Performance Comparison between Merge and Quick Sort Algorithms in Data Structure

Irfan Ali[1], Haque Nawaz[2], Imran Khan[3], M. Ameen Chhajro[5], M. Malook Rind[6]
Department of Computer Science
Sindh Madressatul Islam University (SMIU)
Karachi, Sindh, Pakistan

Abdullah Maitlo[4]
Department of Computer Science
Shah Abdul Latif University (SALU)
Khairpur, Sindh, Pakistan

Abstract—In computer science field, one of the basic operation is sorting. Many sorting operations use intermediate steps. Sorting is the procedure of ordering list of elements in ascending or descending with the help of key value in specific order. Many sorting algorithms have been designed and are being used. This paper presents performance comparisons among the two sorting algorithms, one of them merge sort another one is quick sort and produces evaluation based on the performances relating to time and space complexity. Both algorithms are vital and are being focused for long period but the query is still, which of them to use and when. Therefore this research study carried out. Each algorithm resolves the problem of sorting of data with a unique method. This study offers a complete learning that how both of the algorithms perform operation and then distinguish them based on various constraints to come with outcome.

Keywords—*Performance; analysis of algorithm; merge sort; quick sort; complexity; time and space*

I. INTRODUCTION

An algorithm is any well- defined computational or step by step process for resolving a problem. It takes some values as input and produces some values as output; it will terminate finite number of steps. In mathematics and computer science, an algorithm usually means a logical depiction of the commands which must be performed significant activity [1]. The main factor of analysis of algorithm is to study about time and space and their relationship between the algorithms necessities and number of elements or items being executed or processed. Generally sorting is the method of reorganizing a given set of data and objects within a particular arrangement and that's why, to understand the purpose of the most valuable sorting algorithms have considered as more significant research area nowadays [1, 2] even though, there are many novel sorting algorithms being instigated and used. Many software engineers in their area of programming they are depending on the different sorting algorithms: i-e. Merge, Quick sort etc. sorting is universally globally sorting is performed and it is known as essential activity. Most important sorting application is used in daily life. For example: Phone book faster access to contacts, income tax files, contents of tables, libraries access, dictionaries, search engine [1-9].

A. Computational Complexity

Sorting algorithms, computational complexities, are based on:

- $O(n\log n)$
- $O(\log 2n)$

B. Memory Usage

Classifications of algorithms on the bases of memory

1) When data set is small; internal sorting preferred primary memory only, [2] [1]

2) External sorting uses primary and secondary memory.

II. RESEARCH METHODS AND DISCUSSION

The Algorithms implemented by using java language and discussed the outcomes.

A. Computational Method of Merge Sort

Merge sort algorithm is used DAC (Divide and Conquer) prototype; for example, it split the list of records into two smallest units after that it compare each element with adjacent list and sort the two pieces or units of data sets recursively, consequently it merges and sorted the all the elements in the list. Theoretically, a merge sort perform operation as trails to split the disorder list into n elements sub units or lists, comparing every element of a list of single element observed sorted.

MER-SORT (Data_list, k, m)

1) $k < m$
[Check base case value]

2) Then x = FLOOR $[(k + m)/2]$
[Div step]

MergeData (Data_list, k, x)

[Conquer step]

3) Merge Data(Data_list, x + 1, m)
[Conquer step.]

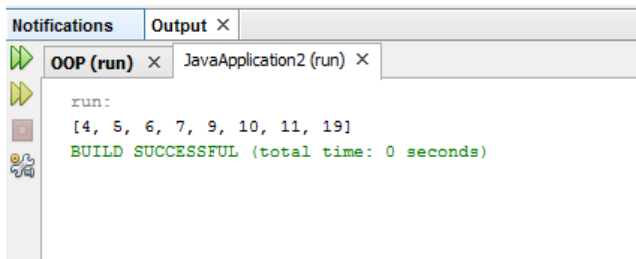*4)* Merge Data (Data_list, x, k, m)
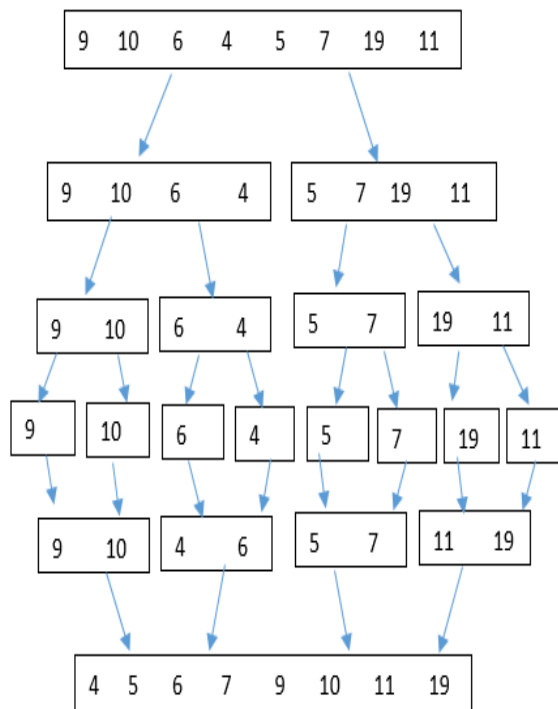[Conquer step]

Fig. 1.   Execution Output Console.

