# Improving the Magnetic Bubble Sort Algorithm

Dave Rey Magsayo[1], Angelbert Lopena[2], Junar Landicho[3]

[1, 2, 3] Department of Computer Science, University of Science and Technology of Southern Philippines, Cagayan de Oro City, Philippines

magsayo.daverey05@gmail.com, lopena.angelbert@gmail.com, junarlandicho@ustp.edu.ph

## ABSTRACT

In computer science, sorting algorithms are a very essential feature of fast organisation, analysis, and data processing. The Magnetic Bubble Sort Algorithm (MBSA) seems to be a very promising algorithm for huge processing data containing lots of redundancies. In this paper, we want to improve the MBSA by optimising its sorting strategy and exploiting its magnetic properties. The Improved Magnetic Bubble Sort Algorithm (IMBSA) sorts the list with much better performance because it reduces redundant comparisons and swaps. Experimental results show that, in comparison to the MBSA, the IMBSA is much more advantageous concerning processing times and memory usage, especially in cases with varying levels of data redundancy. These advancements offer a more proficient method of sorting large datasets in data-intensive applications.

## General Terms

Sorting Algorithms, Algorithm Optimisation, Last Swapped, Early Termination

## Keywords

Bubble Sort Algorithm, Magnetic Bubble Sort Algorithm, Data Redundancy, Sorting Efficiency, Algorithm Optimisation

## 1. INTRODUCTION

Sorting algorithms are fundamental building blocks in computer science. They enable efficient organisation of data, facilitating analysis and processing tasks [1]. Efficiency and memory consumption by the applied sorting algorithms gain more importance when the sizes of datasets continue to increase. In fields such as data science, machine learning, and big data analytics, it's crucial to rapidly and effectively handle large data volumes [6]. Sorting problems are increasingly inseparable from the use of data and information processing. For data processing, sorting is a basic function [11]. Large datasets running from megabytes to gigabytes or even terabytes need sorting algorithms that efficiently work but also manage their memory to deal with such large datasets.

The magnetic bubble sort algorithm has since emerged as the potential successor, with remarkable performance improvements, especially for data incorporations with redundancies. Its efficiency, though, is yet to be improved, remarkably, by the last swapped sorting strategy and the use of early termination. By reducing these inefficiencies, the Optimised Bubble Sort algorithm promises a more efficient and practical solution for sorting large datasets [5].

This research aims to introduce an improved version of the Magnetic Bubble Sort Algorithm, overcoming the aforementioned drawbacks. It is targeted to significantly increase the efficiency of the algorithm in sorting extremely large data sets with varying degrees of redundancy. Optimisation focuses on reducing the number of needless comparisons and swaps, which are the key causes of the original Bubble Sort algorithm's inefficiency [5]. To that end, we will plan

and implement counter-measures, such as an earlier termination of the MBSA.

The proposed algorithm in this study may be useful to offer an alternative to the existing sorting algorithms for data-intensive applications by improving the Magnetic Bubble Sort Algorithm. It is suggested to improve MBSA, as the presence of redundant information in the dataset can significantly enhance the processing speed and memory usage. Hopefully, fields that are involved in handling immense and complex datasets will find it helpful.

## 1.1 MAGNETIC BUBBLE SORT

The Magnetic Bubble Sort algorithm is an optimised version of the bubble sort with respect to the fact that it finds out the redundancies and straightens the process of sorting without any use of the additional memory spaces for queues or stacks, hence being more space efficient for the huge problem sizes. Yet, the same temporal complexity exists as for the ISSA technique. The MBS resorts to the introduction of a block of values instead of neighbouring comparisons and swaps, that is, a continuous chunk of the list. All members of such a block need to have exactly the same value. Such a block could grow by attracting adjoining elements with the same value-like a magnetic effect. It can be as small as one value and as big as the full length of the list. If the next adjacent value is not equal to that block, then that value is considered demagnetized, and a new block is formed. The block is controlled by two pointers indicating the start and the end of the subset, and the latter expands to include more matching values. It is only between the first item in the block and the adjacent value that a swap needs to be done, which gives the impression that the complete block has been shifted.

