Australian Journal of Basic and Applied Sciences, 7(7): 470-475, 2013
ISSN 1991-8178

Grouping Comparison Sort

[1]Ibrahim M. Al Turani, [2]Khalid S. Al-Kharabsheh, [3]Abdallah M. AlTurani

[1,2]Applied Sciences Department, Al-balqa Applied University, Aqaba, Jordan
[3]Computer Sciences Department, Jordan University of Science and Technology

Abstract: There are many popular problems in different practical fields of computer sciences, database applications, Networks and Artificial intelligence. One of these basic operations and problems is sorting algorithm; the sorting problem has attracted a great deal of research this paper suggest a new sorting algorithm, Grouping Comparison Sort (GCS) on an approach of compare-based sorting. GCS is a sort depends on dividing the list of elements into groups, each group consisting of three elements, every group arranged separately. After that compared between groups even to access sorted list of elements. Results on time complexity $O(n^2)$, where n is the size of data being sorted. This paper make comparison between the new suggested algorithm and conventional algorithm such as selection sort, quick sort, with respect execution time to show how this algorithm perform reduce execution time.

Key words*:* sort, swaps, selection sort, quick sort,Time complexity.

INTRODUCTION

Sorting is a process of rearrangement a list of elements to the correct order since handling the elements in a certain order more efficient than handling randomize elements (Pooja Adhikari, 2007).

Sorting and searching are among the most common programming processes, as an example take database applications if you want to maintain the information and ease of retrieval you must keep information in a sensible order, for example, alphabetical order, ascending/descending order and order according to names, ids, years, departments, etc.

Information growth rapidly in our world leads to increase developing sort algorithms. Developing sort algorithms through improved performance and decreasing complexity, it has attracted a great deal of research; because any effect of sorting algorithm enhancement of the current algorithms or product new algorithms that reflects to optimize other algorithms.

Large number of algorithms developed to improve sorting like heap sort, merge sort, bubble sort , insertion sort, quick sort and selection sort, each of them has a different mechanism to reorder elements which increase the performance and efficiency of the practical applications and reduce time complexity of each one.

When comparing various sorting algorithms, there are several factors that must be taken in consideration; first of them is the time complexity, the time complexity of an algorithm determined the amount of time that can be taken by an algorithm to run(Michael Goodrich and Roberto Tamassia 2010; Michael Sipser, 1996). This factor different from sorting algorithm to another according to the size of data that we want to reorder, some sorting algorithm inefficient and too slow. In contrast, there are sorting algorithms characterized by efficient and high speed. The time complexity of an algorithm is generally written in form big O(n) notation, where the O represents the complexity of the algorithm and a value n represent the number of elementary operations performed by the algorithm (Sultanullah Jadoon , Salman Faiz Solehria, Prof. Dr. Salim ur Rehman and Prof. Hamid Jan. February, 2011)

The second factor is the stability, means, algorithm keeps elements with equal values in the same relative order in the output as they were in the input. (Thomas H. Cormen, Leiserson, Rivest and Stein, 2009; Michael Goodrich and Roberto Tamassia, 2010; ADITYA DEV MISHRA & DEEPAK GARG, 2008). Some sorting algorithms are stable by its nature such as insertion sort, merge sort, bubble sort, while some sorting algorithms are not, such as heap sort, quick sort, any given sorting algorithm which is not stable can be modified to be stable (Michael Goodrich and Roberto Tamassia, 2010).

The third factor is memory space, algorithm that used recursive techniques need more copies of sorting data that affect to memory space (Michael Goodrich and Roberto Tamassia, 2010; ADITYA DEV MISHRA & DEEPAK GARG, 2008).

Many previous researches have been suggested to enhance the sorting algorithm to maintain memory and improve efficiency. Most of these algorithms are used comparative operation between the oldest algorithm and the newest one to prove that. this paper use divide and conquer then grouping technique to prove the proposed sorting algorithm.

