**Analysis of Different Sorting algorithms**

**A Comparative Study**

**Rajasthan institute of engineering technology**

**Jaipur,Rajasthan**

**Department of Computer Science and Engineering**

**Anil puri goswami1 , Ankit malpani2 , Vicky kumar3 , Dr. Saroj Hiranwal(Mentor)4**

**1Student of Department of Computer Science and Engineering , Rajasthan institute of engineering technology,jaipur 302026,India**

Email:-[puria8387@gmail.com](mailto:najma.cse@gmail.com)

**2 Student of Department of Computer Science and Engineering , Rajasthan institute of engineering technology,jaipur 302026,India**

Email:-ankitmalpani1975@gmail.com

**3 Student of Department of Computer Science and Engineering , Rajasthan institute of engineering technology,jaipur 302026,India**

Email:-imvickykumar999@gmail.com

**4 Professor of Department of Computer Science and Engineering , Rajasthan institute of engineering technology,jaipur 302026,India**

Email:-principal@rietjaipur.ac.in

***Abstract— There are many popular problems in different practical fields of computer sciences, database applications, Networks and Artificial intelligence. One of these basic operations and problems is sorting algorithm; the sorting problem has attracted a great deal of research. A lot of sorting algorithms has been developed to enhance the performance in terms of computational complexity. Sorting becomes important whenever you can define some order relationship for a type and want to extract data based on that. So, if you have bunch of records of employees with lots of information, you may want to extract a list based on age or date of joining or last vacation leave taken, etc. For large amounts of data, instead of performing a sort every time a sorted list is needed, it is better to invest in some proper data structure, create indexes on each ordered property and store it; and then the sorted list boils down to some traversal.***

Keywords: Sort, Grouping Comparison Sort, Quick Sort, Merge Sort, Time Complexity.

1. **INTRODUCTION**

In computer science, a sorting algorithm is an algorithm that puts elements of a list into an order. The most frequently used orders are numerical order and lexicographical order, and either ascending or descending. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists. Sorting is also often useful for canonicalizing data and for producing human-readable output.

Information growth rapidly in our world leads to increase developing sort algorithms. Developing sort algorithms through improved performance and decreasing complexity, it has attracted a great deal of research; because any effect of sorting algorithm enhancement of the current algorithms or product new algorithms that reflects to optimize other algorithms. Large number of algorithms developed to improve sorting like merge sort, bubble sort, insertion sort, quick sort ,selection sort and others, each of them has a different mechanism to reorder elements which increase the performance and efficiency of the practical applications and reduce time complexity of each one. When comparing between various sorting algorithms, there are several factors that must be taken in consideration; first of them is the time complexity, the time complexity of an algorithm determined the amount of time that can be taken by an algorithm to run [3][7][27]. This factor different from sorting algorithm to another according to the size of data that we want to reorder, some sorting algorithm inefficient and too slow. The time complexity of an algorithm is generally written in form big O(n) notation, where the O represents the complexity of the algorithm and a value n represent the number of elementary operations performed by the algorithm [8].The second factor is the stability[26], means; algorithm keeps elements with equal values in the same relative order in the output as they were in the input. [2][3][9]. Some sorting algorithms are stable by its nature such as insertion sort, merge sort, bubble sort, while some sorting algorithms are not, such as quick sort, any given sorting algorithm which is not stable can be modified to be stable [3]. The third factor is memory space, algorithm that used recursive techniques need more copies of sorting data that affect to memory space [3][9].Many previous researches have been suggested to enhance the sorting algorithm to maintain memory and improve efficiency. Most of these algorithms are used comparative operation between the oldest algorithm and the newest one to prove that.

1. **PERFORMANCE IN AVERAGE CASE BETWEEN SORTING AlGORITHMS**

The following studies are previous study on the same research which make a comparative between different type of sorting algorithms: (Pooja Adhikari,2007) The performance of any computation depends upon the performance of sorting algorithms.Like all complicated problems, there are many solutions that can achieve the same results. This paper choose two of the sorting algorithms among them selection sort and shell sort and compares the various performance factor among them. (Davide Pasetto Albert Akhriev,2011) In this paper we provide a qualitative and quantitative analysis of the performance of parallel sorting algorithms on modern multi-core hardware. We consider several general-purpose methods,which are widely regarded among the best algorithms available, with particular interest in sorting of database records and very large arrays (several gigabytes and more), whose size far exceed L2/L3 cache. (ADITYA DEV MISHRA & DEEPAK GARG,2008)Many different sorting algorithms have been developed and improved to make sorting fast. As a measure of performance mainly the average number of operations or the average execution times of these algorithms have been investigated and compared. There is no one sorting method that is best for every situation. Some of the factors to be considered in choosing a sorting algorithm include the size of the list to be sorted, the programming effort, the number of words of main memory available, the size of disk or tape units, the extent to which the list is already ordered, and the distribution of values This paper implemented of Selection sort, Quick sort, Insertion sort , Merge sort ,Bubble sort and GCS algorithms using C++ programming language, and measure the execution time of all programs with the same input data using the same computer. The built-in function (clock ()) in C++ is used to get the elapsed time of the implementing algorithms, execution time of a program is measured in milliseconds [6].The performances of GCS algorithm and a set of conventional sort algorithms are comparatively tested under average cases by using random test data from size 10000 to 30000. The result obtained is given in Table 1 to Table 6 for each Algorithm and the curves are shown in figure 1.

