Scientific Research and Essay Vol. 4 (8), pp. 740-744, August, 2009
Available online at http://www.academicjournals.org/SRE
ISSN 1992-2248 © 2009 Academic Journals

Full Length Research Paper

Improving the performance of bubble sort using a modified diminishing increment sorting

Oyelami Olufemi Moses

Department of Computer and Information Sciences, Covenant University, P. M. B. 1023, Ota, Ogun State, Nigeria. E-mail: olufemioyelami@yahoo.com or olufemioyelami@gmail.com. Tel.: +234-8055344658.

Accepted 17 February, 2009

Sorting involves rearranging information into either ascending or descending order. There are many sorting algorithms, among which is Bubble Sort. Bubble Sort is not known to be a very good sorting algorithm because it is beset with redundant comparisons. However, efforts have been made to improve the performance of the algorithm. With Bidirectional Bubble Sort, the average number of comparisons is slightly reduced and Batcher's Sort similar to Shellsort also performs significantly better than Bidirectional Bubble Sort by carrying out comparisons in a novel way so that no propagation of exchanges is necessary. Bitonic Sort was also presented by Batcher and the strong point of this sorting procedure is that it is very suitable for a hard-wired implementation using a sorting network. This paper presents a meta algorithm called Oyelami's Sort that combines the technique of Bidirectional Bubble Sort with a modified diminishing increment sorting. The results from the implementation of the algorithm compared with Batcher's Odd-Even Sort and Batcher's Bitonic Sort showed that the algorithm performed better than the two in the worst case scenario. The implication is that the algorithm is faster.

Key words: Algorithm, sorting, bubble sort, bidirectional bubble sort, Batcher's sort, Oyelamis's sort, worst case, swapping, comparison.

INTRODUCTION

Using computer to solve a problem involves directing it on what steps it must follow to get the problem solved. The steps it must follow is called algorithm. An algorithm is a finite sequence of explicit instructions to solve a problem with a finite amount of effort in a finite amount of time (William, 2005; Alfred et al., 2002).

Algorithms are paramount in computer programming, but an algorithm could be of no use even though it is correct and gives a desired output if the resources like storage and time it needs to run to completion are intolerable.

Instructions can be executed any number of times, provided the instructions themselves indicate repetition. However, no matter what the input values may be, an algorithm terminates after executing a finite number of instructions. A program is thus an algorithm in as much as it does not loop infinitely on any input (Sara and Allen, 2000).

Five important features of algorithm are (Donald, 1997):

i.) Finiteness: An algorithm must always terminate after a finite number of steps.
ii.) Input: An algorithm has zero or more inputs- quantities that are given to it initially before the algorithm begins, or dynamically as the algorithm runs. These inputs are taken from specified sets of objects.
iii.) Definiteness: Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously specified for each case.
iv.) Output: An algorithm has one or more outputs-quantities that have a specified relation to inputs.
v.) Effectiveness: An algorithm is also generally expected to be effective, in the sense that its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper.

A sort is a process that rearranges the records of a file into a sequence that is sorted on some key. Sorting organizes a collection of data into either ascending or descending order (Yedidjah and Aaron, 2003; Frank, 2004).

Sorting can be categorized into two categories internal

sorting requires that the collection of data fit entirely in the computer's main memory while in external sorting, the data collectively will not fit in the computer's main memory all at once but must reside in auxiliary storage such as disk (Yedidjah and Aaron, 2003; Frank, 2004; Shola, 2003). Sorting algorithms for serial computers (random access machines or RAMs) allow only one operation to be executed at a time. In sorting algorithms based on a comparison network model of computation, many comparison operations can be performed simultaneously. Comparison networks differ from RAM's in two important aspects. First, they can only perform comparisons. Second, unlike the RAM model in which operations occur serially, that is, one after another, operations in a comparison network may occur the same time or "in parallel".

A sorting network is a comparison network for which the output sequence is monotonically increasing (that is, $b_1 \leq b_2 \leq \ldots \leq b_n$) for every input sequence (Thomas et al., 2003).

Diminishing increment sort as used by Shellsort provides a simple and efficient sorting algorithm. This algorithm improves on Insertion Sort by reducing the number of inversions and comparisons made on the elements to be sorted. It sorts an array 'A' with 'n' elements by dividing it into subsequences and sorts the subsequences. It is called diminishing increment sorting because the increments continue to decrease from one pass to the other until the last increment is 1.

Bubble sort is a kind of internal sorting that compares adjacent items and exchanges them if they are out of order and continues until the file is sorted (Frank, 2004; Robert, 1998). Bubble sort is however, not an efficient algorithm because it is a quadratic-time sorting algorithm. However, efforts have been made to improve the performance of the algorithm. With Bidirectional Bubble Sort, the average number of comparisons is slightly reduced and Batcher's Sort similar to Shellsort also performs significantly better than Bidirectional Bubble Sort by carrying out comparisons in a novel way so that no propagation of exchanges is necessary. Bitonic Sort was also presented by Batcher and the strong point of this sorting procedure is that it is very suitable for a hard-wired implementation using a sorting network.

