# Review on Sorting Algorithms - A Comparative Study

**Muhammad Shahzad, Muhammad Shakeel, Um-A-Kalsoom, Atiq Ur Rehman, M Umair Shoukat**

***Abstract:** sorting of elements is an important task in computation that is used frequently in different processes. For accomplish task in reasonable amount of time efficient algorithm is needed. Different types of sorting algorithms have been devised for the purpose. Which is the best suited sorting algorithm can only be decided by comparing the available algorithms in different aspects. In this paper a comparison is made for different sorting algorithms used in computation..*

***Index Terms:** Best Sorting Algorithm, Bubble Sort Algorithms, Quick Sort Algorithms, Sorting Algorithms, Efficient Sorting.*

## I. INTRODUCTION

Sorting is sequence of data that is an important pillar in computer science. In sorting process data is organized in an order. Different algorithms are used for this purpose. Sorting is of two types one is internal sorting and second is external sorting. Internal sorting is performed when small amount of data that can be hold in the memory is sorted. For sorting the large amount of data, a part of whose exists in some external storage device during sorting process another type of sorting called external sorting is used. The most used sorting orders are numerical and lexicographical orders. In this paper an overview of both internal and external sorting algorithms is discussed.

## II. EXTERNAL SORTING ALGORITHUMS

External sorting algorithms handle the massive data in sorting process successfully. It is not possible to store all data into the main memory of computer and a part of data must reside on some external storage device. The strategy used for such sorting is hybrid sort-merge. In this strategy small portions of data are read from external storage device according to the main memory size, sorted and saved in temporary file and at last these all sorted files are merged together for making a single file containing sorted data. Following sorting algorithms are mostly used for external sorting.

**Revised Version Manuscript Received on December 11, 2017.**

**Muhammad Shahzad**, School of Electrical and Electronics Engineering, North China Electric Power University, Changping District, Beijing, China E-mail: shahzadpansota@hotmail.com

**Muhammad Shakeel**, Department of Electrical Engineering, The University of Lahore- Islamabad Campus, Islamabad, Pakistan, E-mail: engnr.shakeel@gamil.com

**Um-e-Kalsoom**, Department of Computer Science, Virtual University, Islamabad, Pakistan, E-mail: innocentmishi786@gmail.com

**Atiq Ur Rehman**, School of Electrical and Electronics Engineering, North China Electric Power University, Changping District, Beijing, China. E-mail: atiq_marwat@yahoo.com

**M Umair Shoukat**, Department of Electrical Engineering, Government College University Faisalabad Sahiwal Campus, Pakistan, E-mail: umairshoukat@yahoo.com

### A. External Merge Sort

External merge sort is used to sort huge amount of data. It uses chunks of data according to the size of main memory. When every chunk is sorted successfully it merges these chunks into a large file that is now fully sorted. It usually includes two pass sort, firstly a sort pass and afterwards a merge pass. In regular merge sort there are total log n merge passes but in external merge sort it is avoided for the reason that in every merge pass every data value has to be read from and write on the disk and in case of external merge sort data resides on an external storage device that are very slow rather than main memory.

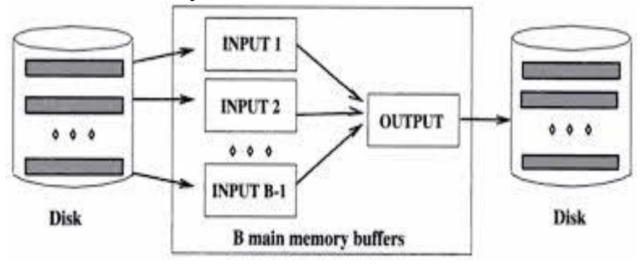


**Figure 1 : I/O and Data Storage Overview**

Sorting large amount of data while having limited main memory requires more than one passes because with small memory and large amount of data a single merge pass cannot produce efficient results. Performing efficient external sort needs O(n log n) time. Every exponential increase in data amount causes linear increase in time used for sorting. If memory is increased then a single merge pass can be efficient. In other words increase in memory decreases the number of passes in sorting process.

The second method to perform external sort efficiently is the use of parallelism. In this method multiple disk drives are used and multiple sorting threads run at the same time increasing the speed of sorting process. Several machines are linked in a speedy network to perform sorting on huge data sets in parallel.

