The Cheroz Algorithm: A Hybrid Approach Combining Insertion Sort and Heap Sort

Journal:	*Transactions on Computers*
Manuscript ID	Draft
Manuscript Type:	Regular
Keywords:	I.1.2.e Performance evaluation of algorithms and systems < I.1.2 Algorithms < I.1 Symbolic and algebraic manipulation < I Comput, F.2.2.g Sorting and searching < F.2.2 Nonnumerical Algorithms and Problems < F.2 Analysis of Algorithms and Problem Complexity <
Topic-select at least 1 and up to 3 areas from TC's scope. Click here for more info.:	f) performance, fault tolerance, reliability, security, and testability, e) specification, design, prototyping, and testing methods and tools, g) case studies and experimental and theoretical evaluations

The Cheroz Algorithm: A Hybrid Approach Combining Insertion Sort and Heap Sort

Cherry Lee H. Jimenez, Imroz Mae S. Khan, Junar M. Landicho
Department of Computer Science, University of Science and Technology of Southern Philippines, Cagayan de Oro City, Philippines.

***Abstract*— This journal offers a merging of the heap sort and insertion sort in the proposed Cheroz Algorithm for sorting. This journal aims at testing the practicability of using such an approach in handling data. Apart from conducting numerous experiments, we also carried out an extensive study on the algorithm's efficiency and effectiveness regarding heap sort as well as insertion sort alone. Our findings indicated that instead of being more effective than either method alone, this combination slightly fared better because some things in it were just unlike each other.**

***Index Terms*— Computer Science, Cheroz Algorithm, Enhanced Insertion Sort, Insertion Sort, Heap Sort, Hybrid Algorithm**

I. INTRODUCTION

In computer science, sorting is an essential operation with wide-ranging applications in diverse fields that include databases, computational problems and data processing. Despite the various algorithms continually made by researchers over the years, they still rely on comparison-based algorithms such as Bubble Sort, Insertion Sort, Selection Sort, and Heap Sort. Sorting is an activity involving arranging data in some order that is supposed to be essential in numerous applications including databases, computation problems, and data processing [8].

The Cheroz algorithm combines the strengths of two sorting methods: Heap Sort and Insertion Sort. Although it does not perform well on larger or almost sorted arrays with time complexity of O(n log n), but quite efficient for smaller ones with time complexity of O(n). In addition, the method has a modified version of Insertion sort which allows almost ordered arrays to be sorted even faster. Furthermore, this technique defines an automatic threshold for switching from one sort to another.

We are carrying out experiments to evaluate efficiency as well as study the nature and adaptability of the Cheroz Algorithm, Heap Sort, and Insertion Sort in different cases like random arrays, sorted arrays and reverse sorted ones.

II. REVIEW OF RELATED LITERATURE

This section from other journals describes the key findings of Heap Sort, Insertion Sort, and Hybrid Sorting Techniques.

A. Heap Sort

Heap sort improves the selection sort because it performs consistently well regardless of the input distribution. In their paper [1], Li et al. introduced a variation of heap sort designed explicitly for sorting arrays. This variation is more space-efficient and particularly effective when dealing with already-ordered sequences. It works by building an empty heap and sequentially inserting ordered element into it thus revealing optimisation possibilities within Heap Sort algorithm itself.

B. Insertion Sort

The worst-case time complexity in case of Insertion Sort does not matter as it is efficient for small datasets and almost sorted arrays just because it is trivially parallelisable on pipelined or vector-based computers. In [2] Soomro et al found out that however insertion sort performs worse than heap sort on smaller datasets hence one should choose the right algorithm based on their input conditions. On its part, binary insertion sort – another version – applies binary search to establish where exactly to insert a new element into a list thereby making fewer comparisons especially in relatively larger lists while still using the same old algorithms [4].

C. Hybrid Sorting Algorithms

Sabah et al. [3] compared six commonly accepted sorting algorithms across different data sets with varying types and sizes. Their research showed that a few algorithms perform better in unique situations while others are less costly in terms of memory or allow larger amount of data to be processed. This is how hybrid strategies reflecting the peculiarities of the given data could be advantageous. This includes Tim-Sort, which combines Merge Sort and Insertion Sort by merging pre-sorted input sections [3].