This paper presents an algorithm that combines the technique of Bidirectional Bubble Sort with a modified diminishing increment sorting to improve Bubble sort. The results obtained from the implementation of the algorithm compared with Batcher's Odd-Even Sort and Bitonic Sort showed that the algorithm is the fastest of the three.

MATERIALS AND METHODS

Oyelami's Sort was developed by modifying diminishing increment sort as used by Shellsort and then applied it to the elements to be sorted before applying Bidirectional Bubble Sort. It is to be noted that the kind of diminishing increment used is different from that of Shellsort but as used by Oyelami (2008) and Oyelami et al. (2007).

Bubble sort

To understand how Bubble Sort works, consider an array containing elements to be sorted. We look through the array and pick the smallest element and put it in position 1. That is the first pass. We look through the remaining list from the second element to the last and pick the smallest and put in position 2 and so on until all the elements are sorted. Consider the array of numbers below for instance:

8 4 3 2

Figure 1 gives a pictorial representation of how the sorting will be done.

Refinements on bubble sort

There have been some improvements on Bubble sort as discussed below:

Bidirectional bubble sort

Bidirectional Bubble Sort also known as Cocktail Sort or Shaker Sort is a variation of Bubble Sort that is both a stable sorting algorithm and a comparison sort. The algorithm differs from Bubble Sort in that sorts in both directions each pass through the list. The average number of comparisons is slightly reduced by this approach (Donald, 1998). This sorting algorithm is only marginally more difficult than Bubble Sort to implement, and solves the problem with so-called turtles in Bubble Sort. Consider the problem of sorting the same set of numbers used for Bubble Sort:

8 4 3 2

The algorithm does the sorting as shown in Figure 2. There are 7 comparisons and 6 swaps in all.

Batcher's odd and even merge sort

If you have a list of keys arranged from left to right, and you sort the left and right halves of the list separately, and then sort the keys in even positions on the list, and in odd positions separately, then all you need do is compare and switch each even position key (counting from the left) with the odd position key immediately to its right, and you will have completely sorted the whole list. The algorithm can be summarized as follows: Sort Algorithm for 2m keys = Sort left half and Sort right half; Then Merge the two halves, and can describe the Merge step as Merge 2m keys = Merge m odd keys and m even keys. Then compare and switch each even key with odd key to its right.

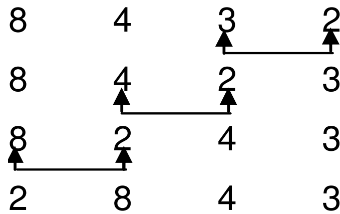
For an illustration of how Batcher's Sort works, consider the numbers 8 4 3 2 considered above. The numbers are sorted as shown In Figure 3.

In all, there are 5 comparisons and 4 swaps and these show the superiority of Batcher's Sort over Bidirectional Bubble Sort.

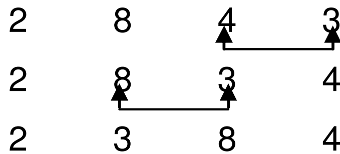
Bitonic sort

A Bitonic sequence is one that monotonically increases and monotonically decreases. A Bitonic sorter is composed of several stages, each of which is called a half-cleaner. Each half-cleaner is a comparison network of depth 1 in which input line i is compared with line i + $\frac{n}{2}$ for I = 1, 2, … $\frac{n}{2}$ (n is assumed to be even). By re-