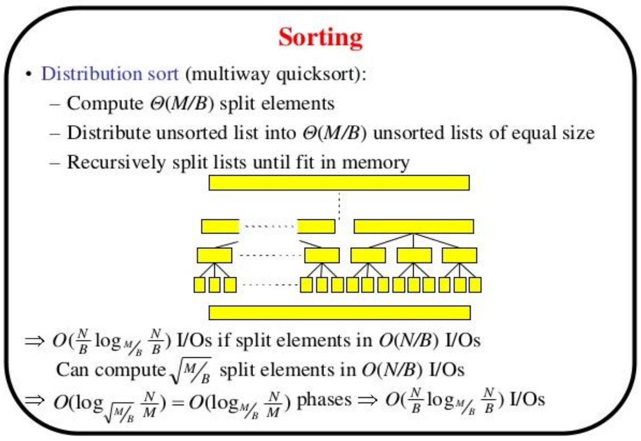
Another technique for enhancing the speed of sorting process is increasing the hardware speed. Use of a large RAM reduces the number of disk seeks and reduces the number of passes. Faster external storage devices such as solid state drives also reduce the number of passes during sorting process. Additional issues that can affect the sorting speed are the CPU performance, number of cores, RAM access latency, disk speed, seek time and I/O bandwidth. In this regard cost efficiency and absolute speed can be critical specifically in cluster environment.

*Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.*

Use of efficient and faster software for sorting also enhances the efficiency of the sorting process.

### B. Distribution Sort

The other major category of external sort is distribution sort. It has further types that fit for different types of data and situations. These include bucket sort, linear probing sort, shuffle sort, merge sort, radix sort, strand sort, distributive partitioning sort etc.



**Sorting**

- Distribution sort (multiway quicksort):
  - Compute $\Theta(M/B)$ split elements
  - Distribute unsorted list into $\Theta(M/B)$ unsorted lists of equal size
  - Recursively split lists until fit in memory

$\Rightarrow O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os if split elements in $O(N/B)$ I/Os

Can compute $\sqrt{M/B}$ split elements in $O(N/B)$ I/Os

$\Rightarrow O(\log_{\sqrt{M/B}}\frac{N}{M}) = O(\log_{M/B}\frac{N}{M})$ phases $\Rightarrow O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os

**Figure 2 : Distribution Sorting Method**

The main technique which is used in distribution sort is the division of data into small chunks using some intermediary structure and then combines app portions in one large output file. Any algorithm that requires sorting of data using comparison of keys at least requisites O(n log n) time for sorting.

## III. INTERNAL SORTING ALGORITHMS

When we have small amount of data that can be fit into the available memory the internal sorting algorithms are the best suited algorithms. These are efficient and quicker option because memory or RAM is faster than any secondary storage device. These are the more efficient in sense that their working is swift and less time consuming. Efficient algorithms are mostly based on average complexity O(n log n). The common types of efficient sorts are merge sort, heap sort and quick sort. Here under is a review of efficient internal algorithms.

### A. Merge Sort

The merge sort algorithms use the sorted lists and merge them. In this process it uses swapping of data elements if it is necessary. It continues the process till the production of final sorted list. It can be easily applied to lists and arrays because it needs sequential access rather than random access. It can handle very large lists due to its worst case running time O(n log n). The O (n) additional space complexity and involvement of huge amount of copies in simple implementation made it a little inefficient.

Merge sort is very popular in practical implementations because of its use and popularity in sophisticated algorithm Timsort. It is used in Python and JAVA programming languages. It is divide and conquer type of algorithm. Initially it divides the supplied data set into sub-lists consisting of only one element as these lists are sorted ones. Then it starts comparing and merging the elements of this sorted list until a single list is left behind. This list will be ordered list of provided data.

The processing method of merge sort can be elaborated through the following image:

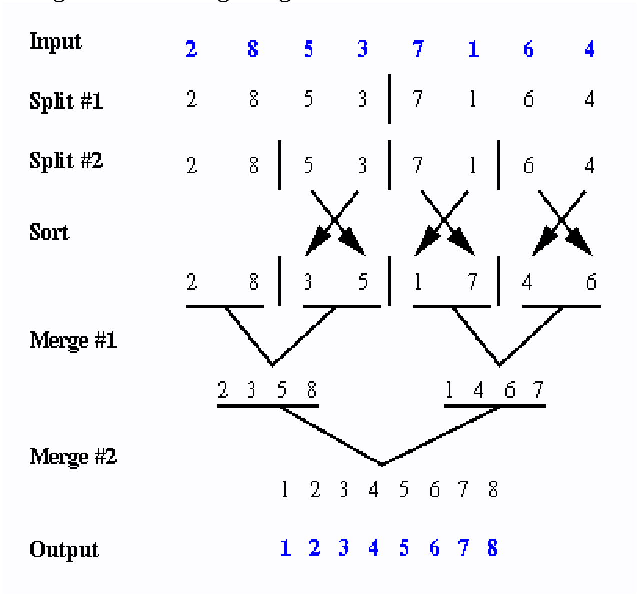
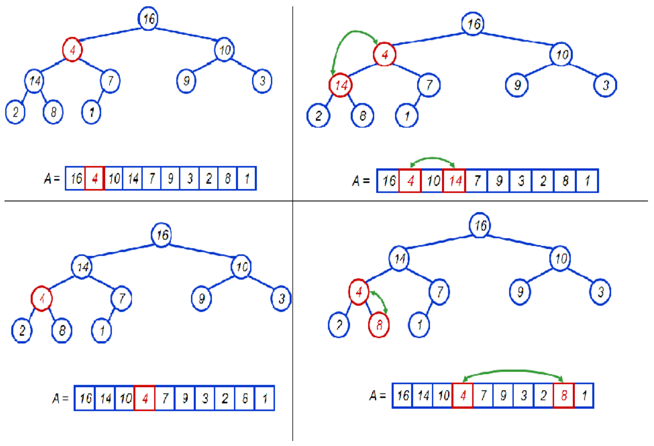


