm focuses to improve the efficiency of the existing algorithm .Efficiency in the existing algorithm can be improved by choosing the appropriate values of 'h'. Selection of the proper value of 'h' is a key point to make it more efficient. Because before comparing all elements of array with each other, it sounds good to arrange elements to some extent so that when the spacing factor is '1' the number of swaps could be reduced maximally [7,8].

Enhance Shell Sort introduces a new mechanism for calculating the value of h. The formula is given below to calculate the first spacing for 'h'.

H= Ceil (n/2).,n is the total number of elements in the array. To calculate the next values of h the following formula is used.

Hs-1=Ceil (hs/2)

For example for 100 elements of array the proposed values of 'h' will be

{50, 25, 13, 7, 4, 2, 1}

But the values of 'h' for standard shell sort algorithm for the same 100 elements are.

{13,4,1}

Now let's take the same example as discussed in Insertion sort and Shell sort to calculate the number of swaps in Enhanced Shell sort algorithm.

20,10,51,92,25,57,48,37,12,86,33,1,113,1,2,228,27,82,60,
100,12,52,3,1,85,65,14,41,71,17, 25,62,14,2,0,83,49,32

Numbers of elements are 38 and now the values of h for Enhanced Shell Sort will be

{20, 10, 5, 3, 2, 1}

In this case, the numbers of swaps are only 85.

II. Comparison of Three Techniques

Now the comparison for the three techniques is made here for the same problem.

TABLE I
COMPARISON

Insertion Sort	Shell Sort	Enhanced Shell Sort
367	170	85

It is apparent that Shell sort reduces the number of swaps up to 50 % as compared to the number of swaps in Insertion sort and Enhanced Shell Sort reduces the number of swaps further up to 50 % as compared to the number of swaps in Shell Sort, thus improving the efficiency of the algorithm.

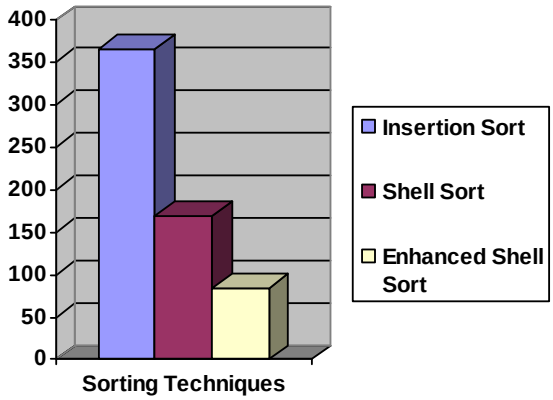

Fig. 1 Comparison of Sorting Techniques

III. Detail Discussion of Results

It is needful that the comparison of execution in terms of numbers of swaps required for each algorithm are made on a wider variety of data, in order to ensure and establish results concretely.

A. For 10 Elements

100,37,12,86,2,127,62,14,3,9
Number of swaps in Insertion sort 30
Number of Swaps in Shell sort 30
Number of swaps in Enhanced Shell sort 12.

B. For 25 Elements

100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200
Number of swaps in Insertion sort 111
Number of Swaps in Shell sort 47
Number of swaps in Enhanced Shell sort 31.

C. For 35 Elements

1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9
Number of swaps in Insertion sort 225
Number of Swaps in Shell sort 109
Number of swaps in Enhanced Shell sort 56.

D. For 65 Elements

12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9
Number of swaps in Insertion sort 917
Number of Swaps in Shell sort 202
Number of swaps in Enhanced Shell sort 105.

E. For 100 Elements

1,41,14,85,2,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9
Number of swaps in Insertion sort 2181
Number of Swaps in Shell sort 756
Number of swaps in Enhanced Shell sort 239

F. For 200 Elements

1,41,14,85,2,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,00,37,12,86,2,127,62,14,3,9,20,30,74,48,37,40,7,101,200,4,900,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9
Number of swaps in Insertion sort 4200
Number of Swaps in Shell sort 1541
Number of swaps in Enhanced Shell sort 471.

