ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

IMPLEMENTATION OF *Cup_Sort()* ALGORITHM WHICH CLASSIFIES ARRAY ELEMENTS BY FINDING THE LENGTH OF DIGIT AND SORTING THEM IN KNOWN ORDER. FURTHER ASSESSING AND ANALYSING THE TIME COMPLEXITY OF ALGORITHMIC FUNCTION *Cup_Sort()* - A CASE STUDY.

Cadet P Dharvin[1](6700), Cadet B M Nethren[2] (6647)
Sainik School Amaravathinagar, Taluka Udumalpet, Tiruppur Dist Tamilnadu

ABSTRACT
In computer science effectiveness of algorithm is exclusively depend on time factor for the execution of included statements within the block of code. Further the amount of memory it is being used for storing data also matters in calculating the space complexity of the program.

The Cup_Sort() algorithm is a unique approach that sorts elements of an array by classifying them based on their digit length before arranging them in increasing order. This classification helps in reducing unnecessary comparisons and makes the sorting process more structured.

This manuscript focuses on the implementation of the Cup_Sort() function and studies its performance in terms of time complexity and space complexity. It also presents a case study to evaluate the efficiency and effectiveness of this algorithm when compared to traditional sorting methods.

KEYWORDS: *Cup Sort (CS), Digit-Length Group (DLG), Array (A), Classified Order (CO), Sorted Order (SO), Time Complexity (TC), Space Complexity (SC).*

1. INTRODUCTION
The Sorting is the process of arranging items into a sequence according to a specific order, such as text from A to Z, numbers from smallest to largest or from largest to smallest, or dates from oldest to newest or newest to oldest. It helps in organizing and understanding data, making it easier to visualize, find, and make decisions. While sorting can refer to simple categorization or organizing physical objects, in the context of computing, it specifically involves using sorting algorithms to reorder elements in an array or list.

Some of the most popular sorting techniques are Radix Sort, Merge Sort, Quick Sort, Counting Sort, Bucket Sort, Insertion Sort, etc. Here we will see the comparison between Bucket, Radix sorting technique with and cup sorting techniques.

Bucket Sort groups numbers by numerical ranges, Radix Sort groups them by individual digit places (ones, tens, hundreds), but Cup Sort is different because it groups numbers by digit length and positive/negative sign, creating category-based clusters instead of value-based or digit-position-based grouping. Thus, Bucket focuses on value intervals, Radix focuses on digit positions, and Cup Sort() focuses on digit count and classification, making it a unique, simple, and structure-based sorting method.

2. RELATED WORK
Related work on sorting techniques by researchers focuses on categorizing algorithms by their method (e.g., comparison-based, non-comparison-based), evaluating their performance in terms of time and space complexity ***(e.g., $O(n^2)$,$O(n \log n)$),*** and exploring their efficiency for different data sizes and types. Key areas of research include understanding fundamental algorithms like Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, and Radix Sort, as well as developing novel, more efficient sorting algorithms to handle increasingly large datasets.

3. METHODOLOGY
The ***Cup_Sort (CS)*** algorithm is a digit-length based sorting algorithm that organizes elements of an array by first classifying them according to the number of digits and then arranging them in increasing order within each group. Unlike traditional algorithms such as Bucket_Sort and Radix_Sort, which rely on divide-and-conquer strategies, Cup_Sort focuses on grouping elements into "Cups" (Cup) based on their digit lengths. These groups act as intermediate classifications that simplify the sorting process.

The execution process of the algorithm is as follows: once the user provides the array of numbers, the Cup_Sort() function determines the digit length of each element and places them into corresponding digit-length groups. Each Cup (group) holds numbers of the same digit count. The elements within each Cup are then sorted individually in increasing order, and finally, the sorted Cups are concatenated to form the final sorted array. This methodology emphasizes clarity and structure, as each intermediate step—classification into Cups and individual sorting—is explicit and easy to follow.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

The algorithm can be implemented using a programming language such as Python, and its efficiency can be analysed in terms of time complexity and space complexity.

4. ALGORITHM: *Cup_Sort()*

Step 1: Start
Step 2: Input
Step 3: Initialize Cups
Step 4: Digit Length Calculation
Step 5: Distribute into Cups
Step 6: Sort within Each Cup
Step 7: Display Cup Contents
Step 8: Merge Cups
Step 9: Display Final Output

5. IMPLIMENTATION

```
from collections import defaultdict
def digit_length(n):
    return len(str(abs(n)))
def Cup_sort_with_Cup_view(arr):
    neg_Cup = defaultdict(list)
    pos_Cup = defaultdict(list)
    for num in arr:
        d = digit_length(num)
        if num < 0:
            neg_Cup[d].append(num)
        else:
            pos_Cup[d].append(num)
    for d in neg_Cup:
        neg_Cup[d] = sorted(neg_Cup[d])
    for d in pos_Cup:
        pos_Cup[d] = sorted(pos_Cup[d])

    print("\n Negative Cups:")
    for d in sorted(neg_Cup):
        print(f"   {d}-digit negatives: {neg_Cup[d]}")
    print("\n Positive Cups:")
    for d in sorted(pos_Cup):
        print(f"   {d}-digit positives: {pos_Cup[d]}")
    combined = []
    for d in sorted(neg_Cup):
        combined.extend(neg_Cup[d])
    for d in sorted(pos_Cup):
        combined.extend(pos_Cup[d])
    final_sorted = sorted(combined)
    return final_sorted
numbers = list(map(int, input("Enter numbers separated
by spaces: ").split()))
final_output = Cup_sort_with_Cup_view(numbers)
print("\n Final Sorted Output :", final_output)
```

6. EXPLANATION OF THECODE

How It Works?