D. Implementation

Durrani and Abdulhayan [7] studied standard sorting algorithms performance revealing significant impact from choice of a programming language on it. They found out that when creating hybrid algorithms, there is the need for optimisation specific to language.

III. Methodology

This section describes the characteristics of the algorithms, how they relate to one another and why certain methods were chosen. It also goes over the experimental design that established if these algorithms serve their purpose.

TABLE I

Algorithm HYBRIDSORT

```
0:  procedure HYBRIDSORT(arr)
0:      n ← length of arr
0:      threshold ← DETERMINE_THRESHOLD(n)
0:      HEAPSORT(arr, 0, n - 1)
0:      if n ≤ threshold then
0:          INSERTION_SORT(arr, 0, n - 1)
0:      else
0:          ENHANCED_INSERTION_SORT(arr, 0, n - 1)
0:      end if
0:  end procedure
```

A. Key Components

1) The threshold is set to a maximum of 500,00 n/20, where n represents the input size. This ensures we can efficiently leverage Insertion Sort to handle ordered subsequences in larger arrays. For smaller arrays, it defaults to Binary Insertion Sort.
2) **Heap Sort:** was implemented using the standard method of building a max heap and extracting the maximum element iteratively to achieve ascending order sorting. This approach ensured an O(n log n) time complexity irrespective of input data distribution.
3) **Insertion Sort:** A simple implementation that iterated through the array, comparing each element with the previous elements and inserting it into its correct position was utilised.
4) **Enhanced Insertion Sort:** This component identified sorted runs (consecutive sorted subsequences) in the array after the Heap Sort phase. It sorted each run using Binary Insertion Sort separately. This approach improved performance, especially when dealing with large, sorted subsequences.

B. Experimental Design

Three types of input data are considered to evaluate the Cheroz Algorithm's performance:

1) **Random arrays:** To assess average-case performance.
2) **Sorted arrays:** To test best-case scenarios and the algorithm's ability to recognise and exploit existing order.
3) **Reverse-sorted arrays:** To examine worst-case behavior.

We use arrays of various sizes, from small (100 elements) to large (1,000,000 elements). This allows us to thoroughly analyze the algorithm's scalability and the effectiveness of its adaptive threshold.

C. Comparative Analysis

In order to comprehensively evaluate Cheroz Algorithm, stand alone versions of Insertion Sort and Heap Sort should be implemented and tested. These algorithms should go through the same input distributions (random, sorted, and reverse-sorted) and array sizes as the Cheroz Algorithm so that execution times can be compared directly in similar conditions to find out which scenarios hybrid approach is better than isolated algorithms or whether there is a need for it to be improved.

To evaluate the performance of the Cheroz algorithm, insertion sort and heap sort in various input data sets with regards to time taken. The number of runs will be made till we get statistical significant values hence giving average reaction times. This process will give us more insight on how each algorithm behavior changes based on different inputs and sizes hence leading to their strengths and weaknesses.

IV. Results and Discussions

	INSERTION SORT			HEAP SORT			HYBRID SORT		
Arrays	**Sorted**	**Random**	**Reverse**	**Sorted**	**Random**	**Reverse**	**Sorted**	**Random**	**Reverse**
100	0.002	0.002	0.098	0.045	0.017	0.014	0.02	0.014	0.016
1000	0.009	0.011	3.202	0.286	0.264	0.19	0.411	0.225	0.251
10000	0.165	0.102	347.895	3.413	2.459	2.903	4.154	3.221	5.928
100000	0.763	0.598	34677.6	47.328	32.726	38.019	35.049	43.594	37.721
1000000	7.359	-	-	470.784	329.084	456.184	419.881	482.139	475.472

Cheroz performs similarly to Heap Sort when dealing with sorted arrays but is slower than Insertion Sort. When working randomly, especially with larger arrays, Cheroz is slower than Heap Sort but still faster than Insertion Sort. Moreover, Cheroz closely matches the performance of Heap Sort with reversed arrays and even outperforms Insertion Sort in this scenario.

