# Linear Time Sorting in Practice

Md.Kazi Rakibul Hassan [1] Rayhan Amin Akash [2] Nipa Akter [3] Md.Saifur Rahman [4] Adison Barua [5]

Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh.
{[1]2220936,[2]2220972,[3]2211056,[4]2221902,[5]2222131,[6]groupMember6}@iub.edu.bd

## Abstract

This project introduces the implementation and analysis of Linear time sorting algorithms,focusing on their efficiency and applicability in sorting datasets. Sorting is an essential operation in computer science that involves arranging elements into a specific order. Linear time sorting is a subset of sorting algorithm with a significant advantage,they can sort a given sort of elements in linear time. There are some linear time sorting algorithms among those we have chosen Counting sort algorithm for this project where we have discussed why we chose this sorting algorithm to solve a real world problem,why it was selected for solving that problem,detailing their underlying principles,implementation strategies,the working process,performance characteristics,the uses of counting sort in various fields etc.

## Introduction

Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array. Counting sort is relatively simple,efficient,fast and reliable linear sorting algorithm,especially in scenarios where the range of values to be sorted is known and relatively small. Its linear time complexity O(n+k), where n is the number of elements and k is the range of values, makes it particularly suitable for scenarios where the input size is large but the range of values is limited. As counting sort is seemingly simple,finds practical applications in various real world scenarios. Among some real world scenarios we have chosen grade sorting in educational environments. Here we have discussed about solving the real world problem with the help of counting sort algorithm. Counting sort provides an efficient solution to the problem of grade sorting in educational environments. The working process of counting sort in grade sorting is given below:
1.Efficient sorting: Counting sort's linear time complexity allows to sorts grades quickly, even the number of students is large. Counting sort efficiently sorts grades by counting the occurrences of each grades within a predefined range(A to F or in numerical order from 0 to 100) and placing them directly into their sorted positions. Educational institutions often needs to sort grades of students. Whether it is generating report cards,analyzing class performance,organizing grades from A to F or marks from 0 to 100. Through counting sort we can efficiently organize these tasks.
2.Stability: Counting sort is a stable sorting algorithm, it means that students with the same grade retain their original order in the sorted output. This stability is crucial to maintain fairness in grading and ensuring the students to with equivalent performance are treated consistently.
3.Ease of implementation: Counting sort is relatively easy to implement,its straightforward implementation allows for easy integration into grade management tool and application,facilitating efficient grading process,making it accessible to educators and developers working on educational software systems.
**Algorithm paradigm:** The Counting Sort algorithm belongs to the category of non-comparison based sorting algorithms. The paradigm of counting sort algorithm resolves around the efficient counting of elements into their sorted positions based on counts. This approach offers simplicity,efficiency,and stability,making counting sort a valuable tool in various fields.

## Rationale for the Algorithm's Selection

The real-world scenario we have chosen for this project is grading sort in educational environment. Counting sort is commonly used in grading sort due to it's simplicity,efficiency,stability for scenarios with a known range of grades. It's linear time complexity makes it highly efficient for sorting large number of students,when the possible grades or marks is limited. Sort's stability ensures that students with the same grade maintain their original order, maintaining fairness in grading. Overall,Counting sort's simplicity,efficiency,ease of implementation to integrate into educational software systems make it an excellent choice for grading sort task in educational environment.

## Methodology

This section details the application of Counting Sort in solving the real-world problem of grade sorting in educational environments. We'll explain the algorithm's usage within the context of student data and its advantages for this specific task. Counting Sort for Grade Sorting:
1. Data Preparation: Student data is gathered, typically consisting of name-grade pairs represented as tuples (name, grade). The assumed range of grades (e.g., 0 to 100) is determined.
2. Counting Phase: A count array is initialized with a size equal to the maximum grade + 1 (to accommodate all possible grades). Each student's grade is used as an index to increment the corresponding element in the count array. This step effectively counts the occurrences of each unique grade within the student data.
3. Modification of Count Array: The count array is iterated from left to right. Each element in the count array is modified to represent the cumulative sum of occurrences up to that point. This calculation essentially determines the final sorted position for each grade based on its frequency.
4. Sorting Phase: A new array is created to store the sorted student data. The student data is processed in reverse order. For each student-grade pair:
The corresponding grade's count in the modified count array is retrieved. This value indicates the correct sorted position for the student. The student-grade pair is inserted into the new array at the determined position. The count for that grade is then decremented to avoid duplicate placements in the sorted array.

