Magnetic Bubble Sort Algorithm

Obed Appiah
University of Energy and Natural Resources
Sunyani, Ghana

Ezekiel Mensah Martey
Christ Apostolic University College
Kumasi, Ghana

ABSTRACT
Sorting a list of items is one basic task in many applications used on the computer. The term describes the arrangement of a set of items in a certain order to make analysis and processing very easy. Numerous sorting algorithms exist however its efficiency and memory space consumption become a major issue when it has to be implemented.Essentially, programmers select sort algorithms that perform well even as the size of the input data increases. In this study, a new algorithm, Magnetic Bubble Sort Algorithm (MBS) is proposed. The MBS is an enhancement of the bubble sort algorithm which offers a far better performance in the case where redundancies occur in the list. The run time of the MBS depends on the number of distinct values that are found in the list to be sorted.The improved bubble sort algorithm is very simple to analyse, considering the fact that the time complexity of the algorithm depends on two main factors that is the size of list (n) and number of distinct values in the list.

General Terms
Algorithms, Sorting Algorithms, Bubble Sort, Exchange Sort

Keywords
Algorithms, sorting algorithms, bubble sort, exchange sort, redundancies in dataset

1. INTRODUCTION
One of the basic problems of Computer Science is *sorting* a list of items.This is the arrangement of a set of items either in increasing or decreasing order. The formal definition of the sorting problem is as follows:

Input: A sequence having n numbers in some random order

$$(a_1, a_2, a_3, \ldots\ldots a_n)$$

Output: A permutation $(a'_1, a'_2, a'_3, \ldots\ldots a'_n)$ of the input sequence such that

$$a'_1 \leq a'_2 \leq a'_3 \leq \ldots\ldots a'_n$$

For instance, if the given input of numbers is 59, 41, 31, 41, 26, 58, then the output sequencereturned by a sorting algorithm will be 26, 31, 41, 41, 58, 59 [2].

Sorting is considered as a fundamental operation in Computer Science and various algorithms have been proposed to improve the sorting process. All sorting algorithms seek to improve the running time or the space used by sorting algorithm to quickly and efficiently sort a given list.One of the main reasons for sorting list is to generally facilitate the process of searching and in today's world, timely search for information is critical for the survival of institutions. Binary search which is one of the fastest search algorithms requires that the list from which the key to be searched must be sorted before the search could be done accurately at all times. Data is generally sorted to facilitate the process of searching. As a result of its vital or key role in computing, several techniques for sorting have been proposed.

There are several sorting algorithms today.Among them are bubble sort, insertion sort, selection sort, merge sort, quick sort, heap sort, radix sort, counting sort, and bucket sort. Other algorithms are mostly enhancement of one of these algorithms or sometimes hybrid of two or more of the existing algorithm. According to Jadoon et al, 2011, there is no ideal sorting algorithm for all types or kinds of dataset but the selection of a particular type or kind may depend on one of the following conditions

- The size of the list (number of elements to be sorted).
- The extent up to which the given input sequence is already sorted.
- The probable constraints on the given input values.
- The system architecture on which the sorting operation will be performed.
- The type of storage devices to be used: main memory or disks[5].

Almost all the available sorting algorithms can be categorized into two groups based on theirdifficulty. The complexity of an algorithm and its relative effectiveness are directly correlated [3,6]. Astandardized notation i.e. Big O(n), is used todescribe the complexity of an algorithm. In thisnotation, the O represents the complexity of the algorithm and (n) represents the size of the inputdata values. The two groups of sorting algorithms have two main run time average cases and they are $O(n^2)$, which includes the bubble, insertion,selection sort and O(nlogn) which includes the merge, heap & quick sort.

1.1 Bubble Sort Algorithm
Bubble sort, is an example of an exchange sort and sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The number of repetition required is usually proportional to the size of the list, that is, the greater the number of items to be sorted, the higher the number of passes required for the sorting to be completed successfully. The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort[9]. It can be practical if the input is usually in sort order or relatively small, but could be worst when the list is already sorted in the reversed order as the required one.

Donald Knuth, in his famous book The Art of Computer Programming, concluded that "the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems", some of which he then discussed in his book[9].

