Comparison of Merge Sort and Counting Sort

C.S.E.-2200

Slide Creation:
Md. Shahidul Islam Dipu (2003040)
Zabed Iqbal (2003011)

Article Writting:
MD.Sartaj Alam Pritom (2003046)
Nafis Raihan (2003005)

Latex Creation
Raihan Ul Islam (2003006)
Soscho Shamuel Gregory (2003007)

Outline

Abstract
Introduction
Background Study
Merge Sort
Counting Sort
Results and Analysis
Introduction to Results
Execution Times
Graph
Analysis
Conclusion
Reference

Abstract

- **Objective:** Conduct a comparative analysis of Merge Sort and Counting Sort.
- **Focus:** Evaluate performance across diverse dataset sizes.
- **Complexities:** Explore time (Merge Sort: $O(n \log n)$, Counting Sort: $O(n + k)$) and space complexities.
- **Characteristics:** Merge Sort's versatility suits various datasets; Counting Sort excels in linear efficiency, ideal for large datasets with limited value ranges.
- **Methodology:** Systematic experimentation with varying dataset sizes.

Abstract

- **Findings:** Empirical evidence on sorting algorithms' performance.
- **Implications:** Practical insights for real-world algorithm selection.
- **Limitations:** Acknowledge dataset-related constraints.
- **Recommendations:** Optimize Merge Sort; explore additional sorting algorithms.
- **Conclusion:** Concise analysis aids sorting algorithm discussions for researchers and practitioners.

Introduction

- Sorting algorithms significantly impact computer science applications.
- Efficient data organization and retrieval are crucial in modern computing.
- Sorting algorithms play a pivotal role in diverse applications.

Introducing Merge Sort and Counting Sort

Md. Shahidul Islam Dipu, Zabed Iqbal

- **Merge Sort:** Known for elegance and reliability. Efficiently breaks down and merges datasets, ensuring intricately ordered data.
- **Counting Sort:** A non-comparative algorithm with $O(n + k)$ complexity. Particularly effective in sorting integers and character frequencies within limited value range datasets.

Introducing Merge Sort and Counting Sort (cont.)

- **Applications:**
  Merge Sort: Ideal for stable sorting in data processing and scientific computing.
  Counting Sort: Efficient for sorting integers and character frequencies.
- Both algorithms contribute to real-world scenarios.

Objectives of this Study

- This research aims to compare Merge Sort and Counting Sort comprehensively.
- Evaluate theoretical foundations and practical performance on diverse datasets.
- Provide nuanced insights into trade-offs and advantages of each algorithm.

Background Study: Merge Sort Overview

- **Divide and Conquer:** Merge Sort employs a "divide and conquer" strategy, breaking down the sorting process into smaller, more manageable tasks.
- **Recursive Sorting:** It recursively divides the array into halves until individual elements, then merges them in a sorted manner.
- **Time Complexity:** Exhibits a time complexity for both Average and Worst case of $O(n \log n)$, making it efficient for large datasets.
- **Space Complexity:** Requires $O(n)$ auxiliary space for the merging operation.

Background Study: Merge Sort Algorithm (Merge)


```
merge(left, right):
result = []
left_index = right_index = 0

while left_index < length(left)
and right_index < length(right):
if left[left_index] < right[right_index]:
result.append(left[left_index])
left_index++
else:
result.append(right[right_index])
right_index++

result.extend(left[left_index:])
result.extend(right[right_index:])
```

Background Study: Merge Sort Algorithm (MergeSort)


```
mergeSort(arr):
if length(arr) <= 1:
return arr

mid = length(arr) / 2
left_half = mergeSort(arr[0:mid])
right_half = mergeSort(arr[mid:])
return merge(left_half, right_half)
```

Merge Sort recursively dissects datasets, sorting smaller components, and seamlessly merges them back into wholes. This divide-and-conquer strategy ensures ordered data and optimization for efficient retrieval.