Fig. 2.   Merge Sort Pictorial Presentation.

The Fig. 1. Shows the execution output console of merge sort.

To investigate the Merge Sort function, the two different processes that we need to reflect is to form its implementation. Fig.2 shows the list is divided into two parts. The first part of list is the length defined by, half log (n) times wherever n represents the no: of elements in the list. Another part of list is merging of the list. Where every element from the list will be computed, and positioned on the output list that is sorted. Hence, the merging procedure outcomes in a list require n procedures and size n. The analysis gives the outcome, split of log n, and each costs n for a whole of n (log n) processes.

- Best Case Complexity O(nlogn)

- Average Case Complexity O(nlogn)

- Worst Case Complexity O(nlogn)

Advantages:

- Time complexity O(nlogn).

- Used both internal and external sorting

- Stable sort algorithm

Disadvantages:

- As a minimum double the memory necessities of the further sorts since it is recursive.

- Required high space complexity

B. Computational Method of Quick Sort

Quick sort algorithm is used DAC (Divide and Conquer) prototype. Quick sort initial splits a list with in two small sub units: one having low item and another having the high items. Quick sort perform operation to sort sub lists recursively. The implementation activities are: to pick element from the list is called a pivot, from the data list.

Recursively type the sub-list of smaller components and the sub-list of larger components.

Quick sort could be extremely economical algorithmic rule and is predicated on dividing of an array of knowledge with in smaller arrays. An oversized array is divided into 2 arrays one among that hold prices smaller than the desired value, say pivot, supported that the divider is created and another array holds prices larger than the pivot value.

Quick type divides an array and so calls itself recursively doubly to type the 2 ensuing sub arrays. This algorithmic program is sort of economical for large-sized information sets as its average and worst case complexness square measure of O(nlogn), wherever n is that the variety of things.

In fact, it isn't essential to separate the list accurately; even though every pivot devides the weather, 99% one side and 1% on another side, the decision depth remains restricted to, therefore the total period of time remains O(n log n) [3].

Algorithm:

1. quick_sort(DATA, start, end):
2. if (start < end)
    a. set point = partition_list(DATA, start, end)
    b. quick_sort(DATA, start, point - 1)
    c. quick_sort (DATA, point + 1, end)
    [End if in step 2]
3. Exit
Function partition_list(Data_list, start_list, end_list)
 1. Set pivotIndx = select Pivot(Data_list, start_list, end_list)
2. Set pivotElement = Data_list[pivotIndx]
3. Set Exchange Data_list [pivotIndx] and Data_list[end_list]
4. Set storeArrIndex = start_list
5. for k = start_list to end_list−1 a)
    if Data_list[k] <= pivotElement
    exchange Data_list[k] and Data_list[storeArrIndex]
    storeArrIndex = storeArrIndex + 1
        [End of if statement in step a]
        [End of for statement in step 5]
6. Exchange Data_list[storeArrIndex] and DATA[end_list]
7. return storeArrIndex

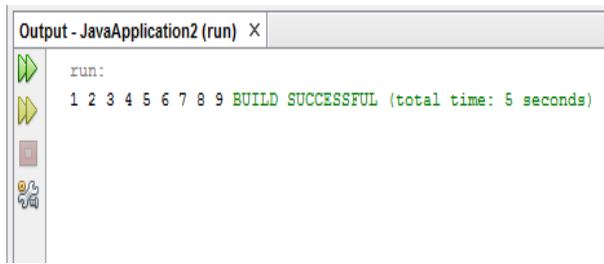Fig. 3. Shows the output execution console of quick sort.
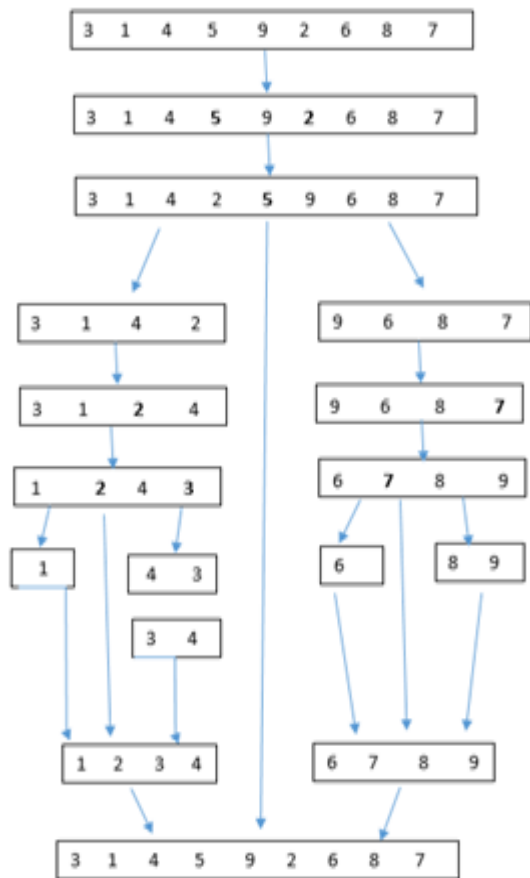
Fig. 3.   Execution Output Console.

