Comparative Study of Traditional Sorting Techniques and Latest Sorting Techniques

Riyazahemed Abdulagani Jamadar
AIISSMS'S Institute of Information Technology, Shivaji Nagar, Pune-411001
Maharashtra, India

Received: 10 May Revised: 18 May Accepted: 26 May

Abstract
Sorting algorithms are one of the most fundamental research areas in the field of computer science and engineering. The need of this research is to make the data or information access faster and efficient in terms of memory consumption. The efficiency of real implementations is usually, at least as important as the theoretical performance of the abstract algorithm. For instance, Quick sort outperforms most practical situations, regardless of the fact that many other sorting algorithms have a better worst-case behavior. In this paper I have presented the study of different sorting algorithms and discussed and compared the differences in both theory and practice. Furthermore, the three different types of performance behaviors investigated such as O(n²) class and O(nlog₂n) class and non-comparison or linear class. Each sorting class outperforms the other class under certain conditions such as non-comparison based sorting algorithms outperforms comparison based sorting algorithm in large data-sets and O(nlog₂n) class algorithm outperforms the O(n²) class algorithms in large data sets.

Keywords – efficiency, performance, sorting, time complexity.

Introduction
Sorting is a paramount operation in computing technologies. The primary objective of sorting is to make the searching, insertion and deletion operations fast. Basically there are two class of sort algorithms based on the data whether stored in the main memory [1] or secondary memory. One class is the internal sort which stores the data in the primary memory (RAM). The other class is the external sort which stores the data on the disk, due to larger data sets. Here primarily the internal sorts have been studied and analyzed, moreover contiguous data structures have been taken into consideration, so that, the order of data items is determined by the position which stored in the memory [1].

The major parameter to measure the performance of a sorting algorithm is compute running time and storage needed for an algorithm. In the context of Designing and analyzing an algorithm, the time complexity refers to the amount of computing time it needs to run to complete the specified task. In this discussion the performance of sorting algorithms has been compared with respect to time complexity. Normally the asymptotic notation used to specify the upper bound of time is big Oh(O) notation, and this study uses this, to establish comparison among different algorithms.

The selection of a particular sorting algorithm is done based on different parameters like, input size, the data type of an item and presence of duplicate values [2]. In practice hardly a single
I. Traditional sorting techniques and their performance analysis

   i. Bubble Sort

   Bubble sort technique is one of the inefficient algorithm as it has a worst-case and average-case complexity of $O(n^2)$, where $n$ is the number of data-items to be sorted. Similar performance exhibiting other simple sorting algorithms such as insertion sort and selection sort have the same worst-case complexity of $O(n^2)$, however the efficiency of bubble sort is relatively lesser than these algorithms. Thus it could be said that the bubble sort should not be considered for a larger data sets.

   Besides this traditional method, there is a modified way of using the bubble sort, it adds a flag variable, which is set if an exchange is done after an entire pass, if there the remains unchanged, means no exchange is made then it clearly indicates that the array is already in order. Thus it gives the best-case complexity of $O(n)$ if the array is already sorted. Another variant of bubble sort is Bidirectional Bubble Sort or Cocktail Sort [3] which sorts the data-set in both directions, this process slightly reduces the number of comparisons. The performance of Butcher’s [4] proposed a method has better running time than both unidirectional bubble sort and bidirectional Bubble sort. From this discussion it is apparent that, Bubble sort is the slowest for large data-sets, with the only advantage of having easy implementation.

   ii. Insertion Sort

   Insertion sort is another basic sorting algorithm, which uses $n - 1$ comparisons for $n$ data-items with the computational complexity $O(n^2)$. If the data-set is already sorted, the computational complexity predicted to be linear i.e $O(n)$. This will be the best-case running time of insertion sort. The average-case running time is also $O(n^2)$ which demonstrates that the insertion sort is also in-efficient for larger data-sets with exception that it performs better than bubble sort. For the reason that it is simple and very efficient for smaller data-sets, insertion sort is used in some sophisticated algorithm such as quick sort, merge sort for greater efficiency.

