Comparison in Time Complexity Performance of Sorting Algorithms

Introduction:

Sorting is one of the most well studied open problem in Computer Science. The process of re-arranging the order of data into a particular order is called sorting. It might be one of the basic tasks in other applications but is sure considered a fundamental task

It is not always possible to tell within the sorting algorithms that which of them is better than the others. Performance is dependent on other factors as well like the data being sorted.

Aim:

The aim of this paper is to compare popularly and widely used sorting algorithms. This will also be a survey based on scholarly articles and possible practical applications. The algorithms included in this paper would be explained and compared for their performance in time complexity. Sorting algorithm would be discussed for their reported advantages and disadvantages.

The goal of this survey is to record and analyze data that makes a detailed comparison of the practical efficiency of the algorithms. Lastly, we will be comparing the algorithms on their complexity in worst case and best case.

Limitation:

Since, so many sorting algorithms are out there, it is not possible to survey all of them in this study. Hence, only some basic and most popular algorithms are considered for analysis.

Analysis of Sorting Algorithms

✓ *Selection Sort*

Selection sort is the most basic of the comparison based sorting algorithms. Its algorithm iterates through all the n data values to find the smallest data which takes n-1 comparisons, next smallest value will take n-2 comparisons and so on it will take in total (n-1) +(n-2) +(n-3) +(n-4) ……1 = n(n-1)/2 $\in O(n^2)$. Selection sort has same time complexity as of insertion sort which we will discuss later but selection sort has an advantage if the write operation is significantly expensive than read operation because insertion sorts make less more swaps($O(n^2)$) than the selection sort swaps($O(n)$).

✓ *Insertion Sort*

Insertion Sort as its name suggests inserts the data onto its ordered place in the final array. For its simplest implementation one can use two arrays one for the final sorted array and one as an input to work on but it can be implemented as an in-place algorithm i.e. using the same input array for repeated swapping and resulting in a sorted array at the end of the procedure, this saves memory space. Although its time complexity is $O(n^2)$ but it is practically faster than Selection Sort and reported to have been a constant time for data set less than 100. Insertion sort has also given path to a hybrid sophisticated sorting algorithm Timsort[1].

✓ *Quick Sort*

Quick Sort is a divide and conquer type sorting algorithm which sorts using comparisons just like Selection and Insertion. Quick sort is the fastest running algorithm on average case i.e. O(n log n) when compared to other sophisticated algorithms. It divides the array into two parts based on a pivot element and further divides divided arrays on new pivots, here divide part is crucial and conquer part is trivial. Some variants of Quick Sort are much more efficient than standard Quick Sort. Unbalanced Partition of the input data set can lead to a slow running time (worst case $O(n^2)$) but a good pivot selection leads to the quickest sorting. Quick Sort is not a stable sort so it is not a good choice if data set contains equal data but if time is significantly important and Input data size is large then quick sort is one of the best option to go with.

✓ *Merge Sort*

Merge Sort is also a divide and conquer type sorting algorithm just like quick sort. It is also dependent on comparison for sorting. In Merge Sort the divide part is trivial and conquer part is crucial in terms of Merge Sort it is called 'Merge' step because it merges two sorted arrays into one sorted array. Running time complexity for Merge Sort is also O(n log n). It is comparable to Quick sort in execution time but quick a well implemented quick sort always beats merge sort. However, Merge Sort has its own advantages i.e. it is stable sort is preferred over quick sort whenever the data set has equal values.

✓ *Counting Sort*

Counting Sort is not a comparison based sorting also it is an integer sorting algorithm. Counting Sort being a non-comparison based algorithm can sort in O(n) time complexity although it is not used as a separate sorting algorithm but is used as a subroutine within another sort called Radix sort. Counting sort is a stable sort but not an in-place algorithm however it can be modified to work as an in-place algorithm but we would be trading off the stability for that. Its practical use is with Radix sort only which is used as a card sorting or date/month/year sorting.

Time Complexity Comparisons in the covered Sorting Algorithms

Algorithms	Averages Case	Worst Case
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Quick Sort	O(n log n)	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)
Counting Sort	O(n)	O(n)

Conclusion:

This survey paper discusses the comparison of 5 sorting techniques Selection Sort, Insertion Sort, Quick Sort, Merge Sort and Counting Sort. For small input sizes, we have Insertion Sort for large Input sizes we have Quick and Merge Sort, if a linear time sorting is required we could use counting sort. Hence, we could just conclude that comparing various sorting algorithms to find the best one based on time complexity might not give us a satisfying result as all these sorts have their own advantages which we can exploit if we are willing to trade off on some other parameter.

Reference:

[1] https://en.wikipedia.org/wiki/Timsort

[2] http://www8.cs.umu.se/education/examina/Rapporter/AshokKumarKarunanithi-final.pdf

[3] http://gdeepak.com/pubs/Selection%20of%20best%20sorting%20algorithm.pdf

[4] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009

[5] https://en.wikipedia.org/wiki/Sorting_algorithm

[6] https://en.wikipedia.org/wiki/Divide_and_conquer_algorithms