First Pass



Second Pass



Third Pass

Figure 1. Illustration of bubble sort

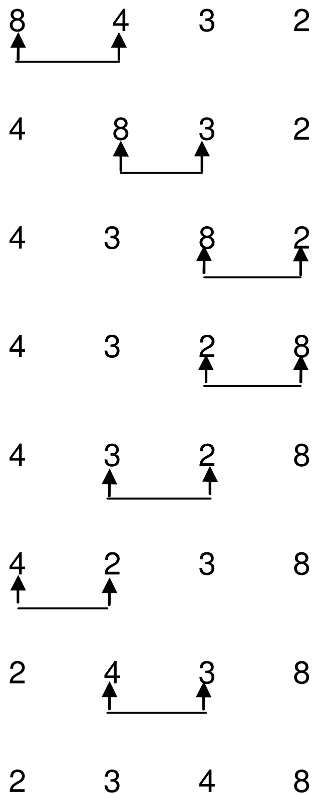


Figure 2. Illustration of bidirectional bubble sort

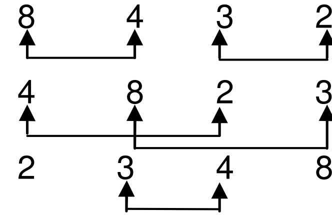
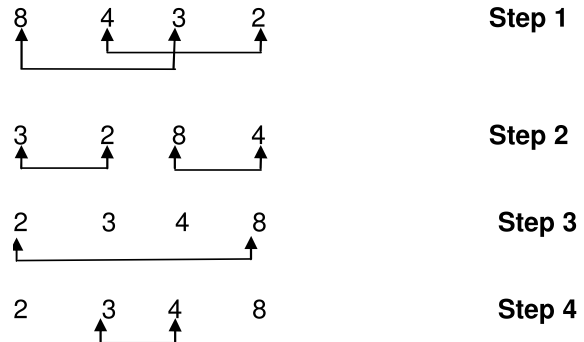


Figure 3. Illustration of batcher's odd and even merge sort.

cursively combining half-cleaners, a Bitonic sorter can be built which is a network that sorts bitonic sequences (Thomas et al., 2003). For an illustration of how Bitonic sort works, consider the usual problem of sorting the numbers: 8, 4, 3 and 2. The numbers are sorted as follows:

Half Cleaners are used in steps 1 and 2 and Bitonic Mergers used in steps 3 and 4.



In all, there are 6 comparisons and 4 swaps.

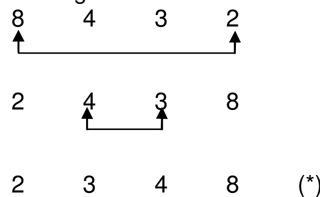
The proposed algorithm (Oyelami's sort)

This proposed sorting algorithm divides the elements to be sorted into subsequences just like Shellsort does but by first of all comparing the first element with the last. If the last is less than the first, the two swap positions, otherwise, they maintain their positions. Later, the second element is compared with the second to the last, if the second to the last element is smaller than the second, they are swapped. Otherwise, they maintain their positions. This process continues until the last two consecutive middle elements are compared or until it remains only one element in the middle. After this, Bidirectional Bubble Sort is applied to sort the adjacent elements. This approach reduces the number of comparisons and inversions carried out significantly.

Consider the worst-case scenario of sorting the following elements used for Batcher's Sort and Bitonic Sort in ascending order:

8 4 3 2

The algorithm works like this:



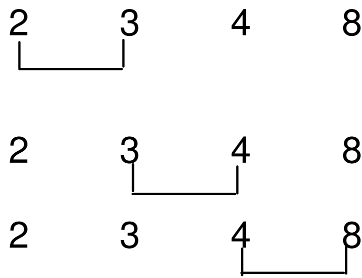


Figure 4. Illustration of Oyelami's sort.

Bidirectional Bubble Sort is now applied to (*) to sort the elements that are adjacent as shown in Figure 4.

Since no swap has occurred, the algorithm stops, eliminating the need to pass from the top back to the bottom. In all, 5 comparisons were carried out and only 2 swaps. This shows that this algorithm performs better than Batcher's that has 5 comparisons and 4 swaps. When compared with Bitonic Sort (6 comparisons and 4 swaps) it performs better. The algorithm is presented below:

```
Oyelami's Sort (array, size)
Begin
1. i = 1
2. j = size
3. while (i < j) do
begin
4. if array[i] > array[j] swap (array, i, j)
5. i =  i + 1
6. j = j – 1
end
[Call Bidirectional Bubble Sort to sort the adjacent elements]
7. Bidirectional Bubble Sort (A, size:int)
End
```

Performance analysis of algorithms

The most important attribute of a program/algorithm is correctness. An algorithm that does not give a correct output is useless. Correct algorithms may also be of little use. This often happens when the algorithm/program takes too much time than expected by the user to run or when it uses too much memory space than is available on the computer (Sartaj, 2000). Performance of a program or an algorithm is the amount of time and computer memory needed to run the program/algorithm. Two methods are normally employed in analyzing an algorithm:

i.) Analytical method
ii.) Experimental method

In analytical method, the factors the time and space requirements of a program depend on are identified and their contributions are determined. But since some of these factors are not known at the time the program is written, an accurate analysis of the time and space requirements cannot be made. Experimental method deals with actually performing experiment and measuring the space and time used by the program. Two manageable approaches to estimating run time are (Sartaj, 2000):

i.) Identify one or more key operations and determine the number of times they are performed.
ii.) Determine the total number of steps executed by the program.