Selections sort approximately the nearest algorithm to the newest algorithm Grouping Comparison Sort (GCS). Because it corresponds to the selections sort first , in terms of classification as compare-based sorting and second most important in terms of the principle of work.

Corresponding Author: Ibrahim M. Al Turani, Al-balqa Applied University, Aqaba, Jordan,
E-mail: Traini111@bau.edu.jo

Related Works:

The nearest algorithm to the proposed algorithm is selection sort the simplest of sorting techniques. Selection sort works very well for small files, has a quite important application because each item is actually moved at most once(Robert Sedgewick and Kevin Wayne, 2011). It has O(n2) time complexity, making it inefficient on large lists.

Selection sort has one advantage over other sort techniques. Although it does many comparisons, it does the least amount of data moving. That means, if your data has small keys but large data area, then selection sorting may be the quickest.(Sultanullah Jadoon , Salman Faiz Solehria, Prof. Dr. Salim ur Rehman and Prof. Hamid Jan., February 2011) selection sort works as the following (Nell Dale, 2011; Anany Levitin, 2006):

1. Start with the first element, scan the entire list to find its smallest element and exchange it with the first element
2. Start with the second element, scan the remaining list to find the smallest among the last elements and exchange it with the second element
3. Continue until to get to the penultimate element

The following figure1 represent an example of implementing selection sort algorithm

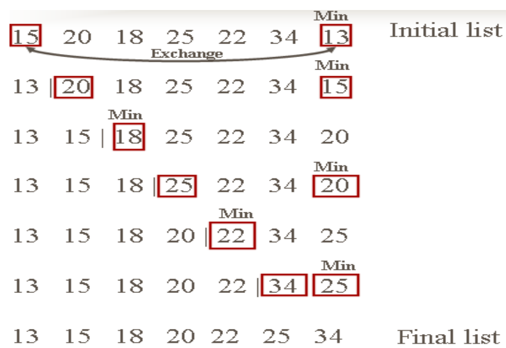

Fig. 1: example of implementing selection sort algorithm

Grouping Comparison Sort:

Concept:

Inserting an array of elements and sorting these elements in the same array (in-place) by dividing the list of elements into groups, each group consisting of three elements, every group arranged separately. After that compared between groups even to access sorted list of elements.

Procedure:

The procedure of the algorithms can be described as follows:

1. Inserting all elements of the array.
2. Calling the “Sort ” function to sort each three elements of the array as separated groups
3. Comparative between the biggest element in the first group (BEij) and the biggest element in the second group (BEik) (i : number of element in the group; j,k : number of group; j<k)
   a. if BEij less than BEik then
      i. Calling “Swap” function to swap between BEij and BEik
      ii. Calling “Sort” function to sort second group
   b. Go to the next group k=k+1,repeate step3 until access all groups to access the biggest element as first element in the array(i.e. first elements in the first group).
4. Go to the second element in the first group.
5. Comparative between the second element in the first group and the biggest element in all groups as in step3 and so on .
6. Finished after Comparative between the third element of the penultimate group and the first element of last group

Pseudocode:

