International Journal of Advanced Computer Research
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
Volume-5 Issue-21 December-2015

A Comparative Analysis of Deterministic Sorting Algorithms based on Runtime and Count of Various Operations

Reyha Verma[1] and Jasbir Singh[2*]
Department of Computer Science, National Institute of Technology, Srinagar, India[1]
Department of Computer Science & IT, University of Jammu, Jammu, India[2]

Received: 24-September-2015; Revised: 01-November-2015; Accepted: 05-November-2015

Abstract

Sorting algorithms find its application in many practical fields of Computer Science. Efficient solving of sorting problem has attracted a great deal of research as it optimizes other algorithms also. The main factor which is taken into consideration while determining the efficiency of a sorting algorithm is the time complexity. Mostly the execution time of algorithms is investigated and compared for analyzing time complexity. This paper presents a comparative analysis of deterministic sorting algorithms. Time complexity of six different algorithms namely, Selection sort, Bubble sort, Insertion sort, Quicksort, Heapsort and Mergesort is determined in terms of number of comparisons, swaps and assignment operations in addition to average execution time. Also, the performance of these algorithms on fully and almost sorted lists was also analyzed. The study indicates that determining the operation's count is essential for analyzing time complexity especially when algorithms are theoretically analyzed.

Keywords

Deterministic sorting, Time complexity, Bubble sort, Selection sort, Quicksort.

1. Introduction

An algorithm that takes as input a sequence of numbers and outputs an ordered permutation of the input sequence is termed as a sorting algorithm [1]. The order may be a numerical, lexicographical or any other order. Information for many applications in computer science is managed and retrieved easily if the data is kept sorted.

Processing data in a certain specific order is more useful than processing randomized data[2]. Moreover, certain applications which require sorted input data tend to be more optimal with efficient sorting.

All the algorithms analyzed in the present paper are having the property that the output of every operation is uniquely defined and predictable. Algorithms with this property are termed as deterministic algorithms [3].

A number of deterministic sorting algorithms have been developed in order to enhance efficiency. Primarily the efficiency of a sorting algorithm is determined by its time complexity. The time complexity is the amount of computer time needed by the algorithm to run to completion. It is estimated theoretically by determining the number of comparisons and swaps [4].

In this paper the time complexity of algorithms namely, Selection sort, Bubble sort, Insertion sort, Quicksort, Heapsort and Mergesort is determined for unsorted, almost sorted and fully sorted lists. The parameters used for analysis are average execution time, number of comparisons, swaps and assignment operations. The objective is to ascertain the efficient algorithm and the effect of comparisons, swaps and assignment operations on the average runtime.

2. Literature Review

Solutions to sorting problems have attracted a great deal of research in the recent years and in this process many sorting algorithms have originated with improved efficiency. Certain algorithms perform more efficiently under certain situations. Over the years researchers have been comparing and analyzing the sorting algorithms to determine their applicability to applications. An example can be found in [6],

*Author for correspondence

where the authors had designed a new sorting algorithm as index sort.

P. Adhikari [2], while comparing various performance factors among selection sort and shell sort algorithm, concludes that shell sort gives better performance and that both the algorithms cannot be used for large arrays. Pasetto and Akhriev [7] provide a comprehensive analysis of the performance of parallel sorting algorithms on modern multi-core hardware. Several best known general-purpose algorithms were considered. The authors provided an insight as to which algorithm is most suited for a specific application along with the shortcomings and advantages of each algorithm.

In [8] the authors had compared the performance of selection sort and quicksort algorithm for sorting integer and string arrays. The algorithms were analyzed on random data and results indicated that selection sort performs better than quicksort and string arrays have lesser processing time than integer arrays. In [9] a statistical comparative study of sorting algorithms, viz. Quick sort, Heap sort and K-sort with asymptotically optimal average case complexity, has been reported.

Based on the studies as available in the literature, the algorithms have been compared by obtaining the corresponding statistical bounds while subjecting these procedures over the randomly generated data from Binomial, Uniform and Poisson distribution. The parameterized complexity analysis is also provided. The performance of the new algorithm is compared with four different sorting algorithms. The authors have concluded that Index Sort algorithm works well for all length of input values.