1.2 Algorithm: *Bubble Sort*(*a*[], n)

Here ***a*** is the unsorted input list and **n**is the size of the list or number of items in the list. After completion of the algorithm the array will become sorted.

```
n← length(A)
repeat for j = 1 to n-1
        repeat for i← 1 to n-1
                if A[i] > A[i+1]
                        //swap(A[i], A[i+1])
                        temp←A[i]
                        A[i] ←A[i+1]
                        A[i+1] ← temp
```

Performance Analysis

Bubble sort is considered to be the most inefficient algorithm for the reason that it has a worst case and average case complexity of $O(n^2)$, where n is the number of elements to be sorted. Likewise some other simple sorting methods such as insertion sort and selection sort have the same average case complexity of $O(n^2)$ however the efficiency of bubble sort is comparatively lesser than these algorithms. Hence thesecomputational complexity shows that bubble sort should not be considered over a large amount of data items as commented by Don Knuth [9].Moreover, there is a better way of implementing the bubble sort described in the modified bubble sort. It suggests a few changes to the standard bubble sort which includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made then it certainly show that the array is already in the desired order. With such modification to the bubble sort, the algorithm does not actually perform any operations on data, but simply evaluate the content of the list and conclude whether data is sorted or not. Which gives the best case complexity of O(n) if the array is already sorted. Another modification of the bubble sort is the bidirectional Bubble Sort or Cocktail Sort which sorts the list in both directions each pass through the list. This process slightly reduces the number of comparisons. Moreover Batcher [1] proposed a method whose running time is better than both straight bubble sort and Bidirectional Bubble sort.

Advantages and Disadvantages Bubble Soft Algorithm

Even though bubble sort is considered to be the most inefficient algorithm, it has some advantage over other algorithms such as simplicity, ease of implementation, and the ability to identify that a list is already sorted if it is efficiently implemented. On the other hand the drawbacks of bubble sort include code inefficient, inappropriate for large volumes of data elements and repetitive problems as well[9].

Table 1.0 shows the time complexity of the algorithm in three different situation of the input list.

Table 1.0: Time Complexity of Bubble Sort Algorithm

Best case	*Worst case*	*Average case*
$O(n^2)$	$O(n^2)$	$O(n^2)$

2. CONCEPT OF MAGNETIZED BUBBLE SORT ALGORITHM

Generally, large dataset will contain a couple of repetitions. For example sorting the ages of citizens of a country with population of about 15 million will contain a lot of repetitions. If age ranges between 0 through 100 then each age value could have a frequency of about 150,000 (15,000,000/100). In terms of population, more than half will be below the ages of fifty(50).Another example will be sorting the heights of a given population of a country. The existingbubble sort will execute such list in the order of$O(n^2)$ in the worst case scenario, but the proposed algorithm can do better. The main concept of the proposed algorithm is to evaluate the data in the list and take advantage of the number of distinct values in the list in order to complete the sorting faster.The proposed algorithm, thus, takes advantage of possible redundancies in the given list to sort and thereby preventing repetition of comparison of values of the same magnitude.

The Counting sort is one popular algorithm that attempts to sort by considering redundancies in the given list as well. The algorithm is an integer sorting algorithm with linear running time complexity. Like radix sort, counting sort also works based on the keys with range between 0and n, where n is the maximum value in the input array. Counting sort assumes that the input consists of an integer values and the range of values from the minimum and maximum is small. It works by countingthe number of occurrences of each element in the input usually called keys and store this information into another array say C. Finally, it determined the positionof each key value in the final sorted array by using some arithmetic operations on the data in array C. It is not a comparison sort and it preserves the relative orderof an element with equal keys. Counting sort is often used as a subroutine in radix sort [7]. Its main challenges are that it is used for integer values only and also the range of values must be small for effective sorting. For example if you are to sort 10 values with least been 0 and maximum of 2,000,000 then the sorting may require memory space of 2,000,000 in order to maintain array C. This generally increases the space needed by the algorithm and the run time may also increase as result of large C size. With such situation, the bubble sort may finish before the Counting sort. It must be noted that it works very fast and in linear time complexity O(n + k) when the range (k) is very small irrespective of the size (n) and the values are all integers.

Another algorithm that tries to takes advantage of redundancies in a given list to effectively sort data is the Improved Selection Sort Algorithm(ISSA). ISSA however sorts efficiently whether values are integers or not. The algorithm's performance is based on the number of distinct values in the given set [4]. Here the sorting does not depend on whether the content of the list are integer values or not. The ISSA is quite robust as compared to the Counting sort, but the latter works far better than the ISSA if all values to be sorted are integers. The ISSA can perform in linear time complexity when the number of distinct values in a given list is very small, but has a worst case of $O(n^2)$ when the list does not contain duplications or very small number of duplicated values. Another challenge of the ISSA is that it required a queue of size *n* in order to work at all times. This makes the space complexity of the algorithm to be $O(2n)$. In situation where there is a limited memory space, the algorithm may not work efficiently and also if the queue is not managed well may lead to overflow.