The Magnetic Bubble Sort Algorithm sorts content-sensitively, meaning that the runtime is very content sensitive in the distribution of data inside the list. It holds the total number of different values in the list. Provided there are big or equally $\boldsymbol{n}$ different values, the runtime goes near $\mathbf{O}(\boldsymbol{n}^2)$. However, if there are few different values, it can sort them in $\boldsymbol{O(n)}$ time. This kind of content sensitivity adds a dynamic twist to MBS, which allows it to perform rather well on some types of input. The method quickens the sorting action with the help of a magnetic block, is highly space efficient, and, at the same time, retains the same time complexity as that of the traditional bubble sort.

## 1.2 CONCEPT OF LAST SWAPPED

The last swapped idea in the improved magnetic bubble sort algorithm is essentially meant to provide maximum performance through the avoidance of comparisons that are redundant. It incorporates restrictions on the set of elements that are to be verified in each run, only those that may still be unsorted. In particular, the updated swapped index actually delimits the portion of the list in which elements may be out of order, such that the algorithm need not consider that already sorted slice of the list in any future passes. This update of the last position at which a swap was seen gives the algorithm direct bookkeeping of its progress towards sorting the list and dynamically shrinks the range for each pass. This effective tracking guarantees that the computational resources are used only on potentially unsorted parts of the list, thus drastically reducing the number of comparisons and swaps. Also, the last swapped principle is very flexible about the distribution of data

in the set. If the lists are partially sorted, for example, the final swapped index immediately reduces the range of unsorted elements, thereby speeding up the process of sorting. This flexibility also makes enhanced bubble sort very successful with nearly sorted data and imparts a considerable improvement.

Kim [8] delves into related ideas regarding rearranging items through exchanges, within permutation groups, highlighting the significance of handling the scope of elements under consideration for exchanges. This theoretical groundwork supports the enhancements in the bubble sort technique, affirming its efficiency, in minimising computing burdens and enhancing sorting efficiency.

## 1.3 CONCEPT OF EARLY TERMINATION

The FLAG principle can be utilised in enhanced bubble sort, stopping the sorting once the list is sorted [1]. In the optimised bubble sort algorithm, this concept of early termination is used to allow the process to check and terminate the process of sorting once the list is sorted, thereby avoiding unnecessary passes. This theory works on the concept of the fact that, if one runs through the list, no swaps are performed. The list is already in order, and unnecessary passes can be avoided. So, checking for swaps in each run enables the algorithm to find out whether elements were swapped; if none have occurred, it means all elements are in their correct positions and the programme can exit early. This is an optimisation where, once the list is sorted, no more comparisons are required and no more passes need to be made. This has the buffers from wasting computation cycles when unnecessary. This is effective for lists that are near sorted, as it can drastically reduce the number of operations that need to be performed. Also, this makes use of the concept that the sorting process has a dynamic character, so it may respond in real time to the list's status. It allows for optimal performance because the sorting ends as quickly as possible when there is no more work to do.

## 2. RELATED LITERATURE REVIEW

The field of data management significantly depends on the existence of efficient sorting algorithms. They are essential to the organisation of information, particularly large data sets. A case in point is Appiah and Martey, who demonstrate that a good sorting algorithm improves the performance of many applications [1].

One of them is the Magnetic Bubble Sort Algorithm, or MBSA, which can deal with redundant data very successfully. Classic bubble sort has been hugely enlarged by MBSA, which extends it to give substantial gains in run time [1]. The efficiency derived from MBSA is quite valuable when datasets are composed of a large number of disparate values.

Hammad states, By understanding the challenges of time and space for different sorting algorithms, one can gain a more profound insight into the performance of algorithms. The performance characteristics of an algorithm should be weighed against the data distribution and application demands in the process of choosing an algorithm. Quicksort, mergesort, and optimised bubble sorting variants are general choices, and each performs well for a unique situation [2].

Organizing data efficiently is an aspect of computer science, with sorting algorithms

playing a role in this process. Different methods have been created to improve performance and simplify tasks. Sedgewick and Wayne [10] take an approach to computer science, highlighting the significance of algorithms, such, as sorting techniques, when dealing with extensive datasets.

The ever-increasing volume of data, commonly referred to as Big Data, calls for more advanced sorting mechanisms. Chen, Mao, and Liu emphasise that effective algorithms in the processing and analysis of large data volumes make it possible to allow data-driven decision-making at a far larger scale [3].

In the context of Big Data, MBSA's ability to handle redundant data becomes all the more important. According to Batista, Wang, and Keogh, it is a precondition for receiving reliable analysis that the redundant data is managed well [4]. The sorting algorithms, including those by MBSA, sort this purpose well, as do the distance measurements that effectively handle data complexity.