3. Methodology

Time complexity of six different algorithms namely, Selection sort, Bubble sort, Insertion sort, Quicksort, Heapsort and Mergesort are analyzed. All these algorithms are a comparison sort. A comparison sort arranges data in sorted order by comparing data items [1]. Each algorithm uses a different mechanism to sort data which is illustrated in the following pseudocodes:

Selection Sort (A)

```
for i ← 1 to length[A] – 1
do min ← i
        for j ← i + 1 to length[A]
            do if A[j] < A[min]
              then  min ← j
exchange A[i] & A[min]
```

Bubble Sort(A) [2]

```
for i ← 1 to length[A]
        do for j ← length[A] down to i + 1
                do if A[j] < A[j - 1]
                    then exchange A[j] & A[j - 1]
```

Insertion Sort(A) [2]

```
for j ← 2 to length[A]
    do key ← A[j]
    Insert A[j] into the sorted sequence A[1 :  j - 1].
     i ← j - 1
     while i > 0 and A[i] > key
          do A[i + 1] ← A[i]
          i ← i - 1
     A[i + 1] ← key
```

Quicksort(A, P, R) [2]

```
if p < r
  then q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q - 1)
        QUICKSORT(A, q + 1, r)
```

Mergesort(A, P, R) [2]

```
if p < r
  then q ← (p + r)/2
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

Heapsort(A) [2]

