Enhancement of Super Sort Sorting Algorithm Using Identical Value Search Concept

Jeanus Marco Caraig[1], Nicole Anne Marie Cruz[1], Vivien Agustin[1], Khatalyn Mata[1], Dan Michael Cortez[1]

[1] jmlcaraig2018@plm.edu.ph
[1] namocruz2018@plm.edu.ph
[1] vaagustin@plm.edu.ph
[1] kemata@plm.edu.ph
[1] dmacortez@plm.edu.ph
[1] Computer Science Department, College of Engineering and Technology
Pamantasan ng Lungsod ng Maynila (University of the City of Manila)
Intramuros, Manila 1002, Philippines

Abstract

The Super Sort Sorting algorithm is a sorting algorithm that uses forward selection, backward selection, merging, and partition to sort a given list or data set. This study will focus on the enhancement of the Super Sort Sorting Algorithm on the forward selection part. This paper introduces an enhanced version of the algorithm where the concept of Identical Value Search was implemented on the first forward pass. This method eliminated the small sublists left behind by the original algorithm's forward pass and reduced the overall runtime of the algorithm.

Keywords: Super Sort Sorting Algorithm; Sorting Algorithm; Sorting; Identical Value Search

1. Introduction

As we live in a digital age where the data and information are abundant and posted on the internet, it is hard to arrange and keep this information orderly. Sorting is a method of arrangement of sorts to keep the items in an orderly manner according to the preferred order of the person. On the other hand, algorithms are a step-by-step process on how to solve a problem to obtain the desired result in the end. With these two ideas combined, sorting algorithms emerged and can be defined as algorithms focused on sorting items with the help of computer processes. An example of a sorting algorithm is Super Sort Sorting Algorithm developed in 2018 by Yash Gugale. This newly proposed sorting algorithm uses forward and backward selection passes, which is the key point or the unique feature of the algorithm, as this is done to compensate for the absence of double-checking in a sorting algorithm. Moreover, these passes will produce unsorted lists, which will undergo a merge sort loop until the whole initial list is sorted. Furthermore, it is said that the Super Sort Sorting algorithm can handle large batches of data.

Although, the Super Sort Sorting algorithm may have some setbacks, such as the production of multiple unsorted lists and the production of many sublists during merge passes, which both affect the costs and the run time of the algorithm. The traversal of sorting among the elements present in the list utilized the concept of comparison of each element with the highest element value. If run with random datasets, the traversal is said to be unreliable since it may cause an increase in the creation of sublists. This study aims to identify the vulnerabilities of the Super Sort Sorting Algorithm and how it can further be optimized and enhanced.

2. Related Studies

Sorting is a method used to arrange items in a specific order. It is used to impose and maintain order in the list. In the Yudav & Gupta (2021) study, sorting is a method for sorting the initial list of given elements and arranging them in the desired order, whether in ascending or descending order. This principle led to the invention of sorting algorithms. These sorting methods are used to sort the list's elements. Because there are so many different ways to sort items or elements in a list, sorting algorithms have become quite diverse. Bubble Sort, Insertion Sort, Merge Sort, and Quick Sort are just a handful of the options. After taking into account computer-dependent variables like internal sorting (sorting based on internal memory), external sorting (if sorting requires a secondary memory), and device complexity (time, space, and computational complexities), Yudav and Gupta (2021) classified Merge Sort, Insertion Sort, and Bubble Sort as stable algorithms. Quick Sort, Heap Sort, and Selection Sort are all unstable algorithms in the meantime.

Although there are many ways on how a list can be sorted, which paved the way for the emergence of variations of the sorting algorithms, the following are the most used in the industry: Merge Sort, Insertion Sort, Bubble Sort, Quick Sort, Heap Sort, and the Selection Sort. First on the list is the Merge Sort. The Merge Sort is based on the concept of dividing and conquering, in which two sorted arrays will be sorted into a single one. In the process of the Merge Sort, the initial list will be divided into two (2) parts, where the total number of elements will be divided into two (2). These two (2) parts will be divided into sorted sublists; afterward, they will be merged again to obtain the final sorted list.