**Figure 3 : Merge Sorting Method**

### B. Heap Sort

The most efficient version of selection sort is heap sort. It firstly determines the largest or the smallest element of the list and places it at the beginning in case of largest and at the end in case of the smallest element. It continues the process for the remaining list and completes the task very efficiently and quickly. For performing the process of sorting it uses a special data structure called heap. Heap is particular type of binary tree. This algorithm makes the heap from given data that is going to be sorted. Once the heap is constructed it ensures that root node is the largest (or smallest in case of opposite sorting order) element of given data.

To create the ordered list the root node is removed and placed at the beginning of list or end of list according to sorting order. When root node is removed the next largest or smallest node moves to the root node and heap is rearranged. By continuing this process a sorted data list is gained.



A = 16 4 10 14 7 9 3 2 8 1
A = 16 4 10 14 7 9 3 2 8 1
A = 16 14 10 4 7 9 3 2 8 1
A = 16 14 10 4 7 9 3 2 8 1

**Figure 4 : Heap Sorting Method**

*Published By:*
*Blue Eyes Intelligence Engineering*
*& Sciences Publication Pvt. Ltd.*

Using the heap structure requires O(log n) time for searching the next largest or smallest data element rather than O(n) for a linear scan in simple selection sort. This lets the heap sort execution in O(n log n) time which is the worst case time complexity.

## C. Quick Sort

Quick sort also belongs to the divide and conquer category of algorithms. It depends on the partitions operation. For performing the partition operation on an array it selects an element of array that divides the array in to two parts. This element is called the *pivot*. Considering this element the centre point all elements that are smaller than pivot are arranged on one side and the larger elements on the other side of the pivot. In the next step the lesser and greater lists are sorted repeatedly.

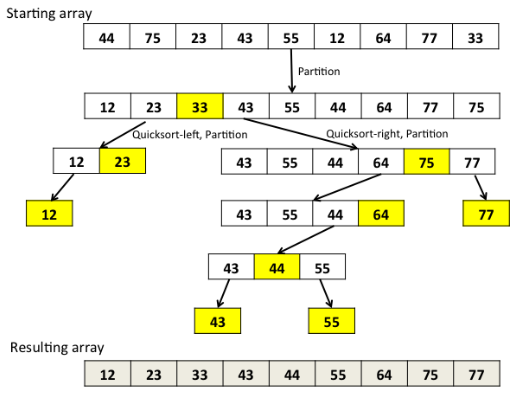
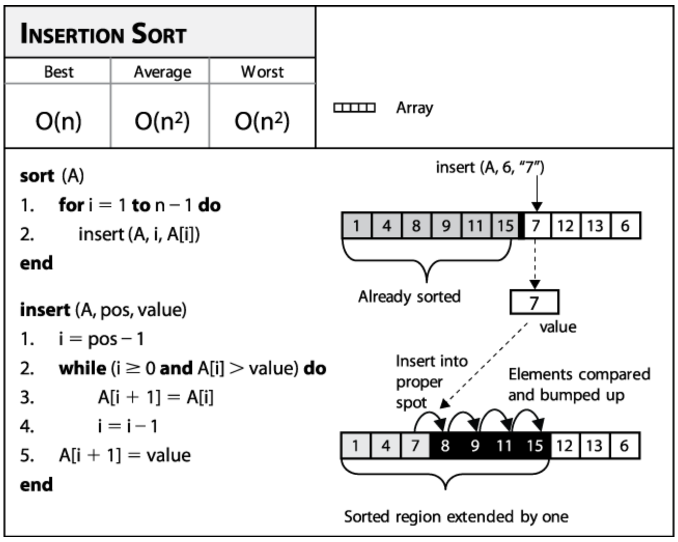