```
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
        do exchange A[1] ↔ A[i]
            heap-size[A] ← heap-size[A] - 1
            MAX-HEAPIFY(A, 1)
```

In selection sort the sorting mechanism is to find the minimum value in the list and exchange it with the first element. The process is repeated with the rest of the list. The time complexity of selection sort is O (n^2) [1][3][5].

In Bubble sort sorting is done by comparing each pair of adjacent elements in the list and swapping them if not in order. In each pass the last element gets to its correct position. The time complexity of bubble sort is O (n^2).

In Insertion sort the sorting mechanism is to insert the elements one by one into their right position into a new sorted list. The time complexity of insertion sort is O (n^2).

Quicksort is based on divide and conquer design technique. A pivot element is chosen to partition the list such that smaller elements are placed before the pivot and greater elements are placed after it. The process is applied to the sublists recursively. The time complexity of quicksort is O ($nlog_2n$) [1][3][5].

Mergesort is also a divide and conquer algorithm which divides the list until it can be sorted easily and then merges the sorted lists to obtain the complete sorted list. The time complexity of merge sort is O ($nlog_2n$).

Heapsort accomplishes sorting by arranging the input data into a heap. A heap is a complete binary tree with the greatest or smallest element at the root node. Heapsort creates a sorted list by deleting the root and placing it at end of the list. The heap is then rearranged and the process is repeated. The time complexity of heap sort is O ($nlog_2n$).

4. Implementation

All the sorting algorithms are implemented in java using netbeans IDE. The algorithms are executed on a window 7 professional machine having Intel(R) core(TM) i5 CPU M 560@ 2.67 GHz and installed memory (RAM) 2.00 GB. Input Dataset for the algorithms is randomly chosen using the algorithms implemented by the class Random of java's util package. This class generates uniformly distributed pseudorandom numbers using a linear congruential formula [4] and starts with a 48 bit seed. For runtime analysis the number of inputs was fixed to 10000 and number of runs (executions) was varied from 500 to 10000 in 20 steps. For determining operation counts the number of inputs was fixed to 1000 and number of runs (executions) was varied from 500 to 10000 in 20 steps. A ninety percent sorted list was taken as almost sorted list. An additional Java API, JFreeChart was used for graphical representation of results obtained. JFreeChart is a free chart library for the Java(tm) platform.

5. Results