Background Study: Merge Sort Visualization

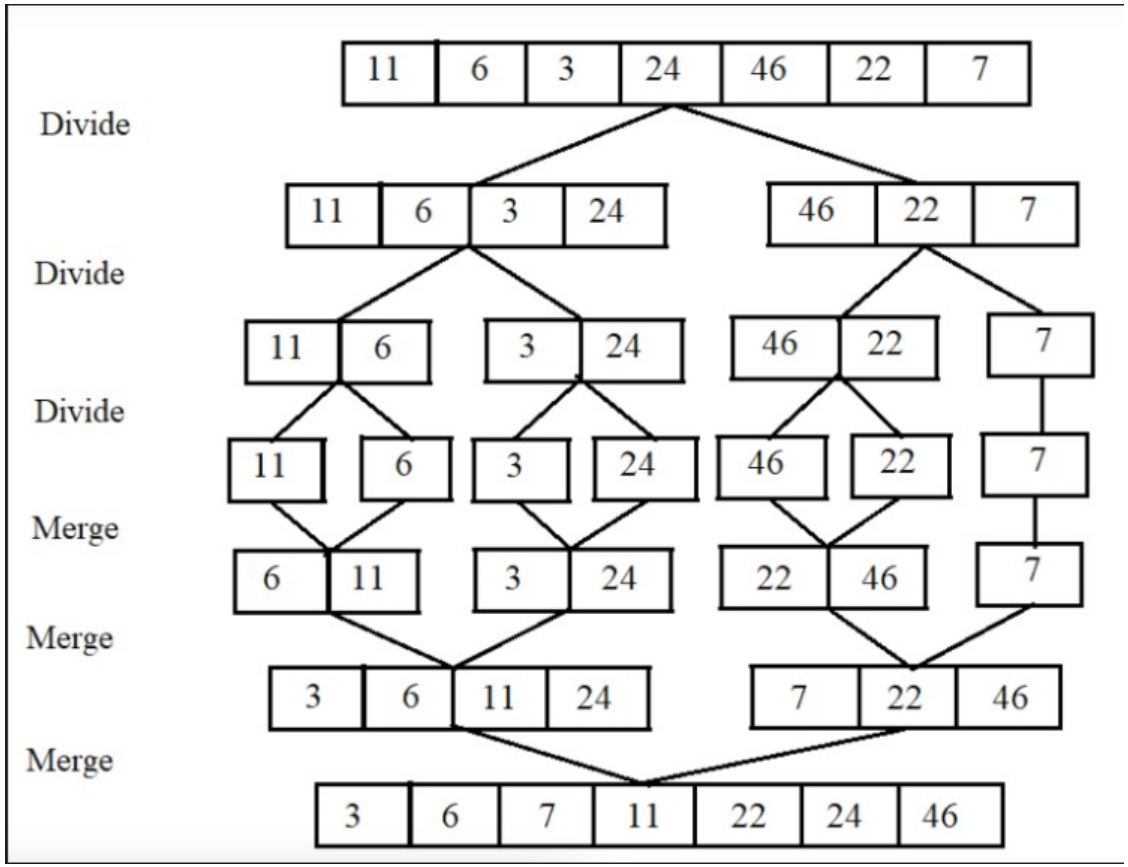

Figure: Process of MergeSort

Background Study: Counting Sort Overview

- **Counting Sort Overview:** Counts the occurrences of each element in the input. Calculates the cumulative count of elements. Places elements in their correct, sorted positions.
- **Time Complexity:** Demonstrates a time complexity of $O(n + k)$, where k is the range of input. Exhibits stability, maintaining the order of equal elements.
- **Space Complexity:** Requires $O(k)$ additional space for the counting array.
- **Visualization:** Visual representation aids in understanding the counting and placement process.

Background Study: Counting Sort Algorithm


```
Count array C[0..k] initialized to 0;
for i from 1 to n do
C[A[i]] += 1;
for i from 1 to k do
C[i] += C[i-1];
for i from n to 1 do
B[C[A[i]]] = A[i];
C[A[i]] -= 1;
```

Counting Sort begins by creating an array filled with zeros and counting occurrences of each distinct element. It then calculates positions for elements and efficiently places them in their final order.