Though generally, the classic variant of the bubble sort is not the most effective, scientists are looking for ways to optimise it. Thomas talks about improvements, that could make a drastic difference in lowering comparisons and swap numbers, thus improving the overall performance [5].

Efficient sorting algorithms are, however, parallel to any data manipulation or processing [9]. MBSA is contrived specifically to deal with redundant data. In very narrow and specialised contexts, this functionality is useful.

Many sorting techniques, including MBSA, have various advantages and disadvantages. Knowing these can help us make better use of them to extract valuable information from data that is growing bigger day by day[7].

## 3. IMPROVED MAGNETIC BUBBLE SORT ALGORITHM

The magnetic bubble sort algorithm is quite a unique method when compared to the other efficient methods of sorting data because it uses the occurrence frequency of different values to sort the available data. This sort algorithm makes an effort to cut down on the time consumed by sorting by utilising the repeat values in the list to avoid the unnecessary comparison of two items of similar magnitudes.

The Magnetic Bubble Sort Algorithm is an algorithm that can be used for sorting out redundant data in datasets through the creation of a magnetic effect. This facilitates the manipulation of data in blocks, which leads to excellent sorting capabilities. Running-time formula,

$$T(n, dV) = O(n \cdot dV)$$

where:

$n$ signifies the overall count of elements within the data collection.

$dV$ represents the total number of unique values contained in the data.

reflects the behaviour of the algorithm, representing a compromise between linear and quadratic complexity depending on the characteristics of the data set.

In order to enhance the performance of the Magnetic Bubble Sort Algorithm (MBSA), we make the following modification: instead of blind comparisons of adjacent items, the algorithm now remembers the location of the last swap. Successive passes through the data set compare only up to this remembered location, reducing many unnecessary comparisons. Also,

after every pass, the algorithm checks if the data set is already sorted. If sorted, the sorting terminates immediately.

The evaluation dataset of both the existing and the proposed optimised MBSA is to be randomly diversified in degrees of data redundancy: 0%, 20%, 40%, 60%, 80%, and 100%. The measurement of the performance of the proposed optimisations to the MBSA would be assessed and compared to the current MBSA, with only one crucial parameter: runtime.

### 3.1 Step-by-Step Process of the Improved Magnetic Bubble Sort Algorithm (IMBSA)

1. The method initialises an unsorted list and a switched variable, which keeps track of the number of swaps, of any-kind-type, that are made in the pass.
2. Then it cycles over the list and swaps adjacent elements if the current element is greater than the next one. Then the elements that are sorted are flagged to be so, and thus the effective size of the list is reduced with respect to the later passes.
3. The traversal and swapping processes are repeated for the other unsorted part of the list. The check of the swapped variable gives information on whether or not any swap was done.
4. The technique, however, tries to use magnetic properties in the sifting to begin equaling or repeating the entries, thus reducing the number of comparisons as well as swapping that is required in the later phases.
5. If there is no swapping in a pass, because of the swapped variable, causes early termination, which means the list has already been sorted and therefore does not have to execute the last computational procedure.
6. The list gets sorted into the desired order, in ascending order, after a few such required passes. These refinements result in much fewer passes than bubble sort, which is standard, and the conventional MBSA too.

Proposed Improvement Implementation in C++// Function to implement the Improved Magnetic Bubble Sort Algorithm

```
procedure improvedMBSort(data) {
  lastSwapped = data.size() - 1;
  sorted = false;

  while (not sorted) {
    swapped = false;
    for (j = 1 to lastSwapped; j++) {
      if (data[j] > data[j + 1]) {
        swap data[j] and data[j + 1];
        swapped = true;
        lastSwapped = j;
      }
    }
    sorted = true;
     // Check if the whole list is sorted after
every iteration
    for (i = 1 to data.size() - 1; i++) {
      if (data[i] > data[i + 1]) {
        sorted = false;
          break;  // Exit inner loop if unsorted
element found
      }
    }
  }
}
```

## 4. RESULTS AND COMPARISON

A comparison in performance was made between the modified magnetic bubble sort method and the original magnetic bubble sort algorithm using a dataset of 1000 elements. The dataset had varying levels of redundancy, represented by different percentages of duplicate values, in six different individual sets. The redundancy ranged between 0% and 100%, incrementing by 20% for every level. Table 1.0 shows the runtime results under test for the two algorithms at various levels of redundancy.