The code implements a Cup Sort with Cup View algorithm, a digit-length-based sorting technique. It takes a list of numbers from the user and separates them into "Cups" according to their digit count and sign (negative or positive). Inside the ***Cup_sort_with_Cup_view()*** function, negative and positive numbers are grouped into separate dictionaries (Cups) based on their number of digits. Each Cup is then sorted individually in ascending order.

The program prints the contents of each Cup clearly, showing how numbers are distributed across different digit lengths for negatives and positives. Finally, the sorted numbers from all Cups are concatenated in order: negative Cups first (from smallest to largest digit length) followed by positive Cups (from smallest to largest digit length), and the fully sorted list is returned. This allows the user to both visualize the sorting process and obtain the final sorted output.

7. COMPLEXITY OF ALGORITHM

In computer science, analysis of algorithms is a very crucial part. It is important to find the most efficient algorithm for solving a problem. It is possible to have many algorithms to solve a problem, but the challenge here is to choose the most efficient one.[1]

There are multiple ways to design an algorithm, or considering which one to implement in an application. When thinking through this, it's crucial to consider the algorithm's **time complexity** and **space complexity.**[1]

8. SPACE COMPLEXITY

The space complexity of an algorithm is the amount of space (or memory) taken by the algorithm to run as a function of its input length, n. Space complexity includes both auxiliary space and space used by the input[3]

Auxiliary space is the temporary or extra space used by the algorithm while it is being executed. Space complexity of an algorithm is commonly expressed using **Big (O(n))** notation[1]

The Space complexity is ignored in this research paper, since the space complexity of particular problem is not considered so important.

9. TIME COMPLEXITY

The time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, n. The time complexity of an algorithm is commonly expressed using asymptotic notations[1]:

Big O - O(n)
Big Theta - Θ(n)
Big Omega - Ω(n)

It's valuable for a programmer to learn how to compare performances of different algorithms and choose the best time-space complexity to solve a particular problem in the most efficient way possible.[1]

Big O notation is used in Computer Science to portrait the performance or complexity of an algorithm.

Big O specifically defines the worst-case scenario of an algorithm, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm. here O stands for order of growth.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 3| March2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

Big Theta(Θ) is used to represent the average case scenario of an algorithm and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

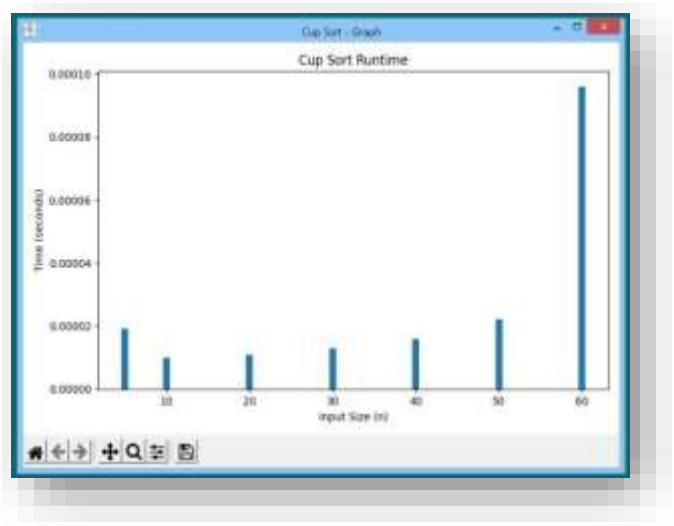
Big Omega (Ω)is used to represent the best case scenario of an algorithm and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

These three methods are the most common and very popular methods of design and analysis of an algorithm which are used for finding the efficiency of the program.

10. RUNTIME COMPLEXITY

Input(n)	*Time Complexity*
5	0.000019
10	0.000010
20	0.000011
30	0.000013
40	0.000016
50	0.000022
60	0.000096

11. RUNTIME COMPLEXITY GRAPH



12. CONCLUSION

The Cup Sort with Cup View algorithm is a new way of sorting numbers by grouping them into "**Cups**" based on how many digits they have and whether they are positive or negative. This makes it easy to see how the numbers are arranged and helps in sorting them efficiently. The runtime results show that the time taken increases slowly as the number of inputs grows, which matches its time complexity of ***O(n log n)*** in the worst case. The algorithm also uses memory efficiently with a space complexity of ***O(n).*** Overall, Cup Sort is a simple, clear, and effective method for sorting and viewing numbers in an organized way.

13.ACKNOWLEDGEMENT

Apart from the efforts of me, the success of any work or project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this research paper.

I express deep sense of gratitude to almighty God for giving me strength for the successful completion of the research paper.

I express my heartfelt gratitude to my parents for constant encouragement while carrying out this research paper.

I express my deep sense of gratitude to **The Principal Sainik School Amaravathinagar** who has been continuously motivating and extending their helping hand to us.

I express my sincere thanks to the academician The Vice-Principal and visionary Administrative Officer for constant encouragement and the guidance provided during this research.

My sincere thanks to **Mr. Praveen Kumar Murigeppa Jigajinni**, Master In-charge, A guide, Mentor and great motivator, who critically reviewed my paper and helped in solving each and every problem, occurred during implementation of this research paper.

12. REFERENCES

1. *https://www.educative.io/edpresso/time-complexity-vs-space-complexity*