Fig. 4.   Quick Sort Pictorial Presentation.

It has been analyzed by using a list has n length of elements, although the partition elements take place at mid of the list shows in fig. 4. More over the complexity will be the log(n) partitions carried out. However, to find themid-pointeach element of the list has been analyzed instead of pivot value. The result has been observed log (n). On other hand in worse case situation did not occur in the middle. It has been observed the mid-point in this case towards the left or right of the mid. That's why it has been observed irregular partition within this case list of n elements sorting split with 0 and n-1 element. After this by using:

n-1 split into 0 size and n-2 size and consequently. As it has been analyzed the recursion requires as $O(n^2)$ sort.

According to above algorithm the base case will be O(nlogn), average case O(nlogn) and worst case $O(n^2)$

Advantages

- No extra memory is required

- Fastest algorithms on average.

- When we have string and integer type of data compassion relatively cheap.

- The list has been traversed successively, and it has been produce very decent locality of location and the performance of cache for arrays

Disadvantages

- In case of recursive function, the average case space complexity is little bit costly, especially when we have large set of data

- Worst case complexity is $O(n^2)$

- Un stable sort algorithm

Comparison of merge and quick sort algorithms on bases on (Base, average and worst case) as showing in the subsequent table 1 [1, 4].

TABLE I.   COMPLEXITY COMPARISON

Complexity	*Merge*	*Quick*
Best Case	O(nlogn)	O(nlogn)
Average Case	O(nlogn)	O(nlogn)
Worst Case	O(nlogn)	$O(n^2)$

III. CONCLUSION

From the above analysis, it has concluded that both the quick and merge sort uses DAC (Divide and Conquer) strategy. Both having the average time complexity of O(nlogn). However, both algorithms are quite different. The merge sort is usually required while sorting a too large set to hold or handle in internal memory. It divides the set into a few subsets of one element and then repeatedly merges the subsets into increasingly larger subsets with the elements sorted correctly until one set is left. Usually this method means that the sorting ultimately deals with only portions of the complete set.

In many cases, implementing the quick sort often yields a faster sort other than O(nlogn) sorting algorithms but the worst-case time is $O(n^2)$. The quick Sort operates by selecting a single element from the set and labeling it the pivot. The set is then reordering to ensure that all elements of lesser value than the pivot appear earlier than it and the entire elements of greater value come after it. This operation is recursively applied to the subsets on both sides of pivot until the entire set has been deemed and sorted.

REFERENCES

[1] M. P.K. Chhatwani and M. J. S. Somani ,"Comparative Analysis & Performance of Different Sorting Algorithm in Data Structure" International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, no.11, pp.500-507, 2013.

[2] M. Gul, N. Amin and M. Suliman "An Analytical Comparison of Different Sorting Algorithms in Data Structure" International Journal of Advanced Research in Computer Science and Software Engineerin, Vol. 5, no.5, pp. 1289-1298, 2015.

[3] M. N. Kumar and M. R. Singh "Performance Comparison of Sorting Algorithms On The Basis Of Complexity" International Journal of Computer Science and Information Technology Research Vol. 2, no. 3, pp. 394-398, 2014.

[4] G. Kocher1 , N. Agrawal "Analysis and Review of Sorting Algorithms"International Journal of Scientific Engineering and Research (IJSER), Vol. 2, no. 3, pp. 81-84, 2014.

[5] K.K. Pandey, R.K. Bunkar, and K.K. Raghuvanshi. "A Comparative Study of Different Types of comparison Based Sorting Algorithms in Data Structure" International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, no. 2, pp. 304-309, 2014.

[6] Raju and D. Narasimha. "An efficient new approach mean based sorting." In Electrical Computer and Electronics (UPCON), UP Section Conference on, IEEE, pp. 1-5., 2015.

[7] Y. You, P. Yu, and Y. Gan. "Experimental study on the five sort algorithms" In Mechanic Automation and Control Engineering (MACE), Second International Conference, IEEE, pp. 1314-1317. 2011.

[8] D. Rajagopal, and K. Thilakavalli. "Comparison of different sorting algorithms in Data Structure based upon the Time Complexity" International Journal of Technology, Vol.6, no. 1, pp. 23-30, 2016.

[9] Z. Marszałek, "Parallelization of modified merge sort algorithm" Symmetry, Vol. 9, no. 9, pp.1-18, 2017.