Worst-case, best-case and average-case analysis of sorting algorithms

The worst-case occurs in a sorting algorithm when the elements to be sorted are in reverse order. The best-case occurs when the elements are already sorted. The average–case may occur when part of the elements are already sorted. The average-case has data randomly distributed in the list (William and William, 2002). The average case may not be easy to determine in that it may not be apparent what constitutes an 'average' input. Concentration is always on finding only the worst-case running time for any input of size n due to the following reasons (Thomas et al., 2003):

i.) The worst-case running time of an algorithm is an upper bound on the running time for any input. Knowing it gives us a guarantee that the algorithm will never take any longer. We need not make some educated guess about the running time and hope that it never gets much worse.
ii.) For some algorithms, the worst-case occurs fairly often. For example, in searching a database for a particular piece of information, the searching algorithm's worst-case will often occur when the information is not present in the database. In some searching applications, searches for absent information may be frequent.
iii.) The "average-case" is often roughly as bad as the worst case.

Analysis of the proposed algorithm

Generally, the running time of a sorting algorithm is proportional to the number of comparisons that the algorithm uses, to the number of times items are moved or exchanged, or both (Robert, 1998). The approach used in this paper is to measure the number of comparisons and exchanges carried out by each algorithm (Batcher's Sort, Bitonic Sort and Oyelami's Sort) in the worst case scenario.

RESULTS AND DISCUSSION

Table 1 shows the result obtained. From the results in Table 1, the proposed algorithm has fewer numbers of comparisons and swaps compared with both Batcher's Odd-Even Sort and Bitonic Sort. The results also show that as the size of the input increases, the proposed algorithm tends to be more efficient as both Batcher's Odd-Even and Bitonic sorts are not good for large values of input. The implication of these is that the proposed algorithm is faster and therefore, more efficient. The algorithm is also recommended for large values of inputs to be sorted.

Conclusion

Bubble Sort is not known to be a good algorithm because it is a quadratic-time sorting algorithm. However, efforts have been made to improve the performance of the algorithm. With Bidirectional Bubble Sort, the average number of comparisons is slightly reduced and Batcher's Sort similar to Shellsort also performs significantly better than Bidirectional Bubble Sort by carrying out comparisons in a novel way so that no propagation of exchange Is necessary. This paper has further improved on Batch-

Table 1. Comparison of Batcher's Sort, Bitonic Sort and Oyelami's Sort Performances.

	Batcher's Odd-Even Sort		Bitonic Sort		Oyelami's Sort	
Size of Input	Number of Comparisons	Number of Swaps	Number of Comparisons	Number of Swaps	Number of Comparisons	Number of Swaps
4	5	4	6	4	5	2
8	19	12	24	14	11	4
16	63	32	80	44	23	8
32	191	80	240	128	47	16
64	543	192	672	312	219	41
128	1471	448	1792	928	191	64
256	3839	1024	4608	2368	383	128

er's Sort using the technique of Bidirectional Bubble Sort and a modified diminishing increment sorting. The experimentation of the proposed algorithm and Batcher's Sort has shown that the proposed algorithm is more efficient. The algorithm is recommended for all sizes of elements to be sorted but much more efficient as the elements to be sorted increases.

REFERENCES

Alfred V, Aho J, Horroroft, Jeffrey DU (2002). Data Structures and Algorithms (India: Pearson Education Asia).

Donald EK (1997). The Art of Computer Programming, Volume I, Fundamental Algorithms; Third Edition. US: Addison-Wesley.

Donald EK (1998). The Art of Computer Programming, Volume 3, Sorting and Searching, Second Edition. Addison-Wesley.

Frank MC (2004). Data Abstraction and Problem Solving with C++. US: Pearson Education, Inc.

Oyelami MO (2008). A Modified Diminishing Increment Sort for Overcoming the Search for Best Sequence of Increment for Shellsort". J. Appl. Sci. Res., 4 (6): 760- 766.

Oyelami MO, Azeta AA, Ayo CK (2007). Improved Shellsort for the Worst-Case, the Best-Case and a Subset of the Average-Case Scenarios. J. Comput. Sci Appl. 14 (2): 73- 84.

Robert S (1998). Algorithms in C. Addison-Wesley Publishing Company, Inc.

Sara B, Allen G (2000). Computer Algorithms. US: Addison Wesley Longman.

Sartaj S (2000). Data Structures, Algorithms and Applications in Java. McGrawHill.

Shola PB (2003). Data Structures With Implementation in C and Pascal. Nigeria: Reflect Publishers.

Thomas HC, Charles EL, Ronald LR, Clifford S (2003). Introduction to Algorithms. The Massachusetts Institute of Technology.

William F, William T (2002). Data Structures With C++ Using STL. Prentice Hall.

William JC (2005). Data Structures and the Java Collections Framework (US: The McGrawHill Companies, Inc).

Yedidjah L, Moshe A, Aaron MT (2003). Data Structures Using Java. US:Pearson Education, Inc.