**Selection sort**

*Selection sort* is an in-place comparison sort. It has O(*n*2) complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

The algorithm finds the minimum value, swaps it with the value in the first position, and repeats these steps for the remainder of the list.[20] It does no more than *n* swaps, and thus is useful where swapping is very expensive.

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 2227 |
| 20000 | 5058 |
| 30000 | 8254 |

**Table 1:-** Running time for Selection sort

**Insertion sort**

*Insertion sort* is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists, and is often used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list similar to how we put money in our wallet.[19] In arrays, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by one. Shellsort (see below) is a variant of insertion sort that is more efficient for larger lists.

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 1605 |
| 20000 | 3678 |
| 30000 | 6135 |

**Table 2 :-** Running time for insertion sort

**Merge sort**

*Merge sort* takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements (i.e., 1 with 2, then 3 with 4...) 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.[21] Of the algorithms described here, this is the first that scales well to very large lists, because its worst-case running time is O(*n* log *n*). It is also easily applied to lists, not only arrays, as it only requires sequential access, not random access. However, it has additional O(*n*) space complexity, and involves a large number of copies in simple implementations.

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 728 |
| 20000 | 1509 |
| 30000 | 2271 |

**Table 3 :-** Running time for merge sort

**Quick sort**

In this sort an element called pivot is identified and that element is fixed in its place by moving all the elements less than that to its left and all the elements greater than that to its right. Since it partitions the element sequence into left, pivot and right it is referred as a sorting by partitioning. It's an O (n log n) Time complexity in average case[21][22]. In Table 4 the execution time and number of elements as follow:

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 489 |
| 20000 | 1084 |
| 30000 | 1648 |

**Table 4 :-**Running time for quick sort

**Bubble sort**

Bubble sort is a simple sorting algorithm that works by repeatedly; it's comparing each pair of adjacent items and swapping them if they are in the wrong order. This passing procedure is repeated until no swaps are required, indicating that the list is sorted [13][23]. It has a O (n2) Time complexity means that its efficiency decreases dramatically on lists of more than a small number of elements [12][24]. In Table 4 the execution time and number of elements as follow:

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 1133 |
| 20000 | 3103 |
| 30000 | 5730 |

**Table 5 :-** Running time for Bubble sort

**Grouping Comparison sort**

In this sort we divide the list of elements into groups; each group contains three elements that compare with the first element of next groups. Performance has been decreased by GCS algorithm, mainly if the input size more than 25000 elements that returned increasing number of comparison, the performance have been improved when size of input is less than 25000 elements. It has a time complexity O (n2) [14]. In Table 6 the execution time and number of elements as follow:

|  |  |
| --- | --- |
| Number of elements | Running time (ms) |
| 10000 | 1124 |
| 20000 | 3374 |
| 30000 | 6687 |

**Table 6 :-** Running time for comparsion sort

1. **COMPARATIVE STUDY AND DISCUSSION**

All the six sorting algorithms (Selection Sort, Insertion sort, Merge sort, Quick sort, Bubble Sort and Comparison sort) were implemented in C++ programming languages and tested for the random sequence input of length 10000, 20000, 30000, All the six sorting algorithms were executed on machine Operating System having Intel(R) Core(TM) 2 Duo CPU E8400 @ 3.00 GHz (2 CPUs) and installed memory (RAM) 2038 MB. The Plot of length of input and CPU time taken (ms) is shown in figure 1. Result shows that for small input the performance for the six techniques is all most nearest, but for the large input Quick sort is the fastest and the selection sort the slowest. the grouping comparison sort for small input (10000) is the third sort and in the large input (30000) is the fifth sort in order between the six sorting algorithms.

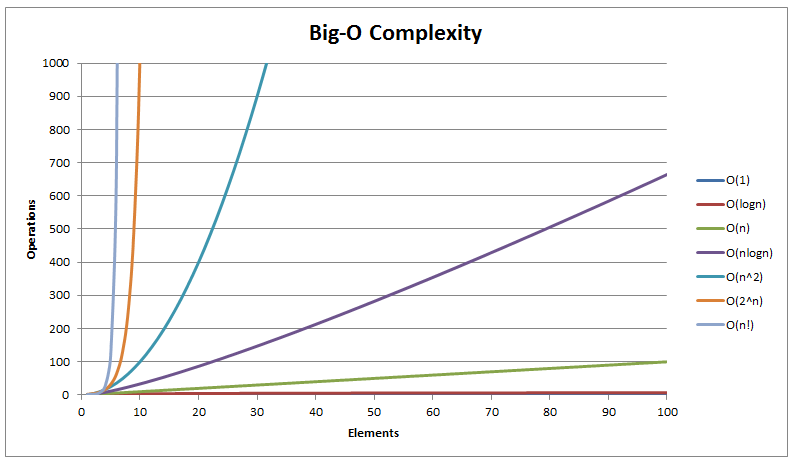
**Graph 1 :-** Running time graph of different sorting algorithms

