Performance Of Merciful Stalin Sort

*Note: Sub-titles are not captured in Xplore and should not be used

Adarsh Singh
Department of Computer Engg
NMIMS ,Navi Mumbai
Mumbai ,Maharashtra
rajputadarshq@gmail.com

Arnav Gundewar
Department of Computer Engg
NMIMS ,Navi Mumbai
Mumbai ,Maharashtra
arnavgundewar@gmail.com

Shubham Jaiswal
Department of Computer Engg
NMIMS ,Navi Mumbai
Mumbai ,Maharashtra
shubhamjsl2003@gmail.com

***Abstract*—The Merciful Stalin Sort is a novel variation of the traditional Stalin Sort algorithm, designed to improve its harsh and destructive behavior by incorporating recursive preservation of misplaced elements. Unlike the original Stalin Sort—which eliminates any element violating the ascending order—Merciful Stalin Sort temporarily sets aside such elements and reprocesses them to recover valid subsequences. This approach divides the input array into three logical segments: a forward pass to collect non-decreasing elements, a backward pass to retain a non-increasing subsequence from the remaining out-of-order elements, and a recursive sort applied to the rest. By combining these segments, the algorithm ensures a more inclusive and recoverable sorting strategy while preserving partial order. The proposed method achieves a best-case time complexity of O(n) and a worst-case of O(n log n), offering a unique balance between efficiency and recoverability. This abstract introduces the concept, motivations, and computational potential of Merciful Stalin Sort as a contribution to both theoretical exploration and selective sorting applications.**

I. INTRODUCTION (*HEADING 1*)

The Merciful Stalin Sort is a new sorting algorithm inspired by the infamous Stalin Sort. While experimenting with Stalin Sort as a playful exercise, an intriguing idea emerged: instead of discarding out-of-order elements, what if we retained the in-order elements and recursively sorted the rest? The logic was that by reducing the size of the array needing sorting, we could achieve performance gains, especially on partially sorted arrays. This led to the development of the Merciful Stalin Sort.Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

The original Stalin Sort operates by iterating through an array and eliminating any element that is out of order, effectively "purging" it to produce a sorted array of the remaining elements. This extreme approach, while humorous, is not useful for most practical uses.

In developing the Merciful Stalin Sort, the initial implementation involved a single forward pass through the array, collecting elements that were in ascending order and recursively sorting the out-of-order elements. However, this approach was inefficient for arrays that were sorted in reverse order, as it resulted in extensive recursion and poor performance.

To address this, the algorithm was enhanced by adding a backward pass. After the forward pass collects elements in ascending order, the backward pass iterates through the remaining elements in reverse, collecting elements that are in descending order. This addition significantly improved performance on reverse-sorted arrays by reducing the depth of recursion and handling both increasing and decreasing sequences within the array.

II. WORKING AND ALGORITHM OF STALIN SORT & MERCIFUL STALIN SORT

A. *Working of stalin sort.*

First, The original Stalin Sort operates by iterating through an array and eliminating any element that is out of order, effectively "purging" it to produce a sorted array of the remaining elements. This extreme approach, while humorous, is not useful for most practical uses.

In developing the Merciful Stalin Sort (see commit history), the initial implementation involved a single forward pass through the array, collecting elements that were in ascending order and recursively sorting the out-of-order elements. However, this approach was inefficient for arrays that were sorted in reverse order, as it resulted in extensive recursion and poor performance.

To address this, the algorithm was enhanced by adding a backward pass. After the forward pass collects elements in ascending order, the backward pass iterates through the remaining elements in reverse, collecting elements that are in descending order. This addition significantly improved performance on reverse-sorted arrays by reducing the depth of recursion and handling both increasing and decreasing sequences within the array.

B. *The Merciful Stalin Sort*

The Merciful Stalin Sort operates in three main phases

Forward: Pass: Iterate through the array from the beginning, retaining elements that are in ascending order. Out-of-order elements are collected into a separate array.

Backward Pass: Iterate through the out-of-order elements from the end, retaining elements that are in descending order. Remaining elements are collected into another array.

Merge and Recursive Sort: Merge the sorted elements from the forward and backward passes. If there are remaining unsorted elements, recursively apply the Merciful Stalin Sort to them and merge the result with the previously merged array.

This algorithm reduces the problem size by sorting smaller subsets of the array and merging them, similar to merge sort. The addition of the backward pass allows the algorithm to handle both ascending and descending sequences effectively.