**Figure 5 : Quick Sorting Algorithm**

This is done in the average time complexity of O(n log n) with low overhead that makes it the widespread in sorting processes and it is the part of many programming libraries. The important thing that must be kept in consideration is time cost of its worst case performance that is O(n$^2$) although it is infrequent and happened only in working with sorted data and the smallest or the largest element is chosen as pivot. The key issue in using the quick sort is the selection of pivot element because poor selections can extremely slower the O(n$^2$) performance. On the other hand selection of good pivot element acquires the O(n log n) performance that is the optimal solution. To yield O (n log n) performance it is the good practice to select the median as pivot element in each step.

## D. Simple Sorts

The most popular simple sorts are insertion and selection sorts that efficient for small amount of data because these have low overhead. If we compare the both algorithms insertion sort is more efficient than the selection sort. It has fewer comparisons and efficient performance on almost sorted data. On contrary selection sort is good choice when write performance is limiting factor because selection sort requires fewer writes. Insertion sort is good choice for small and almost sorted list. It picks the elements from the input list one by one and places them into a new output list at their correct positions. Insertion is costly process as it needs moving all subsequent elements over one insertion. There are variants for algorithms that provide efficient working and for insertion sort, shell sort is a variant that is more efficient than insertion sort.



```
sort (A)
1.  for i = 1 to n − 1 do
2.      insert (A, i, A[i])
end

insert (A, pos, value)
1.  i = pos − 1
2.  while (i ≥ 0 and A[i] > value) do
3.      A[i + 1] = A[i]
4.      i = i − 1
5.  A[i + 1] = value
end
```

**Figure 6 : Simple Sorting Algorithm**

Selection sort is inefficient due to its O(n$^2$)complexity for large lists and has a worse performance rather than an equal insertion sort. Selection sort is efficient where swapping is required as it selects the minimum value and swaps it with the first value in the list. Repeating of the process generates the sorted list in n number of swaps. This quality makes selection sort the most suited algorithm where swapping is expensive.

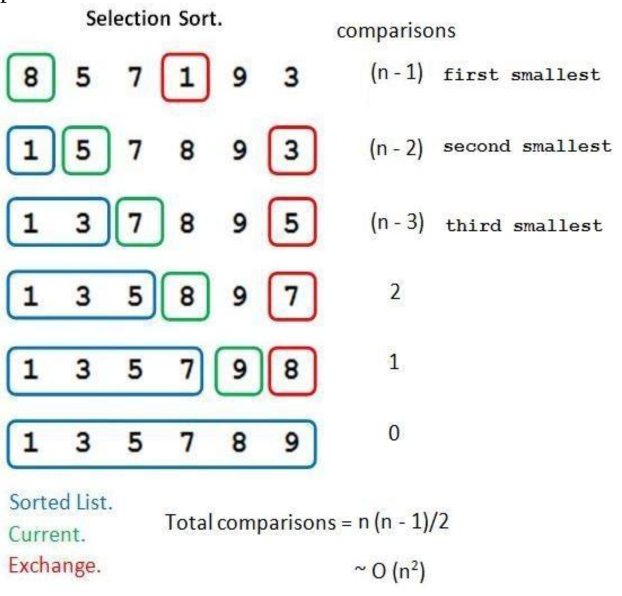


**Figure 7 : Simple Sorting Method**

## E. Bubble Sort

The bubble sort is good as an internal sorting algorithm because it sorts the list of items by comparing the two adjacent items and swapping them if they are not found in order. It performs well if all the data is fitted in the memory as a single chunk of data. Bubble sort has O(n) as the best case performance and O(n$^2$) as the worst case performance and average case performance.

*Published By:*
*Blue Eyes Intelligence Engineering*
*& Sciences Publication Pvt. Ltd.*

Bubble sort has to perform a large number comparison when there are more elements in the list and it increases as the number of items increase that are needed to be sorted. Although bubble sort is quite simple and easy to implement but it is inefficient in coding reference.
Bubble sort simply selects one item from the list and compare it with the adjacent items and put them in order. In this sequence the number of comparisons increases remarkably. Here is a simple elaboration of the bubble sort.

