Selection of Best Comparison Based Sorting Algorithm

Wasim Abbas
Punjab School Education Department
Pakistan
wasimabbasjoyia@gmail.com

ABSTRACT: *Sorting is very common problem in the fields of computer science. Many sorting algorithms are available for sorting. Every sorting algorithm has its own advantages and disadvantages. Detailed analysis and comparison of the performance of these algorithms provided the fact that there is no single best sorting algorithm for every sorting problem. All algorithms are problem specific.*

Some factors needs to be considered in choosing the best algorithm include the size of the list that needs to be sorted , requirements of programming efforts, availability of size of main memory and secondary memory, distribution of elements in the list , duplication of elements in the list and up to what extent list is pre-sorted.

Keywords: Sorting, complexity, stable, internal sorting, efficiency, worst case, average case, best case.

Received: 1 June 2016, Revised 5 July 2016, Accepted 15 July 2016

1. Introduction

In computer science and mathematics sorting algorithm are used to arrange elements in numerical or lexicographical order. Sorting is the most studied problems in computer science because it is combinatorial problem which has many interesting and diverse solutions.[3].

Due to its importance and diversity sorting is still remained hot topic in research field. It is always difficult to find best sorting algorithm, research is going on this topic.my research is also based on this topic that which one is best sorting algorithms inevery sorting problem.

There are no. of sorting algorithms which offer various trades offs in simplicity and in efficiency, in speed and space. This paper will focused only on comparison based sorting algorithms and will present the comparison between different comparison based sorting algorithms and will suggest that which is the best comparison based sorting algorithms for which problem, because

large no. of programmer use comparison based sorting algorithms. comparison based sorting algorithms are more general purpose than non comparison based sorting algorithms, non comparison based sorting algorithms do poorly for non fixed lengths inputs, also non comparison based sorting algorithms often need many assumptions about the input data (integers from small range for count sort, uniformly distributed for bucket sort, etc.).

Even comparison based sorting algorithms are also in huge numbers and to find the best algorithm from them is not an easy task because all algorithms are problem specific. Algorithm performance is based on type of problem, for example some sorting algorithms do well with small no. of elements. Where some are best in the situation where elements are in larger numbers. Some sorting algorithms perform well where data is duplicated.

Actually many factors need to be consider while choosing the best sorting algorithms. The remainder of this paper is structured as follows,

Section 2: Criteria to choose best sorting algorithm.

Section3: Classification of Sorting Algorithms.

Section 4: Famous comparison based sorting algorithms.

Section 5: comparison of different sorting algorithms.

Section 6: time performance Charts of sorting algorithms

Section 7: conclusion [1-7]

2. Criteria To Choose Best Comparison Based Sorting Alogorithms

To define the best comparison based sorting algorithms is not simple, but we can say the best comparison based sorting algorithms is that one uses the minimum no. of comparisons , moves and exchanges for sorting.[2].

All sorting algorithms are problem specific means that some are best in one situation and some algorithms are best in other situation like Insertion sort is widely used for small data sets, where as for larger data sets an asymptotically efficient sort is used like heap sort, merge sort, or quicksort. Efficient implementations generally use a hybrid algorithm, combining an asymptotically efficient algorithm for the overall sort with insertion sort for small lists at the bottom of a recursion. Highly tuned implementations use more sophisticated variants, such as Timsort (merge sort, insertion sort, and additional logic), used in Android, Java, and Python, and introsort (quicksort and heap sort), used (in variant forms) in some C++ sort implementations and in .NET.

So, the criteria to choose the best algorithms is based on situation or based on problem. This paper gives the comparison between famous comparison based algorithms and this comparison provides the basis for choosing best algorithm according to the situation. [1][2][3].

