Comparison of Radix Sort and Tim Sort

An Abstract

Tonmoy

CSE, RUET

January 23, 2024

Presentation Outline

Abstract

[8]This paper presents a comparative analysis of Radix Sort and Tim Sort, two popular sorting algorithms. Radix Sort, a linear time, non-comparative algorithm, is explored with an enhanced version incorporating Counting Sort. Tim Sort, a hybrid algorithm combining Merge Sort and Insertion Sort, is detailed with a focus on its adaptability. The study aims to provide insights into their strengths, weaknesses, and optimal use cases.

Introduction to Sorting Algorithms

Radix Sort Overview

Radix Sort is a linear time, non-comparative sorting algorithm that operates on integers by processing individual digits. It exhibits excellent performance for fixed-size integer keys and is known for its simplicity and efficiency in certain scenarios.[7]
The time complexity of Radix Sort is linear, $O(kn)$, where k is the number of digits in the input integers.

Introduction to Sorting Algorithms

Tim Sort Overview

Tim Sort, a hybrid sorting algorithm derived from Merge Sort and Insertion Sort, is designed for real-world data. Its ability to adapt to various scenarios, along with its efficiency in handling partially ordered data, makes it a popular choice in practice.
The average-case time complexity of Tim Sort is $O(n \log n)$, where n is the size of the input array.[3]

Background Study

Radix Sort Pseudocode with Counting Sort[3]

Algorithm 1: Radix Sort with Counting Sort

Data: Array A, Number of digits d

```
for i ← 1 to d do
    Use Counting Sort to sort array A on digit i;
end
```

Background Study

Counting Sort Pseudocode

Algorithm 2: Counting Sort

Data: Array A, Range of elements k

Result: Sorted array A

```
C ← array of size k initialized with zeros;
for i ← 1 to n do
    C[A[i]] ← C[A[i]] + 1;
end
for i ← 1 to k do
    C[i] ← C[i] + C[i − 1];
end
for i ← n to 1 do
    B[C[A[i]]] ← A[i];
    C[A[i]] ← C[A[i]] − 1;
end
```

Background Study

Tim Sort Pseudocode with Insertion Sort and Merge Sort[5]

Algorithm 3: Tim Sort with Insertion Sort and Merge Sort

Data: Array A

Result: Sorted array A

```
Divide the array into small blocks using Insertion Sort;
while blocks to merge do
    Merge adjacent blocks using Merge Sort;
end
```

Background Study

Insertion Sort Pseudocode

Algorithm 4: Insertion Sort

```
Data: Array A
Result: Sorted array A
for i ← 2 to n do
    key ← A[i];
    j ← i − 1;
    while j > 0 and A[j] > key do
        Swap A[j + 1] and A[j];
        j ← j − 1;
    end
    A[j + 1] ← key;
end
```

Background Study

Merge Sort Pseudocode (Part 1)

Algorithm 5: Merge Sort

Data: Array A, Indices p, q, r

Result: Sorted array A

```
n1 ← q − p + 1, n2 ← r − q;
Create arrays L[1..n1 + 1] and R[1..n2 + 1];
for i ← 1 to n1 do
    L[i] ← A[p + i − 1];
end
for j ← 1 to n2 do
    R[j] ← A[q + j];
end
L[n1 + 1] ← ∞, j ← 1;
R[n2 + 1] ← ∞, i ← 1;
```

Background Study

Merge Sort Pseudocode (Part 2)

Algorithm 5: Merge Sort (continued)

Data: Array A, Indices p, q, r
Result: Sorted array A (continued)

```
for k ← p to r do
    if L[i] ≤ R[j] then
        A[k] ← L[i];
        i ← i + 1;
    else
        A[k] ← R[j];
        j ← j + 1;
```

Implementation of Radix Sort[2]

- **Input:** Array A
- **Output:** Sorted array A using Radix Sort
- Find the maximum number of digits, d, in elements of A
- **For** i from 1 to d:
  - Use Counting Sort to sort array A based on the i-th digit