G. For 300 Elements

25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,1,41,14,85,2,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63
Number of swaps in Insertion sort 21144
Number of Swaps in Shell sort 1503
Number of swaps in Enhanced Shell sort 928.

H. For500 Elements

25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,1,41,14,85,2,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,41,14,85,2,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,10,52,63,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9,1,2,23,12,25,7,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,14,3,9,30,14,90,1,20,30,74,9,5,36,1,100,37,12,86,2,127,62,14,3,9,30,14,37,12,86,2,127,62,14,3,9,30,14,90,1,20,30,74,48,37,40,7,101,200,4,9

Number of swaps in Insertion sort 35144
Number of Swaps in Shell sort 2703
Number of swaps in Enhanced Shell sort1781.

TABLE II
COMPARISON OF INSERTION, SHELL AND ENHANCED SHELL SORT

CASES	Insertion	Shell	Enhanced Shell Sort
A	30	30	12
B	111	47	31
C	225	109	56
D	917	202	105
E	2181	756	239
F	4200	1541	471
G	21144	1803	928
H	35144	2703	1781

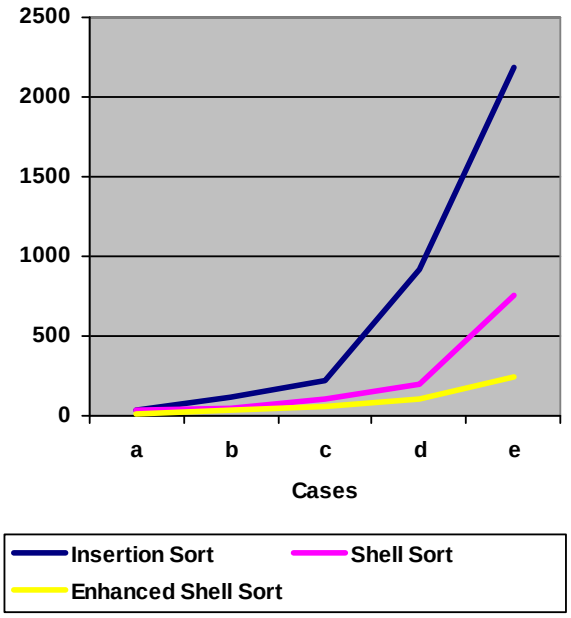

Fig. 2 Comparison Graph

The enhanced shell sorting algorithm is a key towards achieving the excellence in the algorithms to provide the efficient solutions. This has been done by decreasing the swaps required to sort the array of numbers, considerably.

The results show that the new algorithm provides a much efficient way to sort the data and hence causes to save the computational resources. It has been observed that the enhanced shell sorting algorithm can solve the problem in almost 20 times less swaps as compared to insertion sort and in almost half swaps as compared to the shell sorting algorithm. Fig. 3 shows the detailed overview of the number of swaps required to sort different number of elements.

