Relative Split and Concatenate Sort (RSCS-V1)

Abdul Wahab Muzaffar, Naveed Riaz, Juwaria Shafiq, and Wasi Haider Butt

***Abstract*—Computational problems have significance from the early civilizations. These problems and solutions are used for the study of universe. Numbers and symbols have been used for mathematics, statistics. After the emergence of computers the number and objects needs to be arranged in a particular order either ascending and descending orders. The ordering of these numbers is generally referred to as 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. A number of sorting algorithms have been proposed with different time and space complexities. In this research author develop a new sorting technique to keep in view the existing techniques. Author also proposed the algorithm i.e. Relative Split and Concatenate Sort, implements the algorithm and then compared results with some of the existing sorting algorithms. Algorithm's time and space complexity is also being the part of this paper. With respect to complexity sorting algorithms mainly can be divided into 2 categories: O(n2) and O(nlogn). The proposed algorithm Split and Concatenate Sort is under the category of O(n2) and is efficient, in terms of time complexity, than existing algorithms lay in this category. 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.**

***Index Terms*—Relative, concatenate, split, sort, RSCS, time complexity.**

I. 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". It is method by which elements are arranged in a particular order following some characteristic or law. In computer jargon, sorting is to place records or elements is particular sequence based on the rules or arrangement, followed by the record. So, the term "Sorting" is used for all the techniques used to arrange data in the desired order. Sorting gained a lot of importance in computer sciences and its applications are in file systems, sequential and multiprocessing computing, and a core part of database systems. A number of sorting algorithms have been proposed with different time and space complexities. 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. In fact, there were many installations in which the task of sorting was responsible for more than half of the computing time. Therefore a lot more consideration was put on the sorting jargon.

In particular sorting may fall into two categories [3]: 1) Ordering: placing elements of same kind in particular sequence based on their properties. 2) Categorizing: placing elements in same group or under same label based on their properties.

Time Complexity of sorting algorithms mainly falls into two classes i.e. $O(n^2)$ and O(nlogn). $O(n^2)$ algorithms works iteratively, where as those with O(*n*log*n*) time complexity are more efficient and divide-and-conquer in nature while works recursively. O(*n*log*n*) sorting algorithms are: merge sort proposed by Von Neumann in 1945, Shell's sort in 1959, and quick sort by Hoare in 1962 [4].

To search the information efficiently the arrangement of data is very important. To facilitate the human, computers consume a substantial time in ordering the data. The computational problems always have a cumbersome effect on the researchers on one hand and open the opportunities for them on the other hand. The ultimate intention of so much sorting techniques is the cost and complexity reduction of the algorithms [5].

To insert images in *Word,* position the cursor at the insertion point and either use Insert | Picture | From File or copy the image to the Windows clipboard and then Edit | Paste Special | Picture (with "Float over text" unchecked).

International Journal of Computer Theory and Engineering reserves the right to do the final formatting of your paper.

II. Research Objectives

This research is carried out with an objective to propose a new sorting algorithm i.e. Relative Split and Concatenate Sort, analyses of complexity and running time with some of the existing sorting algorithms. This sorting algorithm lies under the class of algorithms having $O(n^2)$ complexity. Author believes this algorithm will contribute a bit more in the existing of computation.

III. Literature Review

In this chapter 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. Comparison of the algorithms is summarized and presented at the end of the chapter in tabular form.

Manuscript received February 24, 2012; revised March 29, 2012.

Abdul Wahab Muzaffar, Juwaria Shafiq, and Wasi Haider Butt are with the National University of Sciences and Technology (NUST), Islamabad, Pakistan (e-mail: Wahab_muzaffar2000@ yahoo.com, juwaria.shafiq@gmail.com, butt.wasi@gmail.com).

Dr. Naveed Riaz is with the Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology (e-mail: nransari@hotmail.com).

A. *Categories of Sorting*

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

1) *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].

2) *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].

B. *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 [6].

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. Various trees of sorting algorithms has been proposed, which shows that the sorting algorithms moves from higher level to abstract algorithm and then to lower ones. In Knuth's introduction to sorting he describes:

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. 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. 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]. They describe sorting process in three steps:

Divide the set S which is to be divided into two parts say S1 and S2. Now sort these two parts getting S1' and S2' and Join the parts to get Sorted list.