2.1 Magnetic Bubble Sort

The MBS implements a similar concept as that of the ISSA, by looking for redundancies and improving the time required to sort a list. Here the algorithm does not required a queue or stack or any extra memory in order to sort the list, meaning that when it comes to space complexity it is far better than ISSA especially with large dataset, however the time needed to sort can usually be estimated to be the same as ISSA

The MBS works primarily like the bubble sort, but this time instead of comparing adjacent values and swapping them in the right order a block of values (subset of the given list) is introduced in the list. The comparison this time is done between the block and an adjacent value. The following principles are generally considered.

1. A block (A continuous subset of the given list to be sort)
2. All members of the block must be of the same value.
3. The block can expand by attracting items or elements of the same value to its self.
4. The least number of values to be in the block must be 1 and the maximum equal to n.
5. Block can expand when adjacent value is equal to the values in the block. That is the adjacent value of equal magnitude is attracted to the block, hence the magnetic effect established.
6. The magnetic bar (the block) is dropped or demagnetised and the new one made whenever the values in the set are not in place with the adjacent one.
7. Block is simply maintained with 2 integer variables (pointers) that points to the start and end of the set. The end expands to accommodate new values of same magnitude
8. In situations where a block has to be swapped with the adjacent value, the first item in the block and the adjacent value are simply swapped. After this operation, it will then appear as if the whole block has been swapped with just a single data. The two pointers of the block or magnetic are adjusted accordingly

Example of Magnetic Bubble Sort (Ascending Order)

Block set is presented in square bracket [ ]
List – A[n]
Initial List

A	2	2	1	5	2	5	4	4	5	5

1st Pass
{[2], 2, 1, 5, 2, 5, 4, 4, 5, 5}
{[2, 2], 1, 5, 2, 5, 4, 4, 5, 5}
{1, [2, 2], 5, 2, 5, 4, 4, 5, 5}
{1, 2, 2, [5], 2, 5, 4, 4, 5, 5}
{1, 2, 2, 2, [5], 5, 4, 4, 5, 5}
{1, 2, 2, 2, [5, 5], 4, 4, 5, 5}
{1, 2, 2, 2, 4, [5, 5], 4, 5, 5}
{1, 2, 2, 2, 4, 4, [5, 5], 5, 5}
{1, 2, 2, 2, 4, 4, [5, 5, 5], 5}
{1, 2, 2, 2, 4, 4, [5, 5, 5, 5]}

At the end of the first pass, 4 items or elements are already in their correct places. The subsequence passes will only will have to deal with six(6) element instead of nine(9) if it had been the traditional bubble sort.

2nd Pass
{[1], 2, 2, 2, 4, 4, 5, 5, 5, 5}
{1, [2], 2, 2, 4, 4, 5, 5, 5, 5}
{1, [2, 2], 2, 4, 4, 5, 5, 5, 5}
{1, [2, 2, 2], 4, 4, 5, 5, 5, 5}
{1, 2, 2, 2, [4], 4, 5, 5, 5, 5}
{1, 2, 2, 2, [4, 4], 5, 5, 5, 5}
3rd Pass
{[1], 2, 2, 2, 4, 4, 5, 5, 5, 5}
{1, [2], 2, 2, 4, 4, 5, 5, 5, 5}
{1, [2, 2], 2, 4, 4, 5, 5, 5, 5}
{1, [2, 2, 2], 4, 4, 5, 5, 5, 5}
4th Pass
{[1], 2, 2, 2, 4, 4, 5, 5, 5, 5}

Sorted List
{1, 2, 2, 2, 4, 4, 5, 5, 5, 5}

The list can be sorted with just four passes, but will require at least nine(9) by the traditional bubble sort.

The Magnetic Bubble Sort Algorithm (MBS) is content sensitive, in that the nature of data distribution of the list will greatly influence the run time of the algorithm. The run time of the MBS depends on the number of distinct values that are found in the list to be sorted. If the number of distinct values is big or equal to n, then the run time of the algorithm can be approximated as $O(n^2)$. However, if the number is very small, the algorithm completes the sorting in the order of $O(n)$.