Meanwhile, the Insertion sort is based on the idea of starting with one element and inserting one element at a time. It uses a card game concept where the players put one card at a time into the stack. The Insertion Sort can be divided into two (2) parts: the linear search and the binary search. The difference between these two searching methods is that the linear uses a unitary method. In contrast, the binary search uses the idea of dividing two (2) separate sublists to further search the element. Bubble sort, on the other hand, two consecutive elements in the list will be compared and be swapped if necessary, depending on the desired arrangement, whether it is in ascending or descending order. There is a traversal happening on each element in the list; thus, the comparison operation on each element is done and will yield the sorted list. The Quick Sort is a divide-and-conquer strategy that utilizes a "pivot" in the array of elements. When the pivot is selected, the QuickSort will produce a list where on the left side of the pivot will be the lesser values than the pivot value, and on the right side are the higher than or equal values to the pivot value. Next, the Heap Sort uses the idea of comparison to sort the elements of the list. The heap sort is like a binary tree, and there is a max heap is a maximum element placed on the root, and the lesser values will be the leaves. Lastly, the Selection Sort is based on the idea of selecting whether it is the maximum or minimum element and will store it in its respective position. It will repeat the same steps until one element remains on the array.

In Computer Science, many people use different sorting algorithms as they are many approaches or ways to sort a list at hand, thus resulting in the generation of many sorting algorithms. It resulted in the emergence of many hybrids or new ideas of sorting to sort the elements faster and more accurately. The Super Sort Sorting algorithm is a unique sorting algorithm that takes a "divide-and-conquer" strategy to sort. The algorithm is notable for its forward, and backward selection passes with multiple merge passes to create the sorted list.

Yash Gugale (2018) explained how the Super Sort Sorting algorithm works by stating that it is based on selecting the order of the initial sorted components in an unsorted list. The unsorted list is then traced using forward, and backward selection passes, and any beginning entries that remain are appended to the unsorted list. From the center, the unsorted list will be partitioned, with each sublist performing forward and selection passes. If any unsorted elements remain on the list, the same technique will be used – partitioning and forward and backward selection passes – until all of the elements have been sorted. The sublists will be combined to create larger sublists.

Since the Super Sort Sorting algorithm is based on the Quick Sort and the Merge Sort concepts, both the Quick Sort and Merge Sort use the idea of the divide-and-conquer approach. The two sorting algorithms' concepts are very similar yet have distinctive differences. The Merge Sort uses the division of an array into two (2) separate parts; the same concept is applied to the Quick Sort. Still, instead of dividing the array from the middle, it uses the concept of selection of a pivot element and performs the partitioning based on the pivot element selected. Another difference between the two sorting algorithm concepts is that the Merge Sort uses the idea of breaking down the array into smaller sublists and using multiple merge passes to create the final sorted list. While the Quick Sort does not require breaking the array into smaller sublists, it will revolve around the idea of comparing each element value to the pivot element value. It will sort them based on the preferred sorting using partitioning. According to Taiwo et al. (2020) study, the authors differentiated the Merge Sort and Quick Sort based on how they perform their sorting and use recursive sorting to produce the final sorted list. The authors noted that the Quick Sort works faster but is only limited to arrays with a smaller number of elements or data.

Meanwhile, the Merge Sort is slower than the Quick Sort but is ideal for arrays with more elements or data. Also, Merge Sort needs additional memory space to store extra arrays and sublists, while the Quick Sort needs space. The authors concluded that Quick Sort is the ideal sorting in smaller batches of data, while in large batches of data is Merge Sort. The authors also noted that the programmers should first be aware of how much data they will be dealing with and what type of sorting method they want to implement so that the programmers may know the advantages and disadvantages of their preferred algorithm.