This taxonomy suggested by Susan M. Merritt [12] divides the sort into two categories: easysplit/hardjoin and hardsplit/easyjoin. The directly algorithms which results from this taxonomy are quick and merge sort, other algorithms are derived conventionally from this scheme. For example, shell sort can still be understood as an application of insertion sort on various subsets of the input set (easysplit/ hardjoin); heapsort is still a selection sort that uses a convenient data structure (hardsplit/easyjoin). In fact it can be argued that all comparison-based sorting algorithms fit neatly into this taxonomy: Each is some instance of an easysplit/hardjoin or a hardsplit/easyjoin algorithm, with some lower level detail or details uniquely characterizing it [12].

C. *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:

1) *Bubble Sort:*

Bubble sort is said to be first sorting algorithm and so pioneer in sorting. It is very easy to understand and easy to implement so most widely used. Bubble sort is a sequential sorting algorithm, it sort the items in passes, in each pass list[*i*] is compared with list [*i*+1] and values are exchanged if not in order. In each pass one value is moved to the left and this will be the least value during the pass [13].These steps continue till the entire list is sorted and no swapping is needed more. Main disadvantage of bubble sort is that it takes n^2 comparisons then the length of the list e.g. if the list is of 10 elements, bubble sort takes 100 comparisons to sort the list. This is very inefficient sort as compared to today's sorting algorithms, still used in different applications. 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]. Bubble sort is more advantages in terms of memory as it takes less memory.

2) *Cocktail Sort:*

Cocktail sort is also based on the same methodology as bubble sort, also known as shaker sort, bidirectional bubble sort, cocktail shaker sort (this also refers to a variant of (selection sort), ripple sort, shuttle sort or happy hour sort, is a variation of bubble sort that is both a stable sorting algorithm and a comparison sort [14]. The cocktail sort is different from bubble sort, as it sorts the list from both directions. Cocktail sort is a bit complex than bubble sort in implementation. It is simple in nature and solves the problem with "turtles" like in bubble sort. The average and the worst case complexity of cocktail sort is equal to bubble sort i.e. $O(n^2)$[5].

3) *Friends Sort:*

Friends sort is a previous effort of the authors of this paper. Friend sort was proposed in 2009. It is a unique sorting algorithm, idea is to assume the first value as smallest and comparing it with the rest of the list and assuming the last value as biggest and comparing it with the rest of the list [5]. The running time of this algorithm is efficient than bubble sort and cocktail sort but inefficient than insertion sort and selection sort. Friend sort shows a behavior of $O(n)$ for a sorted list as a best case. Average and worst case complexity of friend sort is $O(n^2)$.

4) *Selection sort*

Selection sort is another renowned sorting algorithm. It scans the list of items and finds the smallest item by putting it in the first index of the list, then starts scanning the list for second smallest item and put it in the second index. This scanning continues until the largest item and put it in the last index of the list. 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 $O(n^2)$. When we have a sorted list and apply selection sort it shows a behavior of $O(n)$, showing its best case complexity. Selection sort is more advantages in terms of memory as it takes less memory. The number of interchanges and assignments in selection sort depends on the original order of the items in the list, but the sum of these operations do not exceed a factor of n^2[17].

5) Insertion Sort:

Insertion sort is another sorting that is very simple, efficient and well known technique. It takes one item in each pass and inserts it to the exact index in a new list. Insertion sort is very efficient for small lists. Its advantages are simple and easy to implement. 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].

6) Merge Sort:

Another algorithm, based on O($n$log$n$) category or divide-and-conquer principle, is merge sort. It was proposed by John von Neumann in 1945 [19]. The algorithm works by dividing the unsorted list into two, sorting the two sub lists recursively by applying the merge sort again. In the end merging the sub lists. Its average case and worst case complexity is O($n$log$n$), and shows a behavior of O(log$n$) in the best case [19].

7) Quick Sort:

Quick sort is fastest among the sorting algorithms proposed by Von Neumann in 1962 [5]. It is based on divide-and-conquer technique. It takes any item as a pivot and compare it with the rest of the elements in the list, keep track of items less than pivot and greater than pivot. It then divides the list and select pivot from divided lists and continues till single item lefts, at the end it concatenates the whole lists. 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($n$log$n$) [20]. In most real-world data it is possible to make design choices which minimize the probability of requiring quadratic time [20].

8) 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(n^2) [21, 22]. It got its name after its presenter, Donald Shell.

D. Research Methodology

Authors in this research paper propose a new sorting algorithm and implement it in a high level language. Some of the existing research in sorting is also being studied. The proposed Relative Split and Concatenate sort is 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. Authors also show the complexity of Relative Split and Concatenate sort for Best case, Average case, and Worst case.