Result and Analysis

- **Introduction to Results:**
  - This experiment meticulously evaluates and compares Merge Sort and Counting Sort, two fundamental sorting algorithms.
  - Sorting algorithms are crucial in computer science and data processing, impacting the efficiency and speed of applications.
  - Our study aims to analyze algorithm performance across a spectrum of dataset sizes, from small to significantly large datasets.
- **Dataset Variation and Configuration:**
  - Datasets of varying sizes (10,000 to 100,000 data points) were configured for a comprehensive algorithm performance assessment.
  - The selected datasets represent practical scenarios, covering everyday modest-sized datasets to extensive datasets reflecting substantial data processing demands.

Execution Time

Data Size	Merge Sort Time(ms)	Counting Sort Time(ms)
10000	4074.8	534.5
20000	8387.2	691.9
30000	13791.6	979.9
40000	17537.1	1141.7
50000	22100.3	1336.2
60000	26453.6	1571
70000	31218.4	1709.2
80000	35476	1955.8
90000	40404.9	2691.7
100000	45341.8	3288.1

Table: Execution times for Merge Sort and Counting Sort

Graph

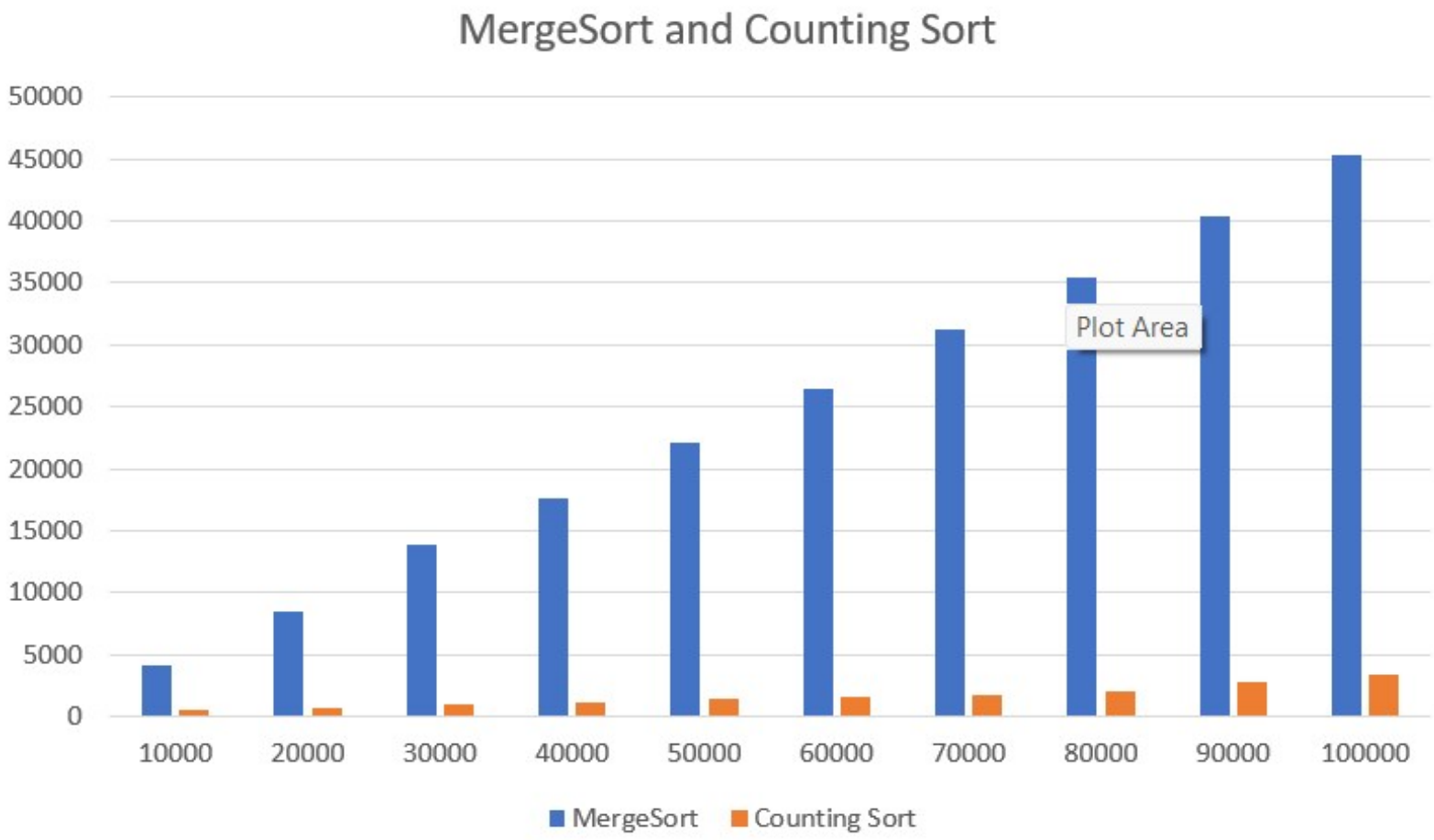

Figure: Performance analysis comparison

Observations

- **Merge Sort:**
  - Time increases consistently with the growth of the dataset.
  - Reflects the expected behavior of $O(n \log n)$ time complexity.
  - Non-linear increase indicates the influence of the logarithmic factor.
- **Counting Sort:**
  - Demonstrates consistently lower times compared to Merge Sort.
  - Linear time complexity ($O(n + k)$) contributes to its efficiency, especially for smaller datasets.

Explanation

- **Merge Sort:**
  - Performs well but exhibits a logarithmic growth in time.
  - Becomes relatively more efficient as dataset size increases.
- **Counting Sort:**
  - Maintains efficiency across all dataset sizes.
  - Linear time complexity allows for faster sorting, especially for smaller datasets.

Recommendations

- **For Smaller Datasets:**
  - *Counting Sort:* Highly efficient due to its linear time complexity. Recommended for datasets with limited sizes.