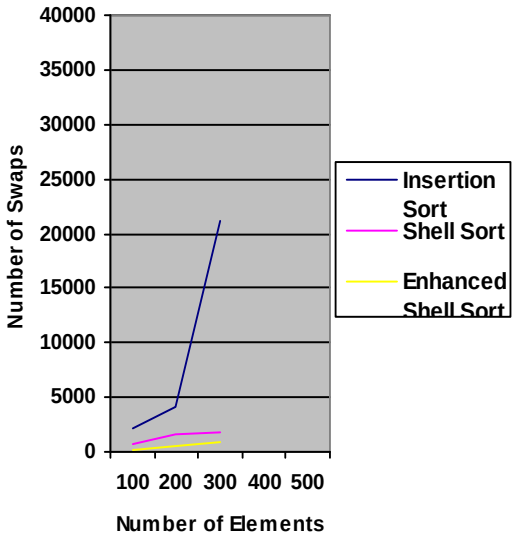

Fig. 3 Elements-Swap Ratio

IV. DISCUSSION

Robert Sedgewick in his paper[11] opens discussion for the performance issues of the Shell sort algorithm and claims that finding a sequence that leads to running times 25% lower than the best known certainly would be of practical interest, Running time can be reduced with the reduction in number of comparisons for the algorithm. We have reduced the number of comparisons up to 60% in some ideal cases but in many cases up to 20%. We executed three algorithms (Insertion sort, Shell sort and Enhanced Shell sort) on same set of data and found some interesting results as can be seen in the comparisons of the algorithms for different cases in Figure 2. The performance of this algorithm would be investigated in future for N= 10^4 or 10^6. But its performance has been proved for N= 10^3 that is useful in normal cases. Some attempts in the past [12][13][14] have shown great concerns about performance. Marcin Ciura [12] shows the results for 128 elements where the data was in sorted form after taking 535.71 swaps but in our case 200 elements have gone through the sorting process from only 471 swaps. These are the standard swaps for 200 elements in our case. They do not change with the change in data but the number of comparisons may vary in certain cases. But overall performance remains better than available performance enhancements techniques.

V. CONCLUSION

This work focuses to provide an enhancement in existing algorithm. Shell sort algorithm gives an average number of comparisons but produces a problem that it does not give least number of swaps. It has been observed that number of swaps produced by Shell Sort can be further reduced. The motivation for reducing the number of swaps is to economically and effectively use the computational resources that are available in terms of processor speed, memory and storage.

Enhanced shell sort algorithm provides a powerful solution

to decrease the number of comparisons as well as number of swaps to a minimum level in shortest possible time, thus decreasing the CPU execution time as well as saving the system memory.

Enhanced shell sort algorithm offers least number of swaps on any size of data. The efficiency and working of this algorithm improves as the size of data grows.

This algorithm has clearly stated simple and easy formula that calculates the values of 'h' in shorter possible time than shell sort algorithm, regardless of number of elements in an array.

This algorithm improves the performance of the existing algorithms up to 60% in some cases. However, in other cases it produced better results in terms of reducing the number of swaps than the existing algorithms. Comparison with existing algorithms is given in the discussion section.

It has been observed that investing on the computer hardware has less significance as compared to investment on the improvement of algorithms in order to improve efficiency. The improved shell sorting algorithms ensure that the number of swaps are reduced in this case and hence provide an efficient way of execution resulting in less computing resources and quick execution.

ACKNOWLEDGMENT

I feel it my principal obligation to thank all my friends who helped me in identifying the need to work in this area. I am also thankful to all those who appreciated this work and provided me a boost to think and to work more effectively in this area of sorting.

REFERENCES

[1] http://www.nist.gov/dada/html/shellsort.htm
[2] http://linux.wku.edu/~lamonml/algor/sort/shell.html
[3] http://oopweb.com/Algorithms/Documents/Sman/VolumeFrames.html?/Algorithms/Documents/Sman/Volume/ShellSort_files/s_shl.htm
[4] Yedidyah Langsam, Moshe J. Augenstan, Aadrew M. Tenenbaum, "Data Structures using C and C++" , 2nd ed, Pearson Education, pp360-366
[5] Larry Nyhoff, "An introduction to Data Structures",2nd ed, pp: 581-585
[6] linux.wku.edu/~lamonml/algor/sort/sort.html
[7] www.cs.hope.edu/alganim/ccaa/shellsort.html
[8] www.inf.fhflensburg.de/lang/algorithmen/sortieren/shell/shellen.htm
[9] www.math.grin.edu/~stone/events/scheme-workshop/shellsort.html
[10] www.cs.odu.edu/~zeil/cs361/Lectures-f02/06sorting/shell/shell.html
[11] Robert Sedgewick, "Analysis of Shellsort and Related Algorithms", Proceedings of the Fourth Annual European Symposium on Algorithms},1996, pp: 1-11
[12] Marcin Ciura "Best Increments for the Average Case of Shellsort",Proceedings of the 13th International Symposium on Fundamentals of Computation Theory,2001, pp: 106—117
[13] Jiang, T., Li, M., Vit´anyi, P.: The average-case complexity of Shellsort. Lecture Notes in Computer Science 1644 (1999), 453–462.
[14] Janson, S., Knuth, D. E.: Shellsort with three increments. Random Structures and Algorithms 10 (1997), 125–142.