E. Comparison of Different Sorting Algorithms

The following table is taken from [23], shows the comparison of different existing sorting algorithms in terms of best, average and worst running time complexity:

TABLE I: COMPARISON OF DIFFERENT SORTING ALGORITHMS

Method	Time			Space	Stability	Type
	Best	Average	Worst			
Bubble	O(n)	O(n2)	O(n2)	Constant	Stable	Exchange
Cocktail	O(n)	O(n2)	O(n2)	Constant	Stable	Exchange
Insertion	O(n)	O(n2)	O(n2)	Constant	Stable	Insertion
Merge	O(log n)	O(nlog n)	O(n log n)	Depends	Stable	Merge Sort
Quick	O(log n)	O(nlog n)	O(n2)	Constant	Stable	Exchange
Selection	O(n)	O(n2)	O(n2)	Constant	Stable	Selection
Shell	O(n)	O(n)	O(n2)	Constant	Stable	Insertion

IV. PROPOSED ALGORITHM

A. Algorithm1 (RSCS-V1)-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.
6) Else if the element is greater than (>) large average, put it in a new list of larger elements.
7) Else put it in a new list of medium elements.
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.

B. Algorithm1 (RSCS-V1)-Psedo code:

1) RelativeSplitandConcatenateSort –V1(List)

```
First: = 0, Last: = LENGTH [List], Part: = Last/3
List1:=1stPart [List], List2:=2ndPart [List]
List3:=3rdPart [List], Avg1:=AVERAGE [List1]
Avg2:=AVERAGE [List2], Avg3:=AVERAGE [List3]
j := 0, len2 := 0, len3 := 0, len4:=0;
For i ← 1 to Last
    if (List[i]>=Avg1 ||List[i]> Avg2||List[i]>Avg3)
            if (len2 == 0)
                large [len2] = List[i];
      else if (List[i] >= large[len2-1])
```

```
        large [len2] = arr1[i]
      else
        j:=len2-1;
        while j > 0 andand List[i] < large[j-1]
          j:=j-1
        For k ← j to len2-1
          large [k + 1] = large[k]
        large [j] = List [i]
        len2:= len2 + 1;
    else if (List[i]<=avg3)
      if (len3 == 0)
        small [len3] = List[i];
      else if (List[i] >= small[len3-1])
        small [len3] = List[i];
       else
        j:=len3-1;
        while j > 0  andand arr1[i] < small[j-1]
          j := j - 1;
        For k ← j to len3-1
          small [k + 1] = small [k]
        small [j] = List[i];
        len3:= len3 + 1;
else if(List[i]<avg1 || List[i]>avg3)
        if (len4 == 0)
          mid [len4] := List[i];
        else if (List[i] >= List[len4 - 1])
          mid [len4] := List[i];
        else
          l := len4 - 1;
          while l > 0 andand List[i] < mid[l – 1]
            l := l - 1;
For m ← len4-1 to 1
 mid[m + 1] := mid[m];   mid[l] = inputlist [i];
        len4 = len4 + 1;
```

C. Algorithm1(RSCS-V1):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=n2

By keen observing it the worst case running cost of algorithm is calculated to be O(*n*2). 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(*n*2) depending upon the elements in the list.

V. Comparison With Existing Sorting Algorithms

A. Proposed Algorithm: Relative Split and Concatenate Sort (RSCS V-1)

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 2.0Ghz
- RAM 256MB

1) Comparison with Bubble Sort

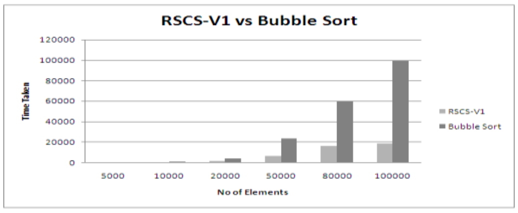

Fig. V.1. RSCS-V1 v/s bubble sort

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-V1) shows much better performance than Bubble sort that is obvious from the graph.

2) Comparison with Cocktail Sort

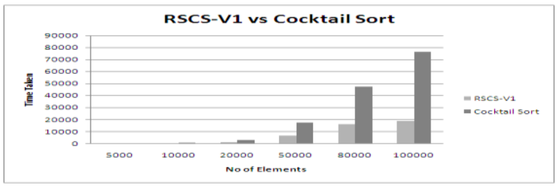

Fig. V.2. RSCS-V1 v/s cocktail sort

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-V1) shows clearly efficiency than Cocktail sort.