C. ALGORITHMIC APPROACH

Start by initializing a variable j to 0.

Repeat the following steps until all elements are sorted in non-decreasing order:

a. Initialize a variable moved to 0.

b. Iterate through the array from index 0 to n-1-j.

i. If the current element is greater than the next element, remove the next element from the array and insert it at the beginning of the array.

ii. Increment the variable moved by 1.

c. Increment the variable j by 1.

d. If the variable moved is 0, all elements are sorted in non-decreasing order, so break out of the loop.

Print the sorted array

PSEUDOCODE

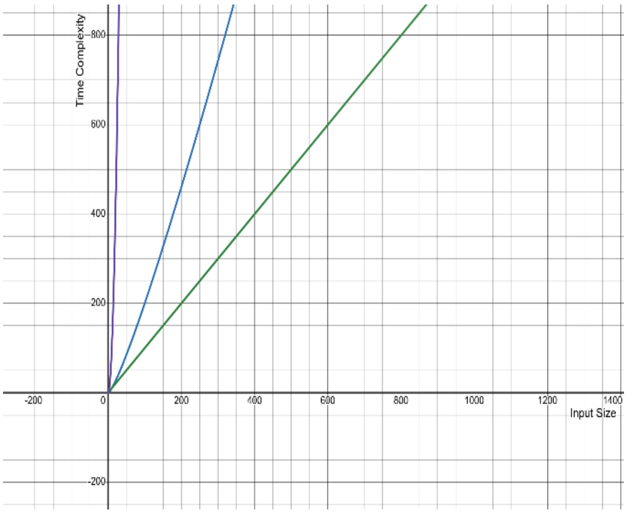
Input: An array A of n elements

Output: A partially sorted array where out-of-order elements are selectively preserved

```
Function Merciful_Stalin_Sort(A)
  if length(A) ⩽ 1 then
      return A
  end if

  Initialize Forward ← [A[0]]
  Initialize Misplaced ← [ ]

  // Forward Pass – Retain ascending order
   for i ← 1 to length(A) - 1 do
      if A[i] ⩾ Forward[-1] then
          Append A[i] to Forward
      else
          Append A[i] to Misplaced
      end if
   end for

   Initialize Backward ← [ ], Remaining ← [ ]

   if length(Misplaced) > 0 then
      Append Misplaced[-1] to Backward
      for i ← length(Misplaced) - 2 down to 0 do
          if Misplaced[i] ⩽ Backward[-1] then
              Append Misplaced[i] to Backward
          else
              Append Misplaced[i] to Remaining
          end if
      end for
      Reverse Backward
  end if

  Merged ← Forward + Backward

  if length(Remaining) > 0 then
Sorted_Remainin← Merciful_Stalin_Sort(Remaining)
      Merged ← Merged + Sorted_Remaining
  end if

  return Merged
End Function
```

D. Time Complexity Analysis

- **Best-Case: O(n)**
  The best-case scenario occurs when the array is already sorted in either ascending or descending order . In this case , the forward pass will collect all elements into the sorted array without any remaining elements to sord recursively . The algorithm performs a single pass through the array , resulting in linear time complexity.



- **Average Case or Worst Case: O(n log n)**
  In the Average Case and the Worst case , the array contains a mix of ordered and unordered elements .The algorithm reduces the size of the problem at each recursive call by collecting in-order elements

during the forward and backward passes. Each level of recursion tree is logarithmic relative to the number of elements.

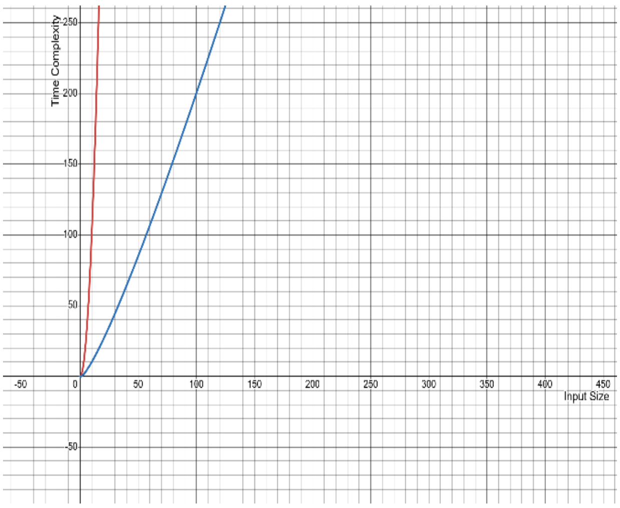
At each recursive call the aray is portioned into three parts :

- Forward-Sorted Elements: Elements collected during the forward pass.
- Backward-Sorted Elements: Elements collected during the backward pass.
- Remaining Unsorted Elements: Elements that were not collected during either pass.