Time complexity of all the six algorithms namely, Selection sort, Bubble sort, Insertion sort, Quicksort, Heapsort and Mergesort is determined in terms of no. of comparisons, no. of swaps, no. of assignment operations and average execution time. In addition for all the algorithms the number of comparisons and number of swaps was obtained fully sorted lists as well as almost sorted list.

The average execution time obtained for the algorithms is shown in figure 1.

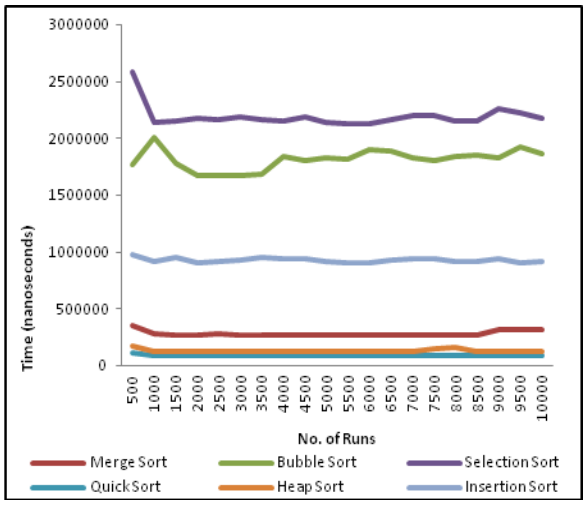

Figure 1: Average Execution Time of Algorithms

The average number of comparisons obtained for the algorithms is shown in figure 2.

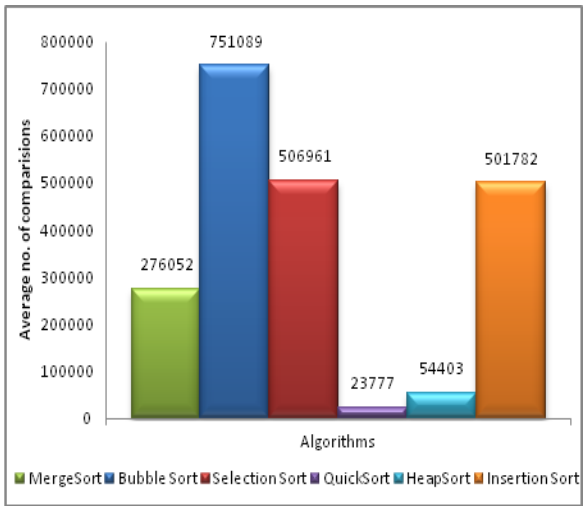

Figure 2: Avg. no. of comparisons of Algorithms

The average number of swaps obtained for the algorithms is shown in figure 3.

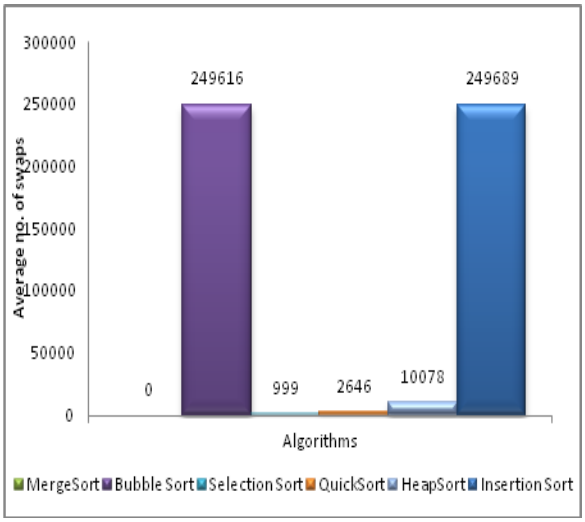

Figure 3: Avg. no. of swaps made by Algorithms

The average number of assignments obtained for the algorithms is shown in figure 4.

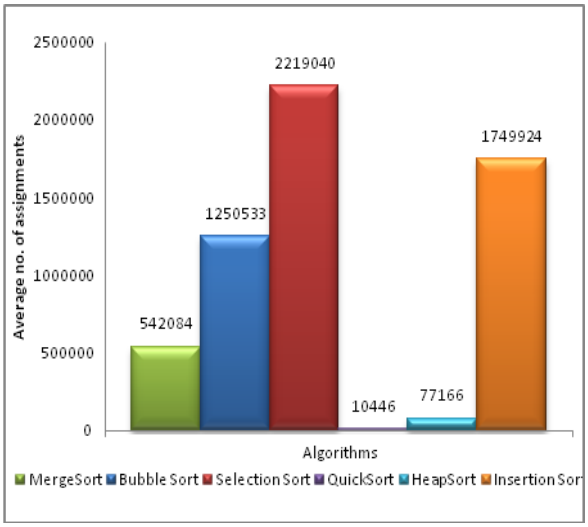

Figure 4: Avg. assignments made by Algorithms

The average number of comparisons obtained for the algorithms for fully sorted list is shown in figure 5.

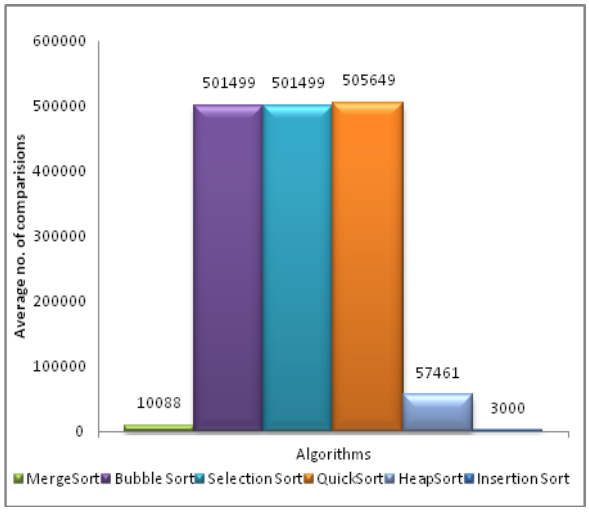

Figure 5: Avg. comparisons for fully sorted lists

The average number of swaps obtained for the algorithms for fully sorted list is shown in figure 6.

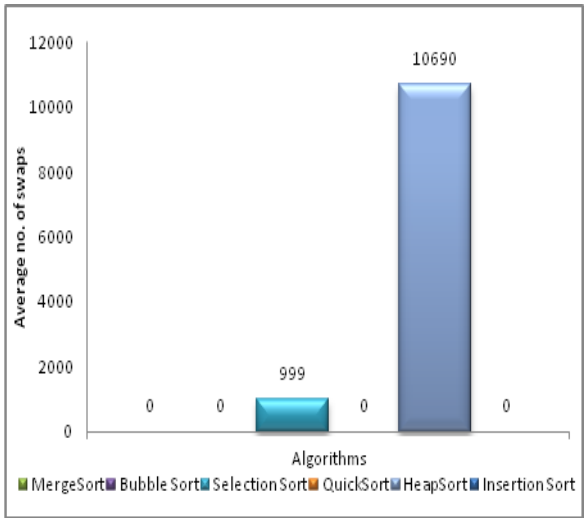

Figure 6: Avg. no. of swaps for fully sorted lists

The average number of comparisons obtained for the algorithms for almost sorted list is shown in figure 7.

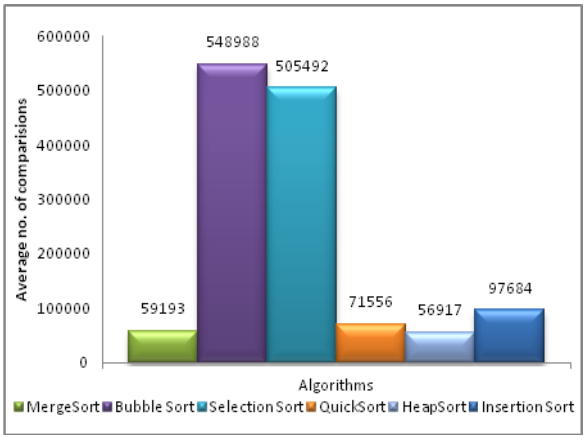

Figure 7: Avg. comparisons for almost sorted lists

The average number of swaps obtained for the algorithms for almost sorted list is shown in figure 8.

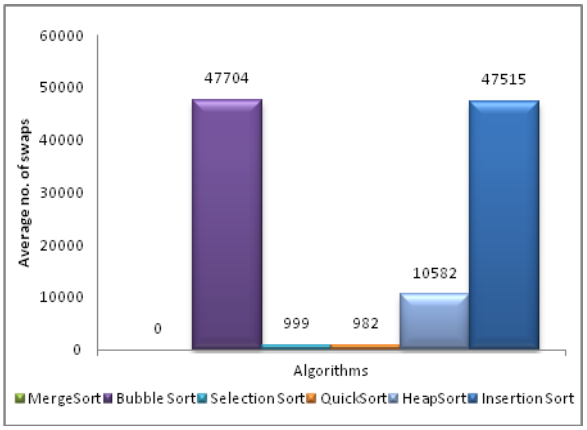

Figure 8: Avg. no. of swaps for almost sorted lists

All the six algorithms are ranked according to their performance in the results obtained. The algorithm with lower value is ranked high. The ranking of algorithms is listed in Table 1.

Table 1: Ranking of Algorithms

Algorithm	Avg. No. of Comparisons	Avg. No. of Swaps	Avg. No. of Assignments	Avg. Run Time
Selection Sort	5	2	6	6
Bubble Sort	6	5	4	5
Insertion Sort	4	6	5	4
Quicksort	1	3	1	1
Merge Sort	3	1	3	3
Heap Sort	2	4	2	2

In the present investigation, Quicksort is observed as the better algorithm then the rest five algorithms in terms of average no. of comparisons, assignment operations and execution time. Also, on random data bubble sort makes more number of comparisons than selection sort. The number of comparisons tends to increase in case of quicksort and decrease in case of Mergesort when the list is fully sorted. Also, a slight increase in no. of comparisons is also observed in Heapsort.