3) Comparison with Selection Sort

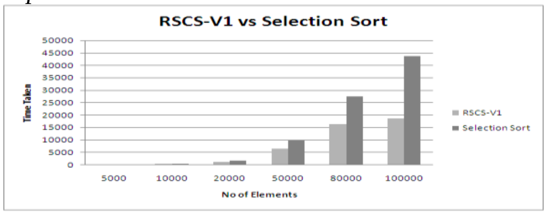

Fig. V.3. RSCS-V1 v/s selection sort

Above graph depicts the performance difference of Selection and the Relative Split and Concatenate sort (RSCS-V1). Along x-axis is the number of items in the input list while along y-axis execution times. Relative Split and Concatenate sort (RSCS-V1) shows a clear domination over Selection sort.

4) Comparison with Insertion Sort

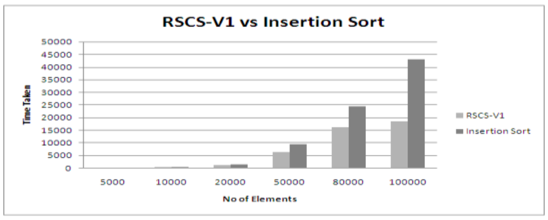

Fig. V.4. RSCS-V1 v/s insertion sort

Above graph depicts the performance difference of Insertion and the Relative Split and Concatenate sort (RSCS-V1). 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-V1) is clearly efficient than Insertion sort.

VI. Conclusion

Proposed sorting algorithm lies under the class of algorithms having $O(n^2)$ complexity. By comparing this sort with existing sorting algorithms, it is depicted from the graphs in the previous section that this sort has clear edge, in running time, over other $O(n^2)$ category algorithms i.e. bubble sort, cock tail sort, insertion sort, and selection sort. Comparisons also shows that Relative Split and Concatenate sort is inefficient than $O(n\log n)$ category algorithms i.e. merge sort and quick sort. Relative Split and Concatenate sort takes more memory which is its trade off in terms of time and space. Summarizing the whole discussion, it is clear from the results that the proposed algorithm has got its position almost above than the middle order algorithms. As it has beaten the old algorithms and has been beaten by the most efficient algorithms. As it is an n2 algorithm so if we only compare it with n2 algorithms, we will see that it is almost among the best n2 algorithms.

VII. Future Work

As in the proposed idea (Relative Split and Concatenate Sort-V1), we take only three averages and then fit elements on the base of comparison of that. This methodology can be made dynamic i.e. recursion may be involved so that in every call again arithmetic mean may be calculated and comparisons may be made. Authors are intended to incorporate binary search, with the current Algorithm, for finding the location of the number to be placed in the list. Also other ways also exist. The algorithm can be enhanced further in a number of ways. Any other existing methodology can be merged with this to get more efficient results. Any enhancement is appreciated and encouraged.

References

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

[2] T. Philippas and Z. Yi,"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)*, 2003.

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

[4] A. Agapitos and S. 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] Z. Iqbal, H. Gull, and A. W. Muzaffar, "A New Friends Sort Algorithm," *2nd IEEE International Conference on Software Engineering and Information Technology*, pp. 326-329.

[6] D. Knuth,"The Art of Computer Programming, Volume 3: Sorting and Searching,'' *Third Edition. Addison-Wesley*, 1997.

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

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

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

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

[11] C. Green and D. Barstow, "On program synthesis knowledge," *Artif. Infd*. vol. 10, pp. 241-279. 1978.

[12] S. M. Merritt, "An inverted taxonomy of Sorting Algorithms. Programming Techniques and Data Structures," *Communications of ACM*, vol. 28, no. 1, ACM, 1985

[13] J. D. Fix and R. 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). [Online]. Available: http://en.wikipedia.org/wiki/Cocktail_sort Accessed December 25, 2009.

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

[16] Heap Sort. (2009). [Online]. Available: http://en.wikipedia.org/wiki/Heap_sort, Accessed October 25, 2009.

[17] S. Lipschutz, *Theory and Problems of Data Structures, Schaum's Outline Series: International Edition*, McGraw-Hill, 1986, pp. 324–325.

[18] S. Lipschutz, *Theory and Problems of Data Structures, Schaum's Outline Series: International Edition*, McGraw-Hill, 1986, pp. 322–323.

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

[20] T. H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, "Introduction to Algorithms, Second Edition," *MIT Press and McGraw-Hill*, 2001, pp.145–149.

[21] B. Shahzad and M. T. Afzal, "Enhanced Shell Sorting Algorithm," *World Academy of Sciences, Journal*, vol. 27.

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

[23] Comparison of different sorting algorithm. [Online]. Available: http://en.wikipedia.com/sorting_algorithm, accessed may 20, 2010.