The pseudocode of GCS algorithm might be expressed as:

```
for count = 0 To size-1
 sort(scarray, count )
 count = count + 3
```

```
4   end for
5   var jump=0,temp=0
6   for count = 0 To size-4
7    if (count mod 3 = 0)
8    jump=jump+3
9    temp= jump
10   end if
11   while(jump <= size)
12   if ( scarray(count) <scarray (jump) )
13   swap1(count, jump)
14   sort(scarray, jump)
15   end if
16   jump = jump +3
17   end while
18    jump =temp
19    count = count + 1
20   end for
21  function sort (array , count)
22    var num1=array(count),num2= array(count+1),num3= array(count+2)
23     if ((num1 <= num2)
24     if  (num2 <= num3))
25    scarray(count)=num3,  scarray(count+1)=num2,  scarray(count+2)=num1
26    else
27   If(num3>=num1)
28   scarray(count)=num2, scarray(count+1)=num3, scarray(count+2)=num1
29   else
30   scarray(count)=num2, scarray(count+1)=num1, scarray(count+2)=num3
31   end if
32   end if
33    else
34    if(num2>=num3)
35   scarray(count)=num1, scarray(count+1)=num2, scarray(count+2)=num3
36   else
37   if(num3>=num1)
38   scarray(count)=num3, scarray(count+1)=num1, scarray(count+2)=num2
39   else
40   scarray(count)=num1, scarray(count+1)=num3, scarray(count+2)=num2
41   end if
42    end if
43   end if
44   end function
45   function swap1(count, jump)
46   var temp;
47   temp= scarray (count), scarray (count)= scarray (jump), scarray (jump)=temp,
48  end function
```

Analysis:

For analyses time complexity T(n) of Algorithm we must compute time cost of function then compute time cost of the main procedure for sort.

- function sort (i.e. Line 21 to line 44) will be executed 9 times at as worst case in the calling function, through the first 3 statements of equality (line 22), access to the 6 selection statements (line 23,line 26, line 29, line 32, line 35 and line 38) and applying 3 statements that depends on only one selection statement (line 24 or line 27 or line 30 or line 33 or line 36 or line 39).
- function swap1 (i.e. Line 42 to line 45) will be executed 4 times at each calling, the 4 statements of equality (line 43 and line 44).
- Main procedure

- Divide the list of elements as groups, each group is sorted represent between line 1 to line 4; Calling sort function in line 2 will be executed n times T(n)line $1\rightarrow 4$ = 9 n
- Statements of equality in line 5 will be executed 2 times
- Repetition statement (for) in line 6 will be executed n-3 times

- Selection statement between line 7 to line 10 will be executed 2n times
- repetition statement (while) in line 11 will be executed

- Each group contains three elements, that compare with the first element of next groups

$$T(n)_{line\ 11} = 3 \times \sum_{i=1}^{\frac{n}{3}-1} \left(\frac{n}{3} - i\right) = 3 \times \left[\left(\frac{n}{3} - 1\right)\left(\frac{n}{3}\right) - \frac{\left(\frac{n}{3} - 1\right)\left(\frac{n}{3} - 1 + 1\right)}{2} \right]$$

$$T(n)_{line\ 11} = \left(\frac{n^2}{6}\right) - \left(\frac{n}{2}\right) times$$

- Selection statement in line 12 will be executed $T(n)_{line12} = (\frac{n^2}{6}) - (\frac{n}{2})$ times
- Calling swap1 function in line 13 will be executed $T(n)_{line13} = \frac{4n^2}{6} - 2n$ times in worst case
- Calling sort function in line 14 will be executed $T(n)_{line14} = 1.5n^2 - 4.5n$ times in worst case
- Statements of equality in line 16 will be executed $T(n)_{line16} = (\frac{n^2}{6}) - (\frac{n}{2})$ times
- Statements of equality in line 18 and line 19 will be executed n + n = 2n

So, T(n) = $12n + 2 + n - 3 + 2n + (1/6\, n^2 - 1/2\, n) + (4/6\, n^2 - 2n) + (2n^2 - 6n) + (1/6\, n^2 - 1/2\, n) + 2n$
$= O(n^2)$

After analysis of this algorithm, it shown that the biggest consumption time are spent in the implementation of line 13 and 14 after verify condition. verify the condition depends on the pre-order of the elements in the other words the nature pre-arranged of input.

The number of calling swap1 and sort functions in the line 13 and 14 respectively from GCS algorithm may elaborate as follows:

- If the input array is already sorted or after implementing part of algorithm between line 1 to line 4 that given sorted array then there is no need to call swap1 and sort functions in line 13 and 14; this represent the **best-case scenario**.
- If each pair of elements needs calling swap1 and sort functions in line 13 and 14; this represent the **worst-case scenario**.
- If some pair of elements need calling swap1 and sort functions in line 13 and 14; this represent the **average-case scenario**.