3. Classification of Sorting Algorithms

Sorting algorithms are often classified by:

- Sorting algorithm are classified on the basis on that which sorting algorithm has what Computational complexity in worst case, average case and best case..
- Algorithms are also classified on the basis of Memory usage and use of other computer resources.
- Recursion. Some algorithms are either recursive or non-recursive, while others may be both (e.g., merge sort).
- Some algorithms are stable whether other are not stable, stable sorting algorithms maintain the relative order of records with equal keys.
- Some algorithms are comparison based while other is not comparison based sorting algorithms. A comparison based sorting algorithms uses comparison operator to compare the two values.
- Algorithms are also categorized on the basis of method like insertion, exchange, selection, merging etc.
- Whether the algorithm is serial or parallel.

- Adaptability: Whether or not the algorithms is adaptable. The adaptive sorting algorithms take into account that up to what extent input is already sorted. [4-9]

4. Famous Comparison Based Sorting Algorithms

Some of the famous algorithms arte as follows.

1) Insertion sort:

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists, and often is used as part of more sophisticated algorithms. It is a sorting algorithm that belongs to the family of comparison sorting. . It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list. Insertion sort has a time complexity of O (n^2) but is known to be efficient on data sets which are already substantially sorted. Its average complexity is n^2=4 and linear in the best case. Insertion sort is an in-place algorithm that requires a constant amount O(1) of memory space. Shell sort is a variant of insertion sort that is more efficient for larger lists.

2) Merge sort:

Belongs to the family of comparison-based sorting. Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements and swapping them if the first should come after the second. It then merges each of the resulting lists of two into lists of four, then merges those lists of four, and so on; until at last two lists are merged into the final sorted list. It has an average and worst-case performance of O(n log n). Unfortunately, Merge sort requires three times the memory of in-place algorithms such as Insertionsort. Merge sort has gaining popularity for its practical implementations in the sophisticated algorithm Timsort, which is used for the standard sort routine in the programming languages Python and Java (as ofJDK7. Merge sort itself is the standard routine in Perl among others, and has been used in Java at least since 2000 in JDK1.3.

3) Selection sort:

Belongs to the family of in-place comparison sorting. It typically searches for the minimum value, exchanges it with the value in the first position and repeats the first two steps for the remaining list. It does no more than n swaps, and thus is useful where swapping is very expensive. On average Selection sort has an O (n2) complexity that makes it inefficient on large lists. Selection sort typically outperforms Bubble sort but is generally outperformed by Insertion sort.

4) Bubble sort:

Is a simple sorting algorithm that belongs to the family of comparison sorting? Bubble sort, and variants such as the cocktail sort, are simple but highly inefficient sorts. They are thus frequently seen in introductory texts, and are of some theoretical interest due to ease of analysis, but they are rarely used in practice, and primarily of recreational interest. It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. Bubble sort has a worst-case complexity O(n2) and in the best case O(n). Its memory complexity is O (1).

5) Heap sort:

Is a comparison-based sorting algorithm, and is part of the Selection sort family. Heap sort is a much more efficient version of selection sort. It also works by determining the largest (or smallest) element of the list, placing that at the end (or beginning) of the list, then continuing with the rest of the list, but accomplishes this task efficiently by using a data structure called a heap, a special type of binary tree. Once the data list has been made into a heap, the root node is guaranteed to be the largest (or smallest) element. When it is removed and placed at the end of the list, the heap is rearranged so the largest element remaining moves to the root. Although somewhat slower in practice on most machines than a good implementation of Quicksort, it has the advantage of a worst-case O (n log n) runtime.

6) Quicksort belongs:

To the family of exchange sorting. Quicksort is a divide and conquer algorithm which relies on a partition operation: to partition an array an element called a pivot is selected. All elements smaller than the pivot is moved before it and all greater elements are moved after it. This can be done efficiently in linear time and in-place. The lesser and greater sub lists are then recursively sorted. This yields average time complexity of O(n log n), with low overhead, and thus this is a popular algorithm.