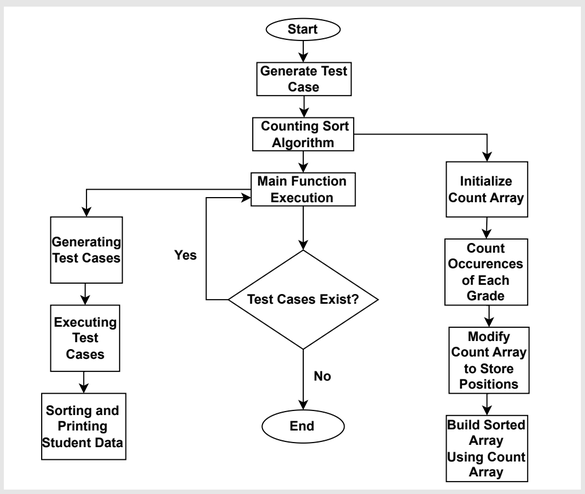


Figure 1. flowchart

## Substitute Use of the Algorithm

Yes, the Counting Sort algorithm used for sorting exam scores can indeed be applied to various other applications due to its efficiency and simplicity, especially in scenarios where the range of input values is known in advance. Here are a few examples of how Counting Sort can be utilized:
1. Inventory Management : Counting Sort can be applied in inventory management systems to sort items based on their stock keeping unit (SKU) numbers or product IDs. If the range of SKUs or product IDs is limited and known, Counting Sort can efficiently organize inventory records.
2. Library Cataloging : Libraries often need to sort and organize books based on their Dewey Decimal Classification (DDC) numbers or Library of Congress (LC) call numbers. Counting Sort can be utilized here to quickly arrange books on shelves in their proper order.
3. Customer Data Analysis : In retail or e-commerce, Counting Sort can be used to analyze customer data, such as sorting customers based on their purchase history, demographic information, or loyalty program status. This can help in targeted marketing campaigns or personalized recommendations.
4. Event Planning : Event planners may need to sort and arrange attendees based on various criteria such as ticket types, seating preferences, or dietary restrictions. Counting Sort can assist in efficiently organizing attendee lists to streamline event management processes.
5. File System Organization : Operating systems often need to organize files and directories based on attributes such as file size, creation date, or file type. Counting Sort can be applied to efficiently sort and arrange files in directory listings or search results.
In each of these applications, Counting Sort offers a fast and efficient solution for organizing data when the range of input values is relatively small and known beforehand. By leveraging its simplicity and linear time complexity, Counting Sort can significantly improve the efficiency of various sorting and organizing tasks in different domains.

## Alternative Solution

While Counting Sort is efficient for sorting integers with a known range, there are scenarios where other sorting algorithms may provide better overall performance, depending on the characteristics of the input data and specific requirements of the application. Here are a few alternatives and potential improvements:
1. Radix Sort : Radix Sort is another linear-time sorting algorithm that is particularly efficient for sorting integers with a fixed number of digits or bits. It can outperform Counting Sort when the range of values is very large, as it doesn't rely on storing counts for each value.
2. Bucket Sort : Bucket Sort is effective when the input data is uniformly distributed over a range. It divides the range into a fixed number of buckets, then sorts elements within each bucket using another sorting algorithm (such as insertion sort or quick sort). This can be advantageous if the input data is not evenly distributed.
3. Merge Sort or Quick Sort : While not linear-time algorithms, Merge Sort and Quick Sort can still offer excellent performance in many scenarios, especially when the input size is large or when the range of values is not known in advance. They have better average-case time complexity compared to Counting Sort.
4. Hybrid Approaches : Depending on the characteristics of the input data, a hybrid approach that combines different sorting algorithms could provide better performance. For example, using a combination of Counting Sort and Quick Sort can exploit the strengths of both algorithms, especially when dealing with large datasets with a relatively small range of values.
5. Parallelization : Implementing parallel versions of sorting algorithms can significantly improve performance, especially on multi-core processors or distributed systems. Techniques like parallel Counting Sort or parallel Radix Sort can exploit parallelism to sort data more efficiently.
6. Memory Optimization : Depending on the available memory and constraints of the system, optimizing memory usage can enhance performance. For example, using compressed data structures or adapting the counting array size dynamically based on the input data distribution can reduce memory overhead.
7. Error Handling : Incorporating error handling mechanisms to handle edge cases, such as out-of-range values or unexpected input formats, can improve the robustness and reliability of the system.
When considering improvements to the existing system, it's essential to analyze the specific requirements, constraints, and characteristics of the application and input data. Experimentation, benchmarking, and profiling can help identify bottlenecks and guide the selection of the most appropriate sorting algorithm or optimization techniques.

## Conclusion

In conclusion, employing counting sort for sorting grades in educational environments has proven highly effective and efficient. Its simplicity, stability, and linear time complexity make it invaluable for handling large datasets of grades or marks. By integrating counting sort, educational institutions can streamline grading processes, generate accurate reports, and ensure fairness in student assessment.