Example:

The following figure2 represent an example of implementing GCS algorithm

Performance Testing:

Quick sort algorithm represent another sorting algorithm related to the suggest algorithm. Quick sort has same classification as compare-based sorting algorithm, it is considered one of the good sorting algorithm in terms of average time complexity it has O(n log n) time complexity (D.S. Malik, 2002; J. L. Bentley and R. Sedgewick, 1997). For these reasons, we will use quick sort algorithm to show performance of the suggest sort algorithm, by comparison of time complexity between them; after provide the inputs that have the same type and size.

Complexity Comparison Between Typical Sort Algorithms:

The comparison of complexity between GCS and conventional sort algorithms are listed in table 1(Cormen T., Leiserson C., Rivest R., and Stein C., 2001). Table1 determines the time complexity of suggest algorithm is equivalent to some conventional sort algorithms. GCS given an additional method to manipulate information.

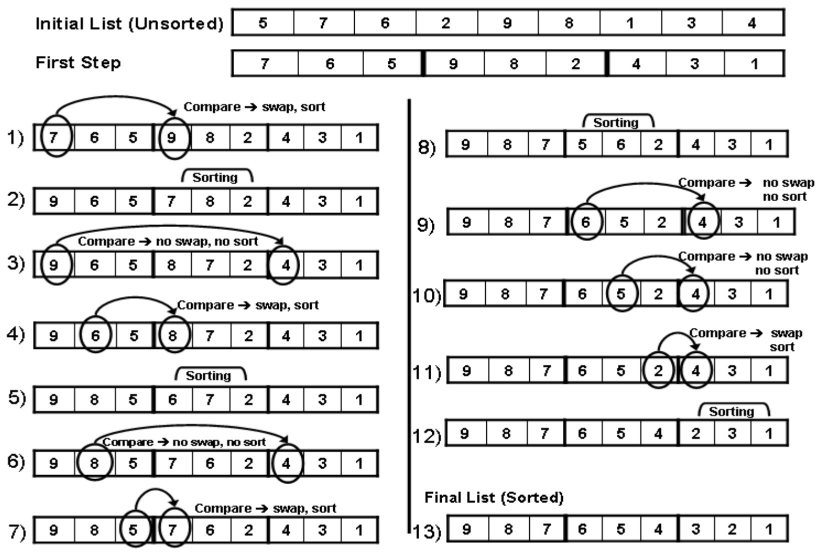

Fig. 2: example of implementing GCS algorithm

Table 1: Time complexity of typical sorting algorithms

Algorithm	Average case	Worst case
Selection sort	O(n^2)	O(n^2)
Quick sort	O (n log n)	O(n^2)
Grouping Comparison Sort	O(n^2)	O(n^2)

Performance In Average Case:

This paper implemented of selection sort quick sort and GCS algorithms using C++ programming language, and measure the execution time of all programs with the same input data using the same computer. The built-in function (clock ()) in C++ is used to get the elapsed time of the implementing algorithms, execution time of a program is measured in milliseconds (Deitel H. and Deitel P., 2001).

The performances of GCS algorithm and a set of conventional sort algorithms are comparatively tested under average cases by using random test data from size 1000 to 30000. The result obtained is given in Table 2 and the curves are shown in figure 3.