7) Shell sort:

It improves upon bubble sort and insertion sort by moving out of order elements more than one position at a time. One implementation can be described as arranging the data sequence in a two-dimensional array and then sorting the columns of the array using insertion sort. It is a generalization of Insertion sort. The algorithm belongs to the family of in-place sorting but is regarded to be unstable. The algorithm performs O (n^2) comparisons and exchanges in the worst case, but can be improved to O($n \log^2 n$). This is worse than the optimal comparison sorts, which are O (n log n). Shell sort improves Insertion sort by comparing elements separated by a gap of several positions. This lets an element take "bigger steps" toward its expected position. Multiple passes over the data are taken with smaller and smaller gap sizes. The last step of Shell sort is a plain Insertion sort, but by then, the array. [1][2][3].

5. Comparison of Algorithms

Following table 1 shows the comparison of different comparison based sorting algorithms in this table, n shows the number of records to be sorted. The columns Best, Average and Worst give the time complexity in each case, under the assumption that the length of each key is constant, and that therefore all comparisons, swaps, and other needed operations would take constant time. Memory denotes the amount of auxiliary storage needed beyond that used by the list itself, under the same assumption. The run times and the memory requirements listed below should be understood to be inside big O notation. These are all comparison sorts, and so cannot perform better than O (n log n) in the average or worst case. [9]

Name	Best	Average	Worst	Memory	Stable
Insertion sort	n	n^2	n^2	1	Yes
Quicksort	n log n	n log n	n^2	(log n) on average, worst case is (n); S edgwick variation is (log n) worst case	No
Heapsort	n log n	n log n	n log n	1	No
Merge sort	n log n	n log n	n log n	n worst case	Yes
Selection sort	n^2	n^2	n^2	1	No
Shell sort	n	$n \log^2 n$	Depends on gap sequence; best known is $n \log^2 n$	1	No
Bubble sort	n	n^2	n^2	1	Yes

Name	Method	Positives/Negatives	Usage suitability
Insertion sort	Insertion	It is good for small list	It is useful where data is small and easily fit into p main memory and can be randomly accessed and no extra space is required to