Sorting algorithms like Heap Sort and Insertion Sort can be difficult because they require extra work to switch between methods. This extra work can slow the sorting process instead of making it faster. Also, using a threshold to decide when to switch methods is not always best, as it only works well for

some situations. This lack of flexibility can lead to not-so-great performance when dealing with data different from what the rule was made for. Making and keeping up these algorithms is complicated and needs much work to ensure they do what they should.

V. Limitations and Recommendations

A. Limitations

In our tests, we discovered that the Cheroz Algorithm was not faster in execution time compared to the standard heap and insertion sort algorithms. The differences in results were only tens to hundreds of milliseconds, which did not meet our expectations. Although the hybrid algorithm may have worked, we saw no performance improvement.

We searched for journals focusing on hybrid algorithms discussing the Heap and Insertion Sort. Despite our efforts, we could not find any relevant hybrid journals, which made it challenging to find comprehensive comparisons for our algorithm.

B. Recommendations

To improve our research, we must reduce the switching overhead between the two algorithms and address the unstable threshold value. While the Cheroz Algorithm's results were disappointing, they provided valuable insights into designing and implementing this hybrid sorting algorithm. Further research and optimization are needed to achieve better performance and implement other algorithms that may improve its effectiveness.

VI. Conclusions

We combined the heap and insertion sorting algorithms throughout this project to create a hybrid sorting approach. We used heap sort for larger data sets and insertion sort for smaller ones. Firstly, we applied heap sort to the entire array. Then, depending on the size of the remaining unsorted array, we either used an enhanced or standard insertion sort. This hybrid method is expected to outperform each sorting algorithm individually.

References

[1] Li, H., Chen, P. and Wang, Y. (2017) Heap Sorting Based on Array Sorting. Journal of Computer and Communications, 5, 57-62. doi: 10.4236/jcc.2017.512006.

[2] Soomro, Inayatullah & Ali, Humaira & Lashari, Haque Nawaz & Maitlo, Abdullah. (2021). Performance Analysis of Heap Sort and Insertion Sort Algorithm. International Journal of Emerging Trends in Engineering Research. 9. 10.30534/ijeter/2021/08952021.

[3] Sabah, Ahmed S. ; Abu-Naser, Samy S. ; Helles, Yasmeen Emad ; Abdallatif, Ruba Fikri ; Abu Samra, Faten Y. A. ; Abu Taha, Aya Helmi ; Massa, Nawal Maher & Hamouda, Ahmed A. (2023). Comparative Analysis of the Performance of Popular Sorting Algorithms on Datasets of Different Sizes and Characteristics. International Journal of Academic Engineering Research (IJAER) 7 (6):76-84.

[4] Ramireddy, Gangadhar Reddy & Pr, Anisha & Reddy C, Kishor & Reddy, M.. (2017). An empirical study and analysis on sorting algorithms. International Journal of Mechanical Engineering and Technology. 8. 488-498.

[5] Chauhan, Yash & Duggal, Anuj. (2020). Different Sorting Algorithms comparison based upon the Time Complexity. 7. 10.1729/Journal.24472.

[6] Bedi, Hermehar Pal Singh & Kaur, Amandeep. (2022). Comparative Study of Different Sorting Algorithms used in Data Structures. 6. 114-119. 10.2017/IJRCS/202203021.

[7] Durrani, Omar Khan & Abdulhayan, Sayed. (2022). Performance Measurement of Popular Sorting Algorithms Implemented using Java and Python. 1-6. 10.1109/ICECCME55909.2022.9988424. Measurement of Popular Sorting Algorithms Implemented using Java and Python. 1-6. 10.1109/ICECCME55909.2022.9988424.

[8] P. K. Chhatwani, “Insertion Sort with its Enhancement,” Int. J. Comput. Sci. Mob. Comput., vol. 3, no. 3, pp. 801–806, 2014.