**Table 1.0: illustrates the performance of both the Improved Magnetic Bubble Sort and Magnetic Bubble Sort algorithms across different levels of data redundancies.**

| Data Redundancies(%) | MBSA (µs) | IMBSA (µs) |
|---|---|---|
| 0% | 6142.8 | 0.9 |
| 20% | 6067.1 | 0.9 |
| 40% | 5947.1 | 0.8 |
| 60% | 5865.5 | 0.8 |
| 80% | 5784.3 | 0.6 |
| 100% | 4730.6 | 0.5 |

The performance differences as MBSA and IMBSA subjected to various redundancy levels of data are performed significantly well, as shown in Table below:. When the redundancy level reaches 0%, IMBSA performs excellently well at 0.9 microseconds, compared to MBSA performing at 6142.8 microseconds. This difference indicates the benefits of IMBSA optimisations such as early termination and tracking the last swapped element, which allows IMBSA to detect when the list is sorted and thus can further reduce the comparisons and swaps. As data redundancy approaches 20%, 40%, 60%, and 80%, MBSA improves gradually, but IMBSA always outperforms MBSA by a large margin, simply because MBSA benefits from the reduction of complexity in sorting redundant data. For instance, at 80% redundancy, the time of IMBSA goes down to 0.6 microseconds; however, MBSA's time goes down to 5784.3 microseconds.

The underlying reason behind IMBSA's performance has to do with its efficient handling of data through optimisations. Early termination allows IMBSA to stop sorting when the list is sorted, and last swapped tracking confines the range of each pass to the unsorted portion, hence decreasing unnecessary operations. By contrast, MBSA-it has a more classical bubble-sorting philosophy-will always make a fixed number of passes and comparisons even when the list is partially sorted, making it slower when there are no optimisations.

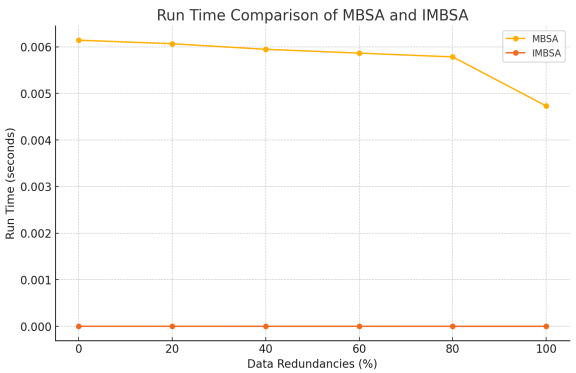


**Figure 1.0 : Estimated run times of Improved Magnetic Bubble Sort and Magnetic Bubble Sort Algorithms with varying data redundancies**

The reason for such a great difference in performance between MBSA and IMBSA actually lies mainly in the optimisation process improved upon by the latter. They really help minimise the number of redundant comparisons and swapping

activities, both of which greatly account for MBSA's ineffectiveness. The IMBSA has an early termination mechanism, which stops sorting once no swap in a pass is found; this considerably reduces duplicate operations.

*Example of Improved Magnetic Bubble Sort (Ascending Order)*

The magnetised set is represented in square brackets [ ].

**Initial List**
4, 2, 5, 2, 4, 2, 4

**Improved Magnetic Bubble Sort Algorithm (IMBSA)**

**1st Pass**
{[4], 2, 5, 2, 4, 2, 4} (lastSwapped = 0)
{2, [4], 5, 2, 4, 2, 4} (No Swap)
{2, 4, [5], 2, 4, 2, 4} (lastSwapped = 2)
{2, 4, 2, [5], 4, 2, 4} (lastSwapped = 3)
{2, 4, 2, 4, [5], 2, 4} (lastSwapped = 4)
{2, 4, 2, 4, 2, [5], 4} (lastSwapped = 5)
{2, 4, 2, 4, 2, 4, [5]}
{2, 4, 2, 4, 2, 4, 5} - check sort [false]

**2nd Pass**
(last Swapped = 5)
// Checks Until Index of 5
{[2], 4, 2, 4, 2, 4, 5} (No Swap)
{2, [4], 2, 4, 2, 4, 5} (lastSwapped = 1)
{2, 2, [4], 4, 2, 4, 5} (No Swap)
{2, 2, 2, [4, 4], 4, 5} (last Swapped = 3)
{2, 2, 2, [4, 4, 4], 5} (No Swap)
{2, 2, 2, 4, 4, 4, 5} - check sort [true]

**Sorted List**
2, 2, 2, 4, 4, 4, 5

Thus, the Improved Magnetic Bubble Sort Algorithm is developed to enhance sorting performance by minimising unnecessary comparisons and swaps, especially in situations where duplicate components exist. It works by offering a more effective method of quick redundant component aggregation for sorting purposes, hence improving the speed. Because of its dynamic adaptability to any data redundancy, the time complexity of IMBSA is almost linear in high redundancy situations and maintains effective sorting times in low redundancy situations.