- **For Larger Datasets:**
  - *Merge Sort:* Becomes a competitive and efficient choice as dataset size increases. Balances efficiency and stability for larger and more complex datasets.

Practical Implications and Recommendations

- **Merge Sort:**
  - Suitable for general-purpose sorting where stability and adaptability are crucial.
  - Versatile, making it viable for diverse sorting requirements in manageable dataset sizes.
  - Particularly beneficial when a stable sorting algorithm is required.
- **Counting Sort:**
  - Ideal for scenarios with a limited range of values.
  - Efficient and scalable, particularly advantageous for large datasets.
  - Well-suited for situations with a constrained range, prioritizing sorting speed over versatility.

Conclusion

- The comprehensive comparison between Merge Sort and Counting Sort has illuminated their distinct characteristics.
- Merge Sort, though stable, demonstrated limitations in efficiency for larger or complex datasets, aligning with its expected time complexity growth of $O(n \log n)$.
- Counting Sort's linear time complexity ($O(n + k)$) showcased remarkable efficiency, particularly excelling in scenarios with large datasets.
- The findings endorse Counting Sort as a preferred choice for fast sorting in applications prioritizing sorting speed.
- Its adaptability to diverse real-world scenarios offers valuable insights for algorithm selection.
- The study underscores the significance of considering algorithmic characteristics and real-world requirements when choosing sorting methods, ultimately guiding towards informed and effective decision-making.

Reference

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. (2009). "Introduction to Algorithms (3rd ed.)." MIT press.
- D. Garg and S. N. Maheshwari. (2017). "A Practical Introduction to Data Structures and Algorithms Using Python." CRC Press.
- GeeksforGeeks - Merge Sort (`https://www.geeksforgeeks.org/merge-sort/`)
- GeeksforGeeks - Counting Sort (`https://www.geeksforgeeks.org/counting-sort/`)
- Wikipedia
- ResearchGate
- Google Scholar (`https://scholar.google.com`)

THANK YOU