There have been studies that employ the Super Sort Sorting method, including one that uses MPI and CUDA to implement the algorithm. The authors of this study used parallel programming to perform sorting and merging tasks simultaneously. According to this study, the Super Sort Sorting algorithm with CUDA takes longer than sequential and MPI sorting for small input sizes. On the other hand, CUDA outperforms both sequential and MPI counterparts when dealing with huge input sizes. The time taken is gradually raised rather than the abrupt increase seen in a sequential program. This study discovered that parallel programming with MPI and CUDA works effectively for huge input sizes when utilizing the Super Sort Sorting algorithm.

Another study applied the Super Sort Sorting algorithm to the cyclic amplitude data for Rotation Angle Estimation of JPEG Compressed Image by Cyclic Spectrum Analysis. In a JPEG compression situation, the Super Sort Sorting approach minimized rotation angle estimation inaccuracy and successfully recognized and located tampering; however, it may be improved in the future.

Nevertheless, the Super Sort Sorting algorithm exhibits some problems or limitations, one of which is the production of multiple sublists and breaking them into smaller sublists to undergo merge passes. This problem can cause an increase in time performance complexity. According to Yerram and Bhonagiri (2020), the greatest disadvantage of the Super Sort Sorting algorithm is that it increases the cost of execution when combining small sublists with fewer components. Another study by Lawrence (2021) mentioned that "the disadvantage is that multiple merge passes may be required with costly writes," which is present in the Super Sort Sorting algorithm. This algorithm requires multiple merge passes for sorting. Moreover, another study by (Sara, M.R.A, et al., 2019) mentioned that the use of an extra array [list] puts merge sort in a disadvantageous position, as it takes some extra time to copy the elements to and from the extra array [list]. Furthermore, the algorithm uses the highest current value method for traversing, thus producing sublists for the same value but different positions in the list.

3. Existing Super Sort Sorting Algorithm

3.1. Overview

The Super Sort Sorting algorithm utilizes the concept of forward and backward selection passes for the sorting and traversal of the algorithm. The algorithm will produce a forward and backward selection that passes sublists containing the current highest element and the elements with a higher or equal value on the current highest element during its traversal.

Figure 1. Selection Pass

```
if L[i] >= current_highest then
        current_highest ←L[i]sortedlist.append(current_highest)
        L.remove(current_highest)
```

Figure 1 shows how the appending of the elements to the sorted list may be the backward or the forward sorted list. Both passes use the same concept on how they append the values on each sorted list; the only difference is that the forward pass starts from the outer leftmost side. Meanwhile, the backward sorted pass begins from the outer rightmost side. If elements are left behind on the forward and backward passes, the algorithm will produce another unsorted sublist. The sublist will be partitioned into two (2) parts, and the forward and backward selection passes will be performed as well. A merge sort will be performed to combine these sublists to produce the final sorted list.

3.2. Problem Statement

The Super Sort Sorting algorithm is a novel sorting algorithm that uses forward, and backward selection passes for sorting, along with merge steps to produce the final sorted list. Upon examination, the researchers found some setbacks or problems encountered in the Super Sort Sorting algorithm. The algorithm produces too many sublists, especially when it encounters the same elements in the list, resulting in additional costs. The Super Sort Sorting algorithm uses forward, and backward selection passes to sort the original list of elements initially, and the remaining elements will be divided into smaller sublists. The whole algorithm will undergo multiple merge passes until the final sorted list is produced. This would result in many unsorted sublists, which are costly to the system. Below is a visualization example of the problem that occurs.

Figure 2. Problem Visualization

6 4 2 1 2 1

//FORWARD SELECTION PASS

6 = HIGHEST CURRENT VALUE

TRAVERSION	FORWARD SORTED LIST
6 > = 6	6
6 > = 4	
6 > = 2	
6 > = 1	
6 > = 2	
6 > = 1	

//BACKWARD SELECTION PASS

1 = HIGHEST CURRENT VALUE

TRAVERSION	BACKWARD SORTED LIST
1 > = 1	1
1 > = 2	2
2 > = 1	
2 > = 2	2
2 > = 4	4

UNSORTED LIST: 1

In figure 2, it can be seen that even if the fourth (from left to right) element 1 is another generated sublist, which is an additional costly write. The production of a sublist is regarded as a costly write which can put the merge sort in a disadvantageous position as it requires it again to write another sublist. From the concept of the Super Sort Sorting algorithm that uses the Merge Sort to merge the generated sublists to create the final sorted list, this could greatly affect the algorithm's performance as it may slow down the time performance of the CPU that runs the algorithm.