## 5. CONCLUSIONS

This paper elaborates on the analysis and optimisation of the Magnetic Bubble Sort Algorithm to sort massive sets with outset levels of redundancy. That led to the inception of the Improved Magnetic Bubble Sort Algorithm, tested to sort in a better way as well as have efficient memory space utilisation for highly data-oriented industries like those of data science, machine learning, and big data analytics.

Through our experiments and evaluations, it came to light that the IMBSA works significantly better than the MBSA. With improved magnetic properties, intelligent sorting, and algorithms that eliminate irrelevant comparisons and swaps, the IMBSA has the tendency to sort data more quickly and efficiently with less memory usage. The more significant the degree of redundancy in the data set to be sorted, the more conspicuously advantageous the performance of the IMBSA over traditional sorting techniques is proven. Our comparison of the various levels of redundancy always shows the advantage of the IMBSA, as it is sorted significantly faster regardless of the assessed level of redundancy. This performance makes visible the effectiveness of the optimisations introduced by the IMBSA and shows how properly fitted the IMBSA

is for sorting tasks under a wide range of real-world circumstances.

## 6. LIMITATIONS AND FUTURE WORKS

The limitation of our work is that the Improved Magnetic Bubble Sort Algorithm denotes very high performance gains only in datasets that really have much redundancy. Though it minimises comparisons and swapping, IMBSA is effective only in datasets where the chances are high for redundant data to occur.

Further work, therefore, should be aimed at the better optimisation of this technique for other distributions of data, thereby enhancing its applicability. Also, a close scrutiny of the performance of IMBSA on very large datasets and against other complex algorithms for sorting will give a clearer picture as to the practical applicability of this algorithm. Enhancement of this algorithm's adaptability and tolerance of differing features in data would facilitate all aspects of its performance and applicability to a wide variety of applications.

## 7. REFERENCES:


[1] Appiah, O.,& Martey, E. (2015). Magnetic Bubble Sort Algorithm. *International Journal of Computer Applications*, 122(21), 24-28.

[2] Hammad, J.(2015). A Comparative Study between Various Sorting Algorithms. *International Journal of Computer Science and Network Security*, 15(3), 11.1

[3] Chen, M.,Mao, S., & Liu, Y. (2014). Big Data: A Survey. *Mobile Networks and Applications*, 19(2), 171-209.

[4] Batista, G.E., Wang, X., & Keogh, E. J. (2014). A complexity-invariant distance measure for time series. *Data Mining and Knowledge Discovery*, 28(3), 634-669.

[5] Thomas, P.(2023). Enhanced Efficiency in Sorting: Unveiling the Optimised Bubble Sort Algorithm. *Journal of Robot Automation Research*, 4(3), 424-428.

[6] Aggarwal, C. C. (2015). Data mining: the textbook (Vol. 1, p. 1). *New York: springer*.

[7] Chauhan, Y., & Duggal, A. (2020). Different sorting algorithms comparison based upon the time complexity. *International Journal Of Research And Analytical Reviews*, (3), 114-121.

[8] Kim, D. (2016). Sorting on graphs by adjacent swaps using permutation groups. *Computer Science Review*, *22*, 89-105.

[9] Bao, C., Xu, L., Goodman, E. D., & Cao, L. (2017). A novel non-dominated sorting algorithm for evolutionary multi-objective optimization. *Journal of Computational Science*, *23*, 31-43.

[10] Sedgewick, R., & Wayne, K. (2016). *Computer science: An interdisciplinary approach*. Addison-Wesley Professional.

[11] Zhu, Z. G. (2020, May). Analysis and Research of Sorting Algorithm in Data Structure Based on C Language. In *Journal of Physics: Conference Series* (Vol. 1544, No. 1, p. 012002). IOP Publishing.