Algorithm in implementation in Python

```
print("\n\nProposed Improved Bubble Sort Algorithm");
#B is the list of values to be sorted
#Let q be the start pointer of the block or magnet
#Let r be the rear pointer or end pointer of the block or magnet.
#Note, element are attracted or added to the block at the rear
#end of the block.
#Let x be a variable to determine or point to the end of the
#unsorted part of the list been sorted. This does not change
#with a common difference as seen in the traditional bubble,
#but it is affected by the size of the block that is moved to the
#end of the unsorted portion during each pass.

i=0;
n=20;
x = n;
whilei<x:
        q=0;
        r=0;
        j=0;
        while j<x-1:
                if B[r]>B[j+1]:
                        temp = B[q];
                        B[q]=B[j+1];
                        B[j+1]=temp;
                        r = r+1;
                        q = q+1;
                elif B[r]==B[j+1]:
                        r=j+1;
                else:
                        r=j+1;
                        q=j+1;
                j=j+1;
        m=(r-q)+1;
        x=x-m;
        i=i+1;
        print("-"*30);
        print (B);
```

3. MAGNETIC BUBBLE SORT ALGORITHM

The improvedbubble sort algorithm is very simple to analyse, considering the fact that the time complexity or run time of the algorithm depends on two main factors.

1. Size of list (n)
2. Number of distinct values in the list. *dV*

Run Time = O(n.dV)

Table 1.0 shows the runtime of a set of n values with different number of distinct values.

Table 1.0: Run time of Magnetic Bubble Sort Algorithm (MBSA)

Number of Distinct Values	Run Time	Big-O
1	T = n	O(n)
2	T= 2n	O(n)
3	T=3n	O(n)
...	...	...
n-2	T = (n-2)n	$O(n^2)$
N	$T = n^2$	$O(n^2)$

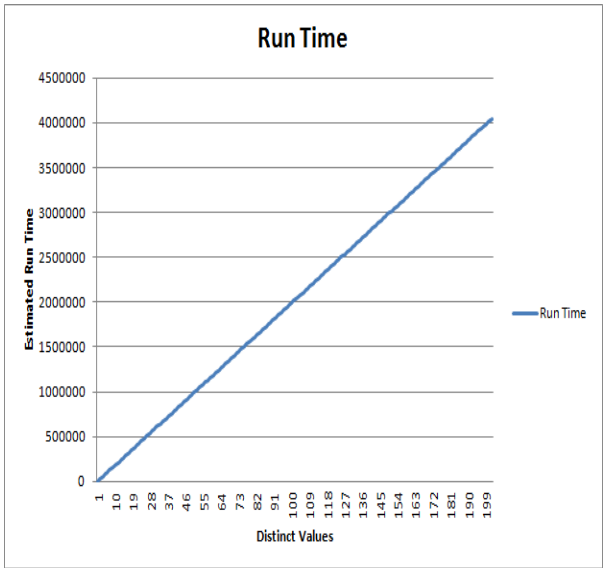

Figure1:illustrates the relationship between distinct values in the list and the running time of the MBSA.

Fig 1: Running time of list of size (n) and number of distinct values using MBSAlgorithm.The performance of the improvedbubble sort could also be enhanced by introducing the FLAG concept in order to terminate sorting when the list is already sorted.

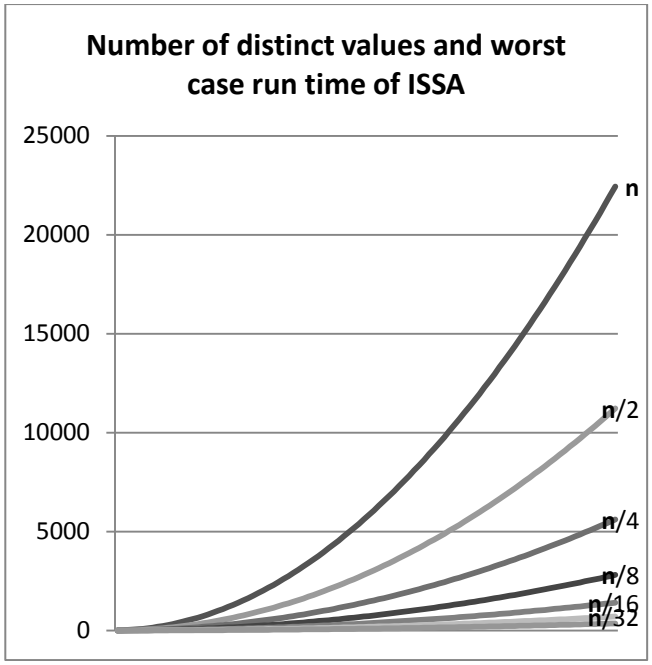

Figure 2: Number of distinct values and run time complexity of MBSA

Fig 2.0 illustrates the relationship between the number of distinct values in a list and the time needed to sort it. The number is illustrated as a ratio of the size of the list (n). If the number of distinct value is half the size of the list, then the algorithm will take about half the time the old bubble sort algorithm takes. From figure 2.0, as the number of distinct values decreases, the running time for the sorting also decreases.