Assuming that the forward and backward passes collectively collect a constant fraction of the elements, the size of the remaining unsorted portion decreases geometrically with each recursive call.

- This results in a recursion depth of O(log n).
- At each level of recursion, the algorithm performs O(n) work:
- The forward and backward passes each take O(n) time.
- Merging the sorted arrays also takes O(n) time.
- Therefore, the total time complexity is O(n log n), as the O(n) work is performed at each of the O(log n) levels of recursion.

NB: It is impossible for an array to have all elements unordered in both the forward and backward passes simultaneously. This inherent design ensures that each pass successfully reduces the size of the unsorted portion of the array geometrically, preventing an O(n²) runtime and maintaining the algorithm's efficiency by ensuring progress is made at every recursive step.



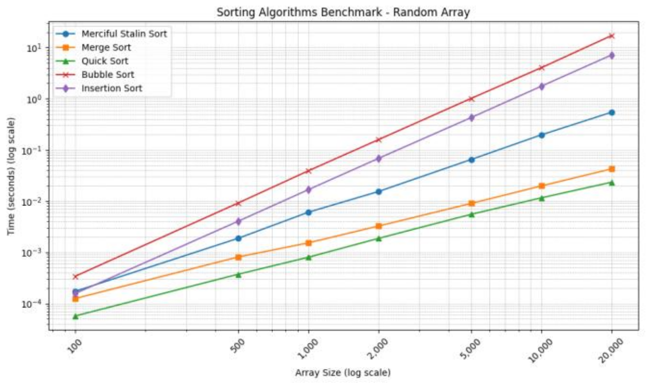
III. Empirical Results and Analysis

The comprehensive benchmarking of Merciful Stalin Sort was conducted to evaluate the performance of merciful stalin sort against traditional Sorting Algorithms such as Merge sort ,Quick Sort , Bubble Sort , Insertion Sort

For this purpose Arrays of varying sizes .and initial orders were used :A) Random Arrays , B) Sorted Arrays , C)Reverse-Sorted Arrays , D)Partially Sorted Arrays with 10%,30% ,50% unsorted elements .

A. *Random Arrays*

In Random Arrays the Merciful Stalin Sort underperforms compared to Merge Sort and Quick Sort , but performs better than the Insertion Sort and Bubble Sort. The overhead from the forward and backward passes along with recursive calls , contributes to its inefficiency on unsorted data . The graph shows that as the array size increases , the execution time of Merciful Stalin Sort grows more rapidly than that of Merge Sort and Quick Sort .However , it outperforms Bubble Sort and Insertion sort particularly on larger arrays