   iii. Selection Sort

   Since selection sort is an iterative algorithm like other simple sorting methods, it uses $n - 1$ comparisons for $n$ data-items with the computational complexity $O(n^2)$. In this technique, selecting the highest data-item needed to compare all n items in the data-set at first iteration and swapping them if required. Likewise to select the next highest data-item in the data-set and so on. Hence it...
requires $O(n^3)$ comparisons and $n-1$ swaps to sort the list of $n$ elements. Since it has the worst-case running time of $O(n^2)$ it is also not efficient for large data-sets. However, when compared to another quadratic complexity algorithm such as bubble sort, selection sort has much better in efficiency.

There are two different of selection sort that are quite popular such as Quadratic Sort and Tree Sort [5]. In Quadratic sort for example, a data-set 1 of 16 data-items are grouped in four sub-groups of four data-items each and then find the largest data-item in each sub-group and storing them in a sub data-set 2. Later the largest item in the sub data-set is found by sequential search and swapped with the last position in the data-set 1. This process will be continued till the data-set gets sorted. Whereas Tree sort uses the concept of the knockout tournament. Like other iterative sorting algorithms selection sort is not efficient for large data-sets and has better efficiency than bubble sort. Because of its simplicity and ease to implement selection sort would recommend for certain conditions. However, it is recommended to use insertion sort instead of selection sort for data-sets with the smaller input size.

iv. Merge Sort

Merge sort is the most popular and efficient sorting method when compared to simple sorting methods and also some sophisticated methods such as heap sort and quick sort. The worst-case and average-case running time complexity of merge sort is $O(n\log_2n)$. As Merge sort use a separate space(data-set) to store the entire sub data-set along with the main data-set(list or an array).

Normally speaking, it is an external sorting algorithm that has additional memory space $O(n)$ for $n$ data-items. This requirement categorizes merge sort as inefficient approach for the application that run on machines with limited memory resource. Thus merge sort is recommended normally for large data-sets with external memory.

v. Quick Sort

Quick sort is another most commonly employed fastest sorting algorithm with average running time complexity of $O(n\log_2n)$ when compared to other classy algorithms. Normally when the leftmost or rightmost data-item is selected as a pivot, it introduces the worst case running time of $O(n^2)$. It starts behaving inefficient if all the input data-items are similar, and the algorithm will take quadratic time $O(n^2)$ to sort. Seldom are these worst-case scenarios encountered.

Qsort proposed by [6] is one of the variants of quick sort, which has better running time and more robust than the traditional method. If speed is the major concern then Quick sort is the better option for large data sets. One of the major advantages of quick sort is that it is fast and efficient for large data sets. However it is not efficient if the data-items are already sorted and for similar data-items. This gives the worst case-time complexity of $O(n^2)$. Looking with respect to space, Quick sort might be space expensive for large data sets due to auxiliary space for recursive-function calls.
vi. Heap Sort

Heap Sort uses the heap data structure (Tree) and is quite slower than merge sort in real application even though it has the same theoretical complexity. Unlike merge sort and quick sort it does not work recursively. In general, heap is a specialized tree based data structure that satisfies the heap property. The tree structure is well balanced, space efficient and fast.

Heap sort works by building a heap from the input data-set and then removing the maximum item from the heap and placing it at the end of the final sorted data-set, i.e. 

\[
(n - 1)^{\text{th}}
\]

position. Every time when it removes the maximum item from the heap it restores the heap property until the heap is empty. Thus it removes the second largest element from a heap and puts it on the 

\[
(n - 2)^{\text{th}}
\]

position and so on. The algorithm repeats this operation until the data-set gets sorted. The worst case and average case running time complexity of heap sort is \(O(n \log n)\). When compared to other classy sorting algorithms with same computational complexity heap sort performs quite slowly in real time application.