## IV. COMPARISON CRITERIA

Efficient sorting algorithms may vary in performance under different factors. So these can be compared or judged on the basis of these factors.
The first point that is considered for judgment is the best case, average case and the worst case performance of the algorithm. With this reference quick sort is exceptionally good in its best case for some input and dreadful for other inputs in its worst case. At the same criterion we observe that other algorithms for instance merge sort are not disturbed or altered in performance by the order of input.
The "constant term" is the other factor that is considered while evaluating the algorithms. The value of constant c will vary for different algorithms. The big-O notations are quite useful for avoiding many process details and present a big scenario. Contrary to the theory, a well applied quick sort must have a much smaller constant multiplier than a heap sort.
The space requisite for an algorithm is also a criterion for the evaluation of an algorithm. In this regard the space required to run an algorithm is considered whether it needs extra space for different data sets or not. Some algorithms under no circumstances require extra pace but some needs extra space for performing well.
Stability of an algorithm is also a valid factor for choosing the best algorithm. Stability is the quality if algorithm for keeping the sequence with identical values. Most of the sorts manage it well but for the heap sort it is not possible.
In the following table a comparison of different algorithms according to the above stated criteria. The tabular form of comparison will display a gross view of all the algorithms.

**Table 1 : Comparison of Sorting Algorithms**

| Sort | Time | | | Space | Stability |
|---|---|---|---|---|---|
| | Avg | Best | Worst | | |
| Bubble sort | O(n^2) | O(n^2) | O(n^2) | Constant | Stable |
| Selection sort | O(n^2) | O(n^2) | O(n^2) | Constant | Stable |
| Insertion sort | O(n^2) | O(n) | O(n^2) | Constant | Stable |
| Merge sort | O(n* log n) | O(n* log n) | O(n* log n) | Depends | Stable |
| Quick sort | O(n* log n) | O(n* log n) | O(n^2) | Constant | Stable |

The following points are also important while choosing a suitable algorithm

- Selection and bubble sort algorithms perform comparisons of items to move them to their final position. So they start with one item outing to the final position and continue sorting for the rest of item unless all the items acquire their final and suitable position.
- On the other hand insertion and quick sort place the sorted items in a temporary position that is nearer to final position. This job is performed iteratively till the whole list is sorted.
- Merging technique is used in merge sort. It chooses the sorted list of one item and merge the unsorted items to it one by one to create a sorted list.
- Time and space complexity of algorithm is also important and has deep effect on the performance.
- O(n) time is the least time that is needed for sorting a list of N items. It is only possible when we assume about data and do not require comparison of it's against each other iteratively. Each element passes through a comparison at least once.

## V. CONCLUSION

In this study different algorithms are observed against multiple factors. It reveals that each of the considered algorithms has advantages and disadvantages and the programmer has to be vigilant about his / her requirement of sorting algorithms. A comparison of algorithm in this paper is showing the performance of sorting algorithms along with their differences to each other.

## ACKNOWLEDGMENT

We would like to state hereby my thanks to Almighty Allah who helped us a lot and enable us for writing this paper. We would like to pay our gratitude to our parents.

## REFERENCES

1. Chudy, M. (2010). Simulation Needs Efficient Algorithms. In Modeling Simulation and Optimization-Focus on Applications. InTech.
2. Atallah, M. J. (1985). Some dynamic computational geometry problems. Computers & Mathematics with Applications, 11(12), 1171-1181.
3. Paira, S., Agarwal, A., Alam, S. S., & Chandra, S. (2015). Doubly Inserted Sort: A Partially Insertion Based Dual Scanned Sorting Algorithm. In Emerging Research in Computing, Information, Communication and Applications (pp. 11-19). Springer, New Delhi.
4. Groppe, S. (2011). External Sorting and B+-Trees. In Data Management and Query Processing in Semantic Web Databases (pp. 35-65). Springer Berlin Heidelberg.
5. Paira, S., Chandra, S., Alam, S. S., & Dey, P. S. A Review Report on Divide and Conquer Sorting Algorithm. In National Conference on Electrical, Electronics, and Computer Engineering, ISBN (pp. 978-93).
6. Paira, S., Chandra, S., Alam, S. S., & Patra, S. S. (2014). Max min sorting algorithm—a new approach of sorting. Int. J. Technol. Explor. Learn.(IJTEL), 3(2), 405-408.
7. Adamson, I. T. (2012). Data structures and algorithms: a first course. Springer Science & Business Media.
8. SurenderLakra, D. (2013). Improving the performance of selection sort using a modified double-ended selection sorting. International Journal of Application or Innovation in Engineering & Management (IJAIEM), Web Site: www. ijaiem. org Email: editor@ ijaiem. org, editorijaiem@ gmail. com, 2(5).
9. Sodhi, T. S., Kaur, S., & Kaur, S. (2013). Enhanced insertion sort algorithm. International journal of Computer applications, 64(21).