3.3. Pseudocode of Super Sort Sorting Algorithm

Check if the list > 1 then
    Generate two empty lists: the forward sorting list and the backward sorting list
        //Forward Selection Pass
            Take the first value on the outer left side most of the list then compare to the next element, the traversal of values will be from left to right
                Append the first value which is the current highest to the forward sorted list
                    If the next element has a higher value or equal value, append it to the forward sorted list
                    Else, skip the element and retain on the list, and move on to the next value

Repeat the process until all elements in the list are traversed
//Backward Selection Pass
Take the first value on the outer right-side most of the list then compare to the next element, the traversal of values will be from right to left
Append the first value which is the current highest to the backward sorted list
If the next element has a higher value or equal value, append to the backward sorted list
Else, skip the element and retain on the list, and move on to the next value
Repeat the process until all elements in the list are traversed
If there are elements that remained on the initial list, create a new sublist to contain, which will be called as unsorted list
Divide or partition the unsorted list into half
Divide the sublists from the unsorted list into smaller sublists
Perform forward and backward selection passes to the smaller sublists
Perform a merge sort to all the sublists, including the forward and backward sorted lists

4. Methodology

4.1. Enhancement of the Algorithm

To solve the problem of producing multiple unsorted lists from the first forward pass, we have implemented an Identical Value Search function on the existing Super Sort Sorting Algorithm. After the algorithm picks up the first value as the current highest value, it will search the list for identical values before traversing the entire dataset, eliminating the small, unsorted lists that come from skipped values due to multiple changes in the current highest value. The researchers used an experimental design to investigate whether the application of the Identical Value Search function would be beneficial to the algorithm.

4.2. Pseudocode of the Enhanced Super Sort Sorting Algorithm

Check if the list > 1 then
Generate two empty lists: the forward sorting list and the backward sorting list
//Forward Selection Pass
Take the first value on the outer left side most of the list then compare to the next element, the traversal of values will be from left to right
Append the first value which is the current highest to the forward sorted list
Search the list for values identical to the current highest element
If the next element has a higher value **~~or equal value~~**, append it to the forward sorted list
Else, skip the element and retain on the list, and move on to the next value
Repeat the process until all elements in the list are traversed
//Backward Selection Pass
Take the first value on the outer right-side most of the list then compare to the next element, the traversal of values will be from right to left
Append the first value which is the current highest to the backward sorted list
If the next element has a higher value or equal value, append to the backward sorted list
Else, skip the element and retain on the list, and move on to the next value
Repeat the process until all elements in the list are traversed
If there are elements that remained on the initial list, create a new sublist to contain, which will be called as

unsorted list
Divide or partition the unsorted list into half
Divide the sublists from the unsorted list into smaller sublists
Perform forward and backward selection passes to the smaller sublists
Perform a merge sort to all the sublists, including the forward and backward sorted lists

4.3. Identical Value Search Function

The researchers came up with searching the list before the first forward pass occurs; hence the Identical Value Search function is created. The identical value search function eliminates the need to produce unsorted lists further. The IVS function takes in the current highest value, traversing the list, searching for identical values inside the list, and appends them to the first forward sorted list. Searching for identical values with the current highest value will eliminate the production of unsorted lists produced by skipping them due to the early change in the current highest value of the original algorithm.

5. Results & Discussion

The original algorithm and the enhanced version of the algorithm have been tested with the same data set, which highlights the problem of the original Super Sort Sorting algorithm. The program was executed on an Intel Core i5 processor @ 2.3GHz with 12GB of DDR4 RAM running Windows 11 Home Single Language 64-bit. The algorithms present in this study were implemented in Python 3.9.

