Performance Comparison between Merge and Quick Sort Algorithms

Course Code: CSE 215
Course Title: Algorithm Lab

Submitted To:

Subroto Nag Pinku

Department of CSE

Daffodil International university

Submitted by:

Name Bipro Roy

ID:191-15-12976

Section:O-13

Daffodil International university

Performance Comparison between Merge and Quick Sort Algorithms

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. 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]. Computational Complexity Sorting algorithms, computational complexities, are based on: • O (nlogn) • O (log2n) 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.

RESEARCH METHODS AND DISCUSSION:

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]

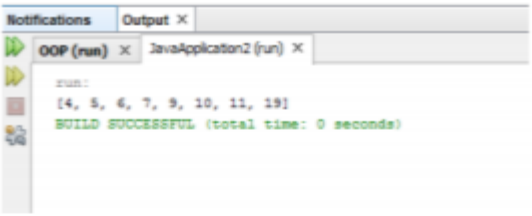

Fig. 1. Execution Output Console.

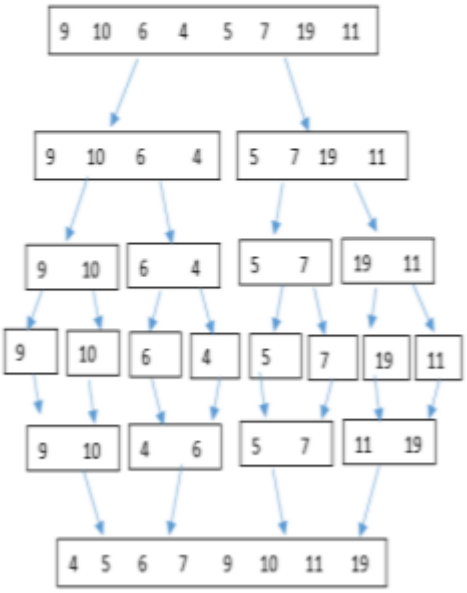

Fig. 2. Merge Sort Pictorial Presentation.

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

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

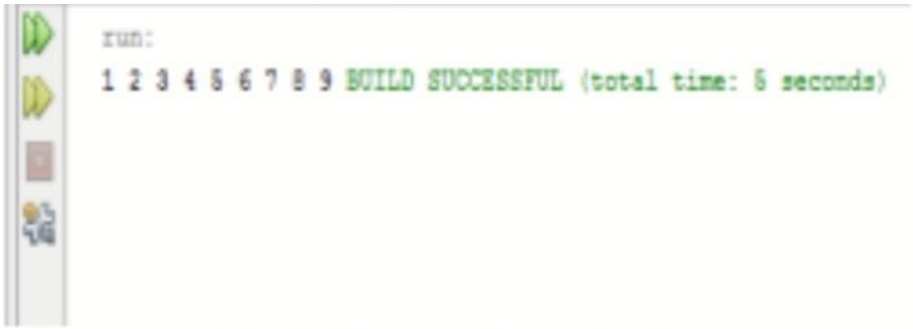

Fig. 3. Execution Output Console.