* 1. **Complexity Comparison between Typical sorting algorithms**

The comparison of complexity between GCS and conventional sort algorithms are listed in table 7[5]. Table 6 determines the time complexity of new algorithm is equivalent to some conventional sort algorithms[25][28]. GCS gave an additional method to manipulate information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Time complexity | | | Space complexity |
| Name | Best case | Average case | Worst case | Worst case |
| Bubble sort | O(n) | O(n^2) | O(n^2) | O(1) |
| Selection sort | O(n^2) | O(n^2) | O(n^2) | O(1) |
| Insertion sort | O(n) | O(n^2) | O(n^2) | O(1) |
| Merge sort | O(nlog n) | O(nlog n) | O(nlog n) | O(n) |
| Quick sort | O(nlog n) | O(nlog n) | O(n^2) | O(log n) |
| Heap sort | O(nlog n) | O(nlog n) | O(nlog n) | O(n) |
| Bucket sort | O(n+k) | O(n+k) | O(n^2) | O(n) |
| Radix sort | O(nk) | O(nk) | O(nk) | O(n+k) |
| Tim sort | O(n) | O(nlog n) | O(nlog n) | O(n) |
| Shell sort | O(n) | O((nlog(n))^2) | O((nlog(n))^2) | O(1) |

**Table 7 :-** Sorting algorithms complexity



**Graph 2 :-**Big-O complexity graph

1. **CONCLUSION AND FUTURE WORK**

This paper discuss a comparison between the new suggested algorithm (GCS) and selection sort, Insertion sort, merge sort, quick sort and bubble sort. It analysis the performance of these algorithms for the same number of elements (10000, 20000, 30000). For small input the performance for the six techniques is all most nearest, but for the large input Quick sort is the fastest and the selection sort the slowest. Comparison sort in average and worst case have the same time complexity with selection, Insertion and bubble sort This research is initial step for future work; in the future we will improve our algorithms Grouping Comparison Sort algorithms (GCS) to optimize software’s in searching method and retrieve data.

1. **REFERENCES**
2. P.Adhikari, Review on Sorting Algorithms, "A comparative study on two sorting algorithm", Mississppi state university, 2007.
3. T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.15-17.
4. M. Goodrich and R. Tamassia, Data Structures and Algorithms in Java,John wiley & sons 4th edition, 2010,pp.241-243.
5. R. Sedgewick and K. Wayne, Algorithms,Pearson Education, 4th Edition, 2011,pp.248-249.
6. <http://en.wikipedia.org/wiki/Insertion_sort>
7. <http://en.wikipedia.org/wiki/Bubble_sort>
8. <https://en.wikipedia.org/wiki/Merge_sort>
9. <http://www.sorting-algorithms.com>
10. <https://afteracademy.com/blog/comparison-of-sorting-algorithms>

[10] <https://www.hackerearth.com/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/>

[11] <https://python-textbok.readthedocs.io/en/1.0/Sorting_and_Searching_Algorithms.html>

[12] International Journal of Computer Science and Security (IJCSS), Volume (7) : Issue (3) : 2013

[13] D.Garg,” Selection O. Best Sorting Algorithm”, International Journal of Intelligent Information Processing 2(2),pp.363-368.

[14] I. trini, k. kharabsheh, A. trini, (2013,may)."Grouping Comparison Sort", Australian Journal of Basic and Applied Sciences.pp221-228.

[15] R. Sedgewick, Algorithms in C++, Addison–Wesley Longman,1998,pp 273–274.

[16] A. Levitin, Introduction to the Design & Analysis of Algorithms, Addison–Wesley Longman, 2007, pp 98–100.

[17] <http://corewar.co.uk/assembly/insertion.htm>

[18] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.15-21

[19] Katajainen, Jyrki; Pasanen, Tomi; Teuhola, Jukka (1996,MAR). "Practical in-place mergesort". Nordic Journal of Computing. (3). pp. 27–40.

[20] Kronrod, M. A. (1969). "Optimal ordering algorithm without operational field". Soviet Mathematics - Doklady (10). pp. 744.

[21] T. Cormen, C.Leiserson, R. Rivest and C.Stein, Introduction To Algorithms, McGraw-Hill, Third Edition, 2009,pp.145-164.

[22] A. LaMarca and R. E. Ladner.(1997), "The Influence of Caches on the Performance of Sorting." Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms,. pp. 370–379.

[23] D.Knuth, The Art of Computer Programming,Addison-Wesley,Third Edition 1997,pp. 106– 110 .