Table 1. Algorithm runtime comparison

Sorting Algorithm	Average Time to Sort (ms) in 5 different datasets	Variance
Original Super Sort Sorting Algorithm	11.04	6.328
Enhanced Super Sort Sorting Algorithm	6.26	6.783

Table 1 shows that the Enhanced Super Sort Sorting Algorithm needed less time to sort the dataset than the original Super Sort Sorting algorithm. This result was possible due to the reduction in the amount of sublists created during the overall runtime of the program. A t-test is conducted to prove the difference between the average runtime of the Original Super Sort Sorting algorithm and the Enhanced Super Sort Sorting algorithm. The t-score calculated was 7.175, which proves at 95% confidence level that there is a significant difference between the average runtime of the two algorithms.

Table 2. Algorithm sublist creation

Sorting Algorithm	Approximate Sublists Created (20 values)
Original Super Sort Sorting Algorithm	23
Enhanced Super Sort Sorting Algorithm	18

The implementation of the Identical Value Search function also aimed to reduce the number of sublists being created during runtime. This was expected to reduce the memory load during the algorithm's runtime. The reduction in the amount of sublists created is shown in Table 2.

6. Conclusion

In this paper, the researchers proposed a modification of the original Super Sort Sorting algorithm. The Identical Value Search function is implemented in the algorithm, resulting in better runtime and reducing the number of unsorted lists produced. Implementing the Identical Value Search function resulted in a more efficient running of the algorithm since it searches the current highest value in the list despite its position in the list. With this enhancement, the enhanced Super Sort Sorting algorithm was able to find the same value of the current highest and store it in the respective selection pass sorted list. This resulted in the reduction of the unsorted elements in the initial list. This shows that the method implemented has successfully enhanced the algorithm's overall performance.

For future works, the researchers recommend implementing the algorithm in C or C++ or other languages of your choice to enhance the speed of execution and overall runtime.

Acknowledgments

First and foremost, the researchers would acknowledge and thank God for guiding and enlightening them during their journey. Their families for their unwavering love and support. To their professor, Ma'am Vivien Agustin, for her counsel as their thesis adviser. Dr. Dan Michael Cortez and Dr. Khatalyn Mata, their other academics, for their guidance as the researchers' thesis coordinators. To the faculty and staff of the Pamantasan Ng Lungsod Ng Maynila Computer Science Department under the College of Engineering and Technology for their support and encouragement.

References

Dai, S., Zhang, Y., Song, W., Wu, F., & Zhang, L. (2019). Rotation Angle Estimation of JPEG Compressed Image by Cyclic Spectrum Analysis. Electronics, 8(12), 1431.

Gugale, Y. (2018, April). Super sort sorting algorithm. In 2018 3rd International Conference for Convergence in Technology (I2CT) (pp. 1-5). IEEE.

Lawrence, R. (2021, March). Adaptive flash sorting for memory-constrained embedded devices. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (pp. 321-326).

Pereira, S. D., Ashwath, R. B., Rai, S., & Kini, N. G. (2021). Super sort algorithm using MPI and CUDA. In Intelligent Data Engineering and Analytics (pp. 165-170). Springer, Singapore.

Sara, M. R. A., Klaib, M. F., & Hasan, M. (2019). EMS: AN ENHANCED MERGE SORT ALGORITHM BY EARLY CHECKING OF ALREADY SORTED PARTS. International Journal of Software Engineering and Computer Systems, 5(2), 15-25.

Taiwo, O. E., Christianah, A. O., Oluwatobi, A. N., & Aderonke, K. A. (2020). Comparative Study of Two Divide and Conquer Sorting Algorithms: Quicksort and Mergesort. Procedia Computer Science, 171, 2532-2540.

Yadav, R., Yadav, R., & Gupta, S. B. (2021). Comparative study of various stable and unstable sorting algorithms. In Artificial Intelligence and Speech Technology (pp. 463-477). CRC Press.

Yerram, B., & Bhonagiri, J. K. (2020, July). An Efficient Sorting Algorithm for binary data. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-4). IEEE.