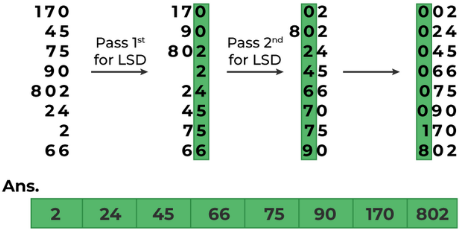


Figure: Radix Sort Visualization

Implementation of Tim Sort

- **Input:** Array A
- **Output:** Sorted array A using Tim Sort
- Divide the array into small blocks using Insertion Sort
- **While** there are blocks to merge:
  - Merge adjacent blocks using Merge Sort

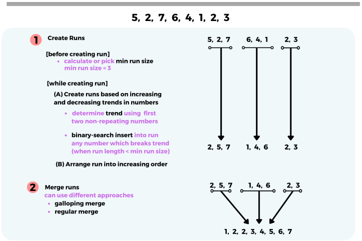


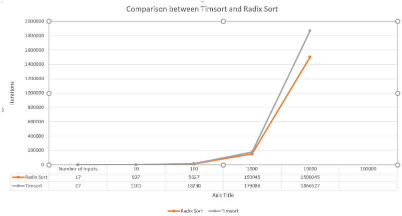
Figure: Tim Sort Visualization

Comparison Table

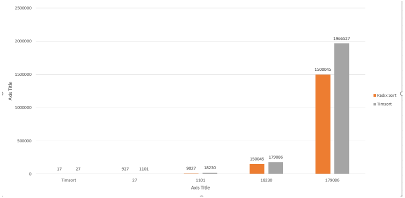
Table: Comparison between Timsort and Radix Sort

Topics	**Radix Sort**	**Timsort**
Time Complexity	$O(nk)$	$O(n \log n)$
Space Complexity	$O(n + k)$	$O(n)$
Application	Sorting items of fixed-length. Sorting strings where product of the length of the largest item is and number of item is not too large.	[6]Timsort is a hybrid sorting algorithm derived from merge sort and insertion sort. Where is a balance between time and space complexity is desired.

Time complexity



(a) Chart Comparison



(b) Bar Comparison

Figure: Comparison between Timsort and Radix Sort.

Time complexity

Table: Comparison between Radix Sort's and Timsort's iterations

Number of Inputs	Radix Sort	Timsort
10	17	27
100	927	1101
1000	9027	18230
10000	150045	179086
100000	1500045	1666527

1. Radix Sort's time complexity is almost similar to $O(n * k)$, where n stands for the number of items in the data set, and k is the length of the longest number in the data set.
2. Timsort's time complexity is almost similar to $O(nlogn)$, where n stands for the number of items in the data set[4].

Conclusion

- In conclusion, Radix Sort and Timsort are both effective sorting algorithms, each with its own strengths and use cases.
  Radix Sort:
  - Radix Sort demonstrates efficiency in scenarios where the range of input values is not significantly larger than the number of elements.
  - With a time complexity of $O(n \cdot k)$, where n is the number of elements and k is the length of the longest digit.[1]

  Timsort:
  - Timsort, a hybrid sorting algorithm derived from merge sort and insertion sort, exhibits a time complexity of $O(n \log n)$.

References I

[1] Khalid Suleiman et al. Al-Kharabsheh. Review on sorting algorithms: A comparative study. *International Journal of Computer Science and Security (IJCSS)*, 2013.

[2] Arne Anderson and S. N. Implementing radix sort. *Experimental Algorithmic (JEA)*, 1998.

[3] J. Hammad. A comparative study between various sorting algorithms. 15(March):6, 2015. March.

[4] Md. Khairullah. Enhancing worst sorting algorithms. *International Journal of Advanced Science and Technology*, 2013.

[5] S. K. P. Megha Jain. Bitonic sorting algorithm: A review. Volume 113 - No. 13:2, 2015. March.

[6] S. B. Sahni. Gpu radix sort for. *IEEE*, 2011.

References II

[7] Avinash Shukla and A. K. K. S. Review of radix sort & proposed modified radix sort for heterogeneous data set in distributed computing environment. *Journal Name*, 2(5):6, 2012.

[8] A. K. Verma. A new approach for sorting list to reduce execution time. page 7, 2013. 13 October.