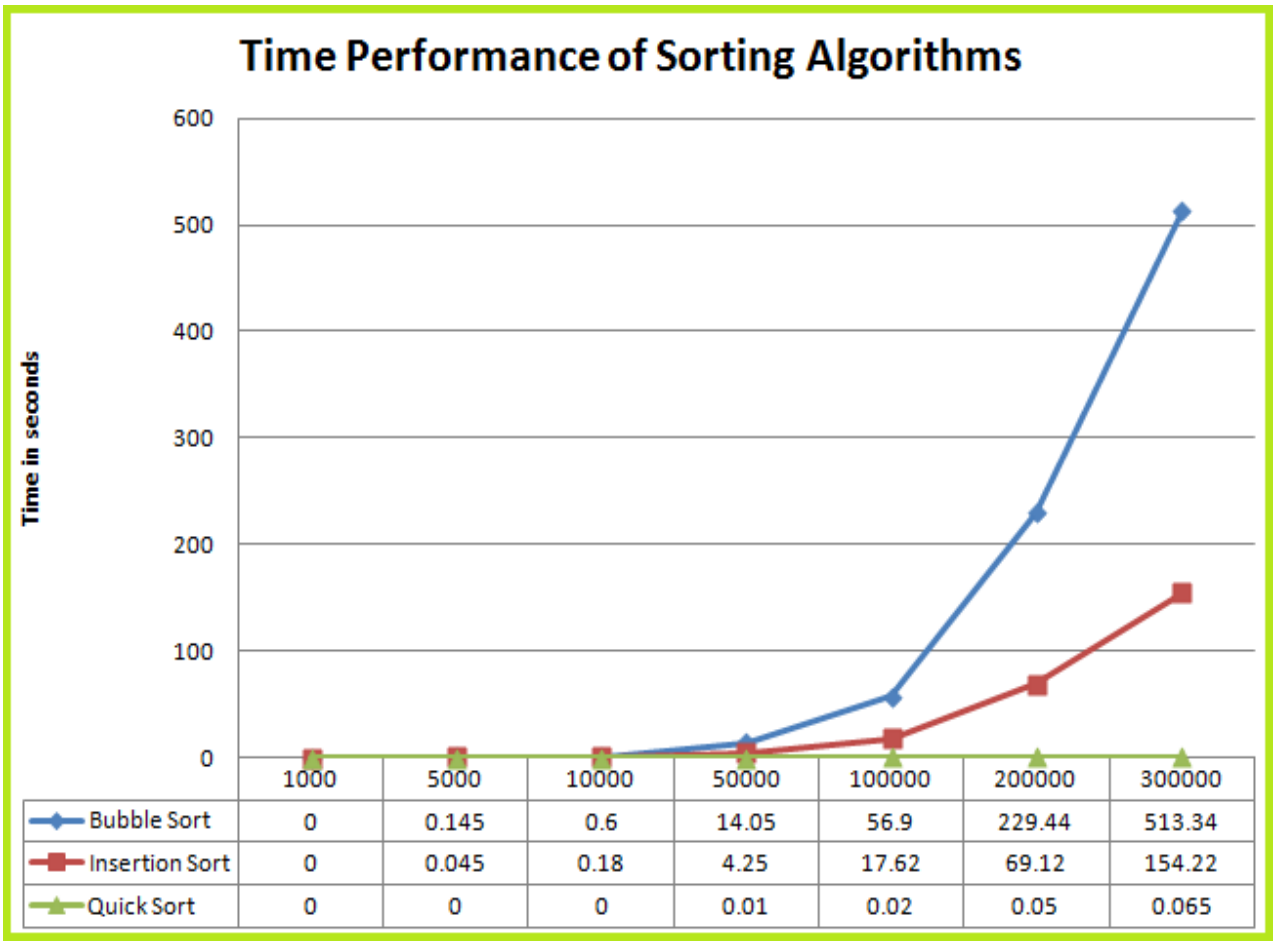
			sort the record (internal sorting). It also suits when element are repeated in the list, and sort the list in order to maintain the relative order of the record with equal keys.
Quicksort	Partitioning	It uses O (log n) space. fastest sorting algorithm	It suits when inputs are too large. It also suits when element are repeated in the list, and sort thee list in their relative order are not maintained with equal keys.
Heapsort	Selection	It does not require recursion and extra memory buffer ,it is slower than quick and merge sort	It suits when inputs are too large. It also suits when element are repeated in the list, and sort the list in their relative order are not maintained with equal keys
Merge sort	Merging	It fast recursive sorting algorithm which is best for large lists	It suits when inputs are too large. It also suits when element are repeated in the list. , and sort the list in order to maintain the relative order of the record with equal keys.
Selection sort	Selection	It is slow Algorithm; it improves the performance of bubble sort.	It is useful where data is small and easily fit into p main memory and can be randomly accessed and no extra space is required to sort the record (internal sorting It also suits when element are repeated in the list, and sort the list in their relative order are not maintained with equal keys
Shell sort	Insertion	Small code size, no use of call stack, reasonably fast, useful where memory is at a premium such as embedded and older	It suits when inputs are too large. It also suits when element are repeated in the list, and sort the list in their relative order are not maintained with equal keys

Bubble sort	Exchanging	Straight forward, simple and slow, Small code size.	It is useful where data is small and easily fit into p main memory and can be randomly accessed and no extra space is required to sort the record (internal sorting. It also suits when element are repeated in the list. , and sort the list in order to maintain the relative order of the record with equal keys.

Table 2. Shows advantages and disadvantages and also show in which situation which is the best sorting algorithm and which method is used for sorting. [2][9]

6. Time performance Charts of Sorting Algorithms

Following Charts Char1, chart 2 and chart 3 shows the time performance of different sorting algorithm. [1-9].