Many variants of heap sort were developed, and most of the methods are primarily minimizing the number of comparisons and the running time. The Bottom-Up-Heap Sort [7] beats the quick sort. Generally heap sort works slower than other sorting methods with same computational complexity. However, heap sort is best suited for large data sets for as it does not use recursion.

vii. Radix Sort

Radix sort is a linear sorting algorithm and doesn't compare any data-items unlike other sorting methods such as insertion sort and quick sort. Radix sort works by sorting data-items with keys. Radix sort is a stable sort as it maintains the relative order of data-items with similar keys.

Two types of radix sorts are used in practice, such as least-significant-digit (LSD) and most-significant-digit (MSD). The Least-significant-digit method is implemented by processing the integer representation starting from the least digit and shift in order to obtain the most significant digit. Likewise the Most significant digit works in the opposite-way.

The efficiency of radix sort is difficult to describe when compared to other classy algorithms. In theory, the average run time complexity of radix sort is \(O(d \times n)\). The practical efficiency depends on the value of \(d\), where \(d\) is the number of digits in each array value. If the number of digits is constant for all the values in the array, then the performance of radix sort would be more efficient than other sophisticated algorithms. However, with different number of digits in array values, the algorithm required \(O(n \log_{10} n)\) computational time, which is almost identical to other classy algorithms such as quick sort and merge sort. Therefore radix sort would become inefficient for applications with distinct array values. It is known that breaking the \(O(n \log_{10} n)\) computational complexity is difficult for any key comparison method, however the radix sort has this potential to sort \(N\) keys in \(O(n)\) operations. Because of this, many variants are advanced such as A Fast Radix Sort [8] which is faster than any quick sort variants and Forward Radix Sort [9] which combines the advantages of the conventional left-to-right and right-to-left radix sort, thus it will work very well in practice. One of the advantages of radix sort is that its efficiency does not reflect upon with the type and size of input data being sorted. Likewise, when compared to the other integer sorting algorithm, radix sort can handle larger keys more efficiently. On the other hand it is less flexible and complex to program for a wide variety of functionality and requirements. Moreover radix sort
takes more memory space than other classy algorithm, hence a problem with the memory storage space is often considered as primary concern then radix sort won't be a good choice.

Modern sorting techniques and their performance analysis

Timsort [10] is a sorting algorithm that is efficient for real-world data and not created in an academic laboratory. Tim Peters created Timsort for the Python programming language in 2001. Timsort first analyses the list it is sorting and then selects an approach based on the analysis of the list. Python, Android Platform, and in GNU Octave uses Tim sort as default sorting algorithm for their library.

Timsort’s big O notation is $O(\log_2 n)$. Timsort’s sorting time is the same as Merge sort, which is faster than most of the other sorts you might know. Timsort actually makes use of Insertion sort and Merge sort.

Peters designed Timsort to use already-ordered elements that exist in most real-world data-sets. It calls these already ordered elements “natural runs”. It iterates over the data collecting the elements into runs and simultaneously merging those runs together into one.

Timsort outperforms if:

- Data-sets have preexisting internal structure
- stable sort

Comparative study of most widely used algorithms

Below are some common Big-O functions while analyzing algorithms.

- $O(1)$ - constant time
- $O(\log_2 n)$ - logarithmic time
- $O((\log_2 n)c)$ - polylogarithmic time
- $O(n)$ - linear time
- $O(n^2)$ - quadratic time
- $O(n^c)$ - polynomial time
- $O(c^n)$ - exponential time
- $O(n!)$ - factorial time

Here is the model graph representing Big-O complexity of some functions [11]
Conclusion

This study compares various traditional sorting algorithms like bubble, insertion, selection, merge and quick sorts and suggests trade-offs for selecting sorting algorithm based on input size and distribution of data-items. Invariably it’s been found that merge and quick sorts out-perform for larger data-sets, where as insertion and selection sorts are best suited for smaller data-sets. This paper has discussed a new approach devised by combining insertion and merge sort called Tim’s sort which is faster than any of the traditional sorting technique. Finally it has provided the behavior of sorting algorithms along with their time complexities.

References

[7] Ingo Wegener, "Bottom-up-heap sort, a new variant of heap sort beating on average quick
sort," Lecture Notes in Computer Science, vol. 452, p. 516