Table 2: Execution time for the three algorithms (ms)

Algorithm	Size of input					
	1000	10000	15000	20000	25000	30000
Selection sort	302	3580	5623	7947	10068	12785
Quick sort	269	3245	5045	6859	8483	10721
Grouping Comparison Sort	23	1721	3692	6337	9479	13314

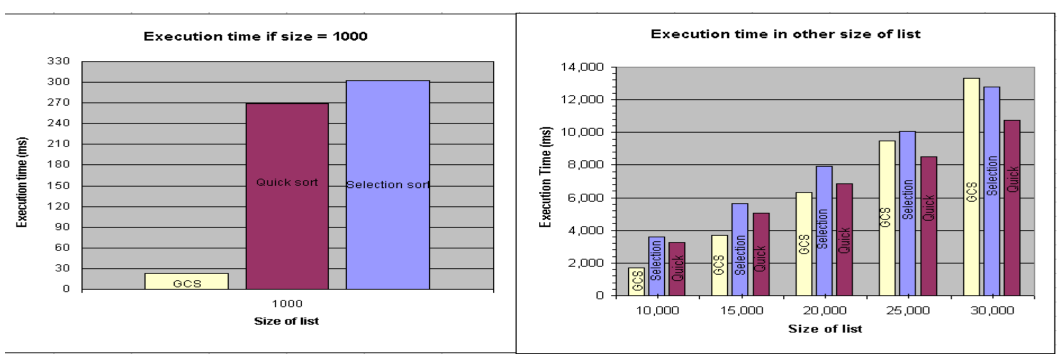

Fig. 3: Execution time for the three algorithms (ms)

It is obvious that the suggested algorithm required less execution time than conventional algorithms(i.e. Selection sort and Quick sort) , especially if the size of list is less than 20,000 element, but if the size of the list between 20,000 to 25,000 have a closed execution time more to selection sort than quick sort , finally, if the size of list is greater than 25000 elements; conventional algorithms needs less execution time than GCS .

Conclusions:

Through this paper present a new sorting algorithm. Grouping Comparison Sort (GCS) algorithm; identified efficiency through time complexity as shown in analysis section has O(n2). and performance through the practical experience to showing the amount of time spent in order to sort list as shown in performance in average case section; It appeared performance has been decreased by GCS algorithm, mainly if the input size more than 25000 elements that returned increasing number of comparison, the performance has been improved when size of input is less than 25000 elements.

Applications must be arranged to improved the performance of operations, this algorithm may give us an extra option for arrange data depends on size of data and at any application are used.

In future work we hope to enhance GCS algorithm by searching in features to improve performance and reduce execution time, especially if the size of the list is more than 30,000 elements.

REFERENCES

ADITYA DEV MISHRA & DEEPAK GARG, 2008. "SELECTION OF BEST SORTING ALGORITHM ", International journal of intelligent information Processing.

Anany Levitin, 2006. Introduction to the Design and Analysis of Algorithms ,Second Edition.

Bentley, J.L. and R. Sedgewick, 1997. "Fast Algorithms for Sorting and Searching Strings", ACM-SIAM SODA, 97: 360-369.

Cormen, T., C. Leiserson, R. Rivest and C. Stein, 2001. Introduction to Algorithms, McGraw Hill.

Deitel, H. and P. Deitel, 2001. C++ How to Program, Prentice Hall.

Malik, D.S., 2002. C++ Programming: Program Design Including Data Structures.

Michael Goodrich and Roberto Tamassia, 2010, Data Structures and Algorithms in Java ,4th edition.

Michael Sipser, 1996. Introduction to the Theory of Computation.

Nell Dale, 2011. C++ Pulse Data Structure, Fifth Edition.

Pooja Adhikari, 2007. Review on Sorting Algorithms, A comparative study on two sorting algorithm, Mississppi state university

Robert Sedgewick and Kevin Wayne, 2011. Algorithms, 4th Edition.

Sultanullah Jadoon, Salman Faiz Solehria, Prof. Dr. Salim ur Rehman and Prof. Hamid Jan. 2011. " Design and Analysis of Optimized Selection Sort Algorithm". IJECS-IJENS Vol: 11 No: 01.

Thomas, H., Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, 2009. Introduction To Algorithms, Third Edition.