Decreasing distinct values:

$$\frac{n}{1}, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \ldots, \frac{n}{n-2}, \frac{n}{n-1}, 1$$

4. ANALYSIS OF MBAS AND BUBBLE SORTWITH ASAMPLE DATASET

A given set of data of size 1000 wasfinally used to analyse the performances of the proposed sort algorithmand the traditional bubble sort. The number of redundancies in the set was quantified in terms of percentages and 11 different sets of valueswere used to test the algorithms. The data redundancies in set 1 through 11were 0%, 10%, 20%, 30%, 40%,50%, 60%, 70%, 80%, 90%, 100%. Table 2.0illustrates the run times for the various algorithms on the various categories of the dataset.

Table 2. Estimated run times of Bubble Sort and Magnetic Bubble Sort Algorithms when input dataset were not sorted

Redundancies in Percentages	Old Bubble Sort	Magnetic Bubble Sort
'0%	499500	499500
'10%	499500	449550
'20%	499500	399600
'30%	499500	349650

'40%	499500	299700
'50%	499500	249750
'60%	499500	199800
'70%	499500	149850
'80%	499500	99900
'90%	499500	49950
'100%	499500	1000

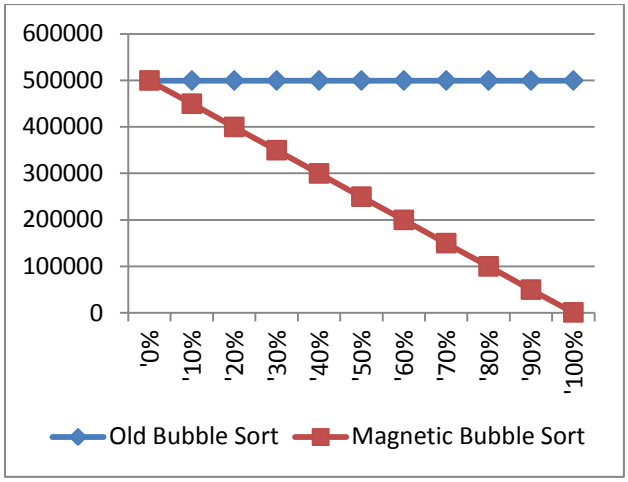

Figure 3.0: Estimated run times of Magnetic Bubble Sort and Old Bubble Sort algorithms when input dataset were not sorted

The observation was made that as the number of redundancies in the list increases, the run time of the proposed algorithm works far better that the traditional or old bubble sort as well ass the cocktail or bidirectional bubble sort algorithm.

5. CONCLUSION

This paper proposed a new bubble sort algorithm which performs better than the existingor the old bubble sort algorithm and in most cases may have a run time in order of *O(n)* which is ideal for sorting relatively large set of data. The strength of the algorithm depends on the distinct values in the list and where there are more of such redundancies or repetitions in the list, it performs better than the existingbubble sort algorithm and also a couple of the optimized bubble sort. In term of space complexity, the algorithm is better than the ISSA which require at least (2n) memory space before the algorithm will work even though its run time performance is almost the same as the proposed bubble sot algorithm. In situation where the number of distinct values is very small, the algorithm may perform better than even the quick sortand merge sort algorithm which have running time of *O(nlogn)*.

6. REFERENCES

[1] Batcher, K. E., Sorting networks and their applications, Spring Joint Computer Conference, AFIPS Proc, 1968

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2001. Introduction to Algorithms. MIT Press. Cambridge. MA. 2nd edition. 2001

[3] "Design and Analysis of Hybrid Selection Sort Algorithm".International Journal of Applied Research and Studies (iJARS) ISSN: 2278-9480 Volume 2, Issue 7 (July- 2013) www.ijars.in

[4] Hayfron-Acquah J. B., Appiah O., Riverson K., Improved Selection Sort Algorithm, International Journal of Computer Applications 01/2015

[5] Jadoon, S., Solehria, S. F., Qayum, M., "Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study" International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 02, 2011

[6] Kapur, E., Kumar, P. and Gupta, S., "Proposal of a two way sorting algorithm and performance comparison with existing algorithms". International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.2, No.3, June 2012

[7] Karunanithi A. K., A Survey, Discussion and Comparison of Sorting Algorithms, June 2014

[8] Khairullah, M. "Enhancing Worst Sorting Algorithms". International Journal of Advanced Science and Technology Vol. 56, July, 2013

[9] Knuth, D. The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0.