One interesting observation had been made in case of bubble sort and selection sort. Selection sort had lesser no. of comparisons and swaps as compared to bubble sort algorithm but had larger execution time than bubble sort. This is attributed to the large number of assignment operations in selection sort in comparison to that in bubble sort. Thus, indicating the importance of determining count of various operations while analyzing time complexity of Algorithms.

6. Conclusion and Future Work

In this paper we analyzed six different deterministic algorithms on the basis of average number of comparisons, swaps and assignment operations along with the average runtime. Quicksort had been observed to be the better algorithm. It has also been observed that it is essential to ascertain the count of each basic operation for theoretically analyzing time complexity of algorithms.

Future work will continue in the direction of analyzing sorting algorithms with larger data sets. Efforts will be to use more algorithms and to take into account the other factors like memory usage, stability and adaptability while analyzing algorithms.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", 2nd ed., The MIT Press, McGraw Hill, 2001.
[2] P. Adikari, "Review on Sorting Algorithms: A comparative study on two sorting algorithms", Mississipi state university, Mississippi 2007.
[3] E. Horowitz, S. Sahni and S. Rajasekaran, "Fundamentals of Computer Algorithms", 2nd ed., Universities Press, 2009.
[4] D. E. Knuth, "The Art of Computer Programming", Sorting and Searching, Volume 3, Addison Wesley, april 1998.

[5] J. Hubbard, "Shaum's Outline of Data Structures with Java", 2nd ed., McGraw Hill education, 2009.
[6] Bharadwaj, Ashutosh, and Shailendra Mishra. "Comparison of Sorting Algorithms based on Input Sequences." International Journal of Computer Applications 78.14 (2013): 7-10.
[7] D. Pasetto and A. Akhriev, "Comparative Study of Parallel Sort Algorithms", IBM Dublin Research Lab, Mulhuddart, Dublin 15, Ireland.
[8] A. M. Aliyu and P. B. Zirra, "A Comparative Analysis of Sorting Algorithms on Integer and Character Arrays", The International Journal of Engineering and Science (IJES), volume 2,Issue-7, p 25-30, 2013.
[9] A. Kumari, N. K. Singh and S. Chakraborty, "A statistical comparative study of some sorting algorithms", International Journal in Foundations of Computer Science & Technology (IJFCST), Vol.5, No.4, July 2015.

Reyha Verma (Jammu, India, 13/09/2015) is a 3rd year B.Tech student, Department of Computer Science and Engineering, National Institute of Technology, Hazratbal , Srinagar (J&K State).

Jasbir Singh (Jammu, India, 13/09/2015) is MCA from University of Jammu, India and currently working as Assistant Professor in Department of Computer Science & IT in University of Jammu. His research interest is Network Security and Analysis Design of Algorithms.
Email: jasbirmca@gmail.com