	1000	5000	10000	50000	100000	200000	300000
Bubble Sort	0	0.145	0.6	14.05	56.9	229.44	513.34
Insertion Sort	0	0.045	0.18	4.25	17.62	69.12	154.22
Quick Sort	0	0	0	0.01	0.02	0.05	0.065

Chart 1.

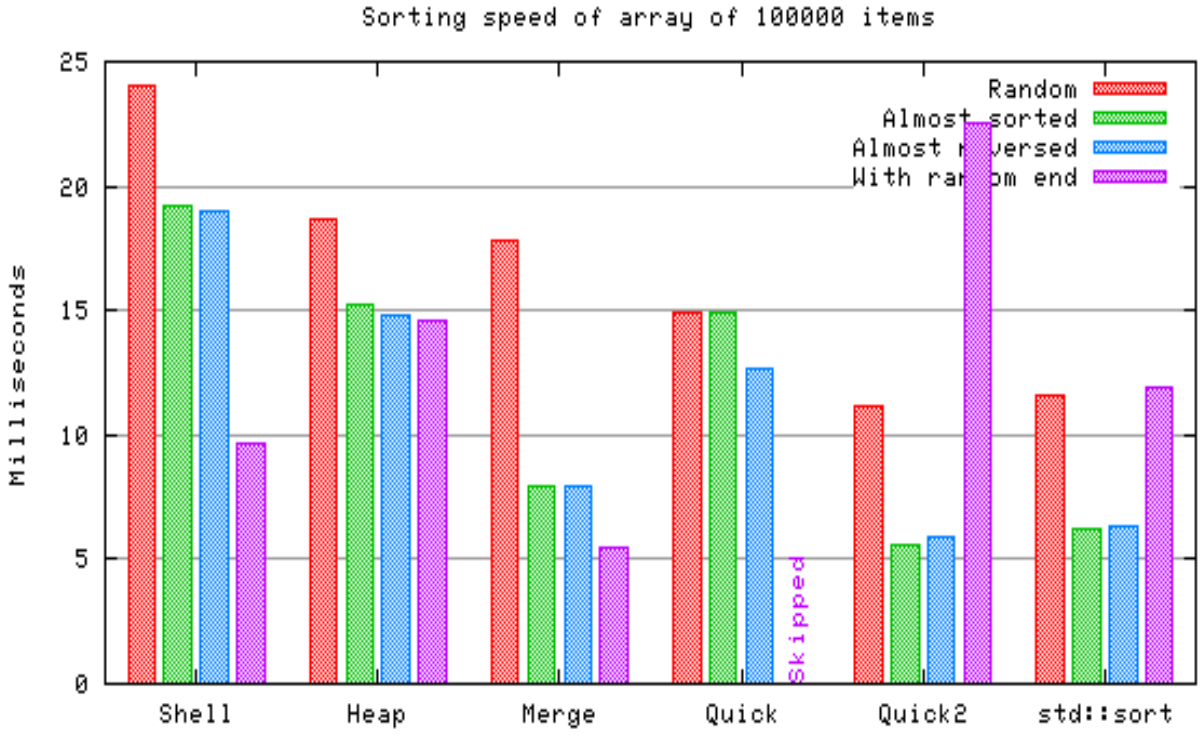


Chart 2

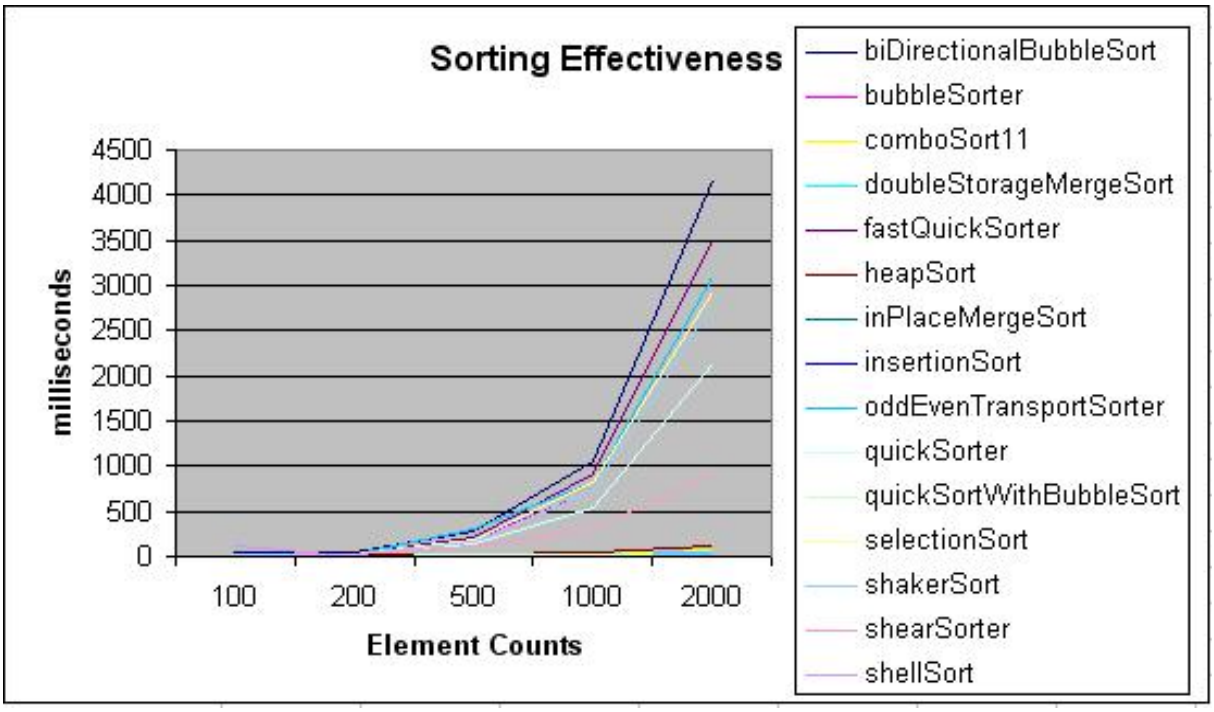


Chart 3

7. Conclusion

The overall goal of my research is to discuss the best comparison based algorithm for specific sorting problem. I have given the comparison of different sorting algorithm in tabular form, in charts form and in text form. From this comparison it is very much clear that we cannot pick single best algorithm for every sorting problem. These all algorithm are problem specific, we have to choose the algorithm on the basis of type of the problem, which sorting algorithm efficiently performs in which situation depends on many factors like size of list, distribution of elements in list and up to what extent list is pre-sorted etc.

References

[1] Sultanullah Jadoon, Salman Faiz, Salim ur Rehman, Hamid Jan "Design & Analysis of Optimized Selection Sort Algorithm"

[2] Adiyta Dev., Deepak Graig "Selection of Best sorting Algorithm"

[3] Christian Bunse, Hagen H¨opfner, Essam Mansour, Suman Roychoudhury "Exploring the Energy Consumption of Data Sorting Algorithms in Embedded And Mobile Environments"

[4] Demuth, H. Electronic Data Sorting. PhD thesis, Stanford University, 1956.

[5] Huang, B. C.; Langston, M. A. (December 1992). "Fast Stable Merging and Sorting in Constant Extra Space". Compute. J. 35(6): 643–

[6] http://www.algolist.net/Algorithms/Sorting/Selection_sort

[7] Kagel, Art (November 1985). "Unshuffled, Not Quite a Sort".

[8] Franceschini, G. (June 2007). "Sorting Stably, in Place, with O(n log n) Comparisons and O(n) Moves". Theory of Computing Systems 40 (4): 327–353. doi:10.1007/s00224-006-1311-1.