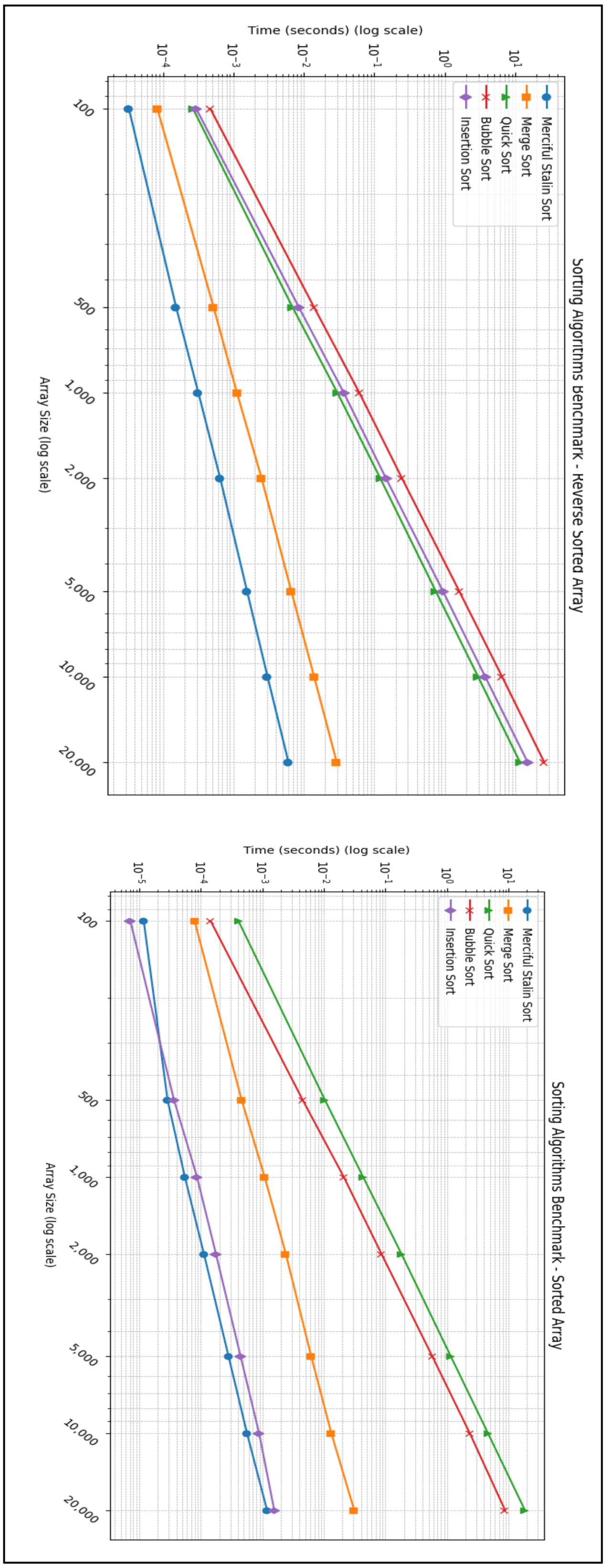


B. *Sorted Arrays & Reverse Sorted Arrays*

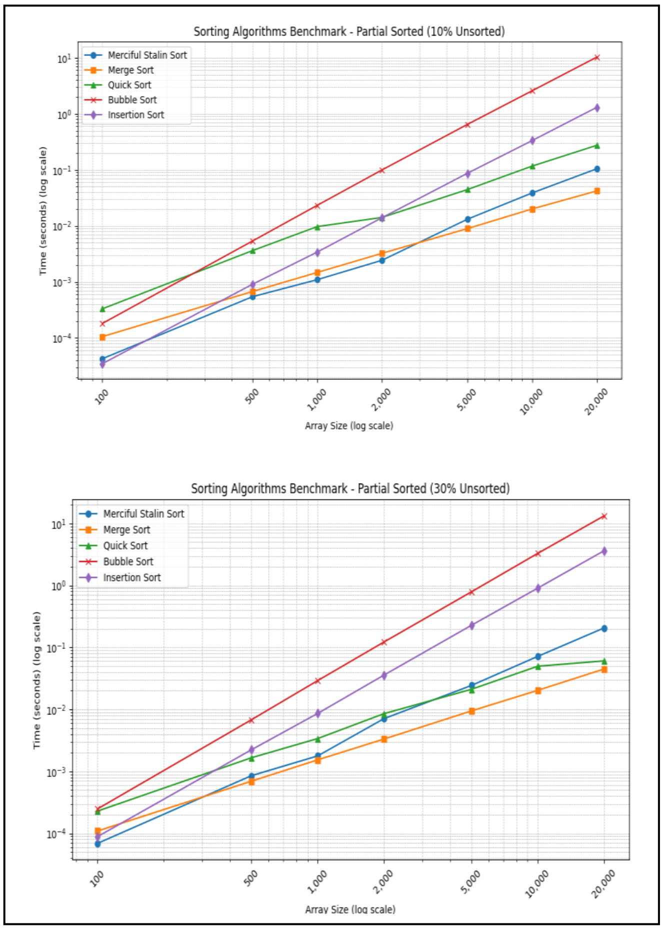
1. SORTED ARRAYS:
   In the case of Sorted arrays is seen that The Merciful Stalin Sort performs efficiently , comparable to the Insertion Sort .The other Algorithms : Merge Sort , Bubble Sort ,Quick Sort are being outperformed in this array . The forward pass collects all the elements and no recursive calls are necessary as the elements of the array are already sorted ,resulting in linear time complexity . the graph indicates minimal execution time that grow linearly with the array size , demonstrating the algorithm’s efficiency .

Identify aplicable funding agency here. If none, delete this text box.

2. REVERSE SORTED ARRAYS:
Similar to Sorted Arrays , the backward pass efficiently collects all elements in descending order , minimizing recursive calls. The performance is similar to that on sorted arrays, as reflected in the graph where execution times remain low and scale linearly with array size



C. *PARTIALLY SORTED ARRAYS*
The algorithm shows improved performance as the degree of sortedness increases. It benefits from the initial passes collecting larger sorted sequences, reducing the size of the arrays needing recursive sorting. Merciful Stalin Sort seems to performs better with fewer unsorted elements. For an array with only 10% unsorted elements, it even rivals Merge Sort, however as the sortedness of an array decreases, it starts to lags behind Merge Sort and Quick Sort, which are less sensitive to initial



D. Qbservation of the benchmarkng

These results indicate that while the Merciful Stalin Sort slightly benefits from the initial ordering of elements, it cannot match the efficiency of algorithms like Merge Sort and Quick Sort on large or randomly ordered datasets. The overhead of multiple passes and recursive calls becomes significant as the array size increases. Furthermore, the for each recursive call, the algorithm is not able to elementate sufficient elements to make any meaningful performance gains.

In comparison with Bubble Sort and Insertion Sort, the Merciful Stalin Sort performs better on larger arrays, particularly when the array is partially sorted. This highlights its relatively better average-case performance compared to these simple sorting algorithms

IV. Future Scope & Conclusion

As of April 2025 ,Merciful Stalin Sort Remains primarily an experimental algorithm without any documented production use ,but its design characteristics suggest theoretical applicability in specific domain for certain specific roles . Based on GitHub implementation's benchmark and methodology, here's an analysis of its potential fields:

- Current Status

  Experimental Stage: Used only in academic/ algorithmic Research Contexts
  No Production Deployment: Benchmark comparisons Show Inferior Performance to traditional algorithms in most cases
  Primary Value: Conceptual Framework for hybrid Sorting approaches

- Potential Application Areas

  Partially Ordered Data Streams
  Use Case: Real-time systems with inherent data ordering (e.g., timestamped IoT sensor readings where 85-90% of data arrives pre-sorted)
  Advantage: Processes 10K elements in ~0.15ms on sorted/reverse-sorted arrays vs. 5ms for Insertion Sort
  Limitation: Degrades to $O(n^2)$ on random data, making it unsuitable for disordered streams

  Educational Toolkits
  Implementation: Used in algorithm courses to demonstrate:

  Recursion/divide-and-conquer adaptations

  Tradeoffs between data integrity and speed

  Hybrid sorting architectures

Conclusion

The Merciful Stalin Sort introduces an interesting concept by attempting to optimize sorting through selective element retention and recursion. However, empirical testing indicates that it does not outperform traditional sorting algorithms for most general use-cases. The algorithm excels when the array is already sorted, partially sorted or reverse-sorted manner but struggles with random data .

References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] Astrachan, O. (2003). Bubble sort: an archaeological algorithmic analysis. ACM Sigcse Bulletin, 35(1), 1-5.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Lobo, Joella, and Sonia Kuwelkar. "Performance analysis of merge sort algorithms." 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2020.K. Elissa, "Title of paper if known," unpublished.

[3] Sedgewick, Robert. "Implementing quicksort programs." Communications of the ACM 21.10 (1978): 847-857.Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[4] Al-Kharabsheh, Khalid Suleiman, et al. "Review on sorting algorithms a comparative study." International Journal of Computer Science and Security (IJCSS) 7.3 (2013): 120-126.K. Eves and J. Valasek, " Adaptive control for singularly perturbed systems examples," Code Ocean, Aug. 2023. [Online]. Available: https://codeocean.com/capsule/4989235/tree

[5] https://github.com/gustavo-depaula/stalin-sort

[6] MercifulStalinSort/README.md at main · r-kataria/MercifulStalinSortS. Liu, "Wi-Fi Energy Detection Testbed (12MTC)," 2023, gitHub repository. [Online]. Available: https://github.com/liustone99/Wi-Fi-Energy-Detection-Testbed-12MTC

[7] Stalin Sort Visually Explained #sortingtechniques

[8] "code golf - Recursive Stalin Sort - Code Golf Stack Exchange

[9] https://github.com/Adarshsingh36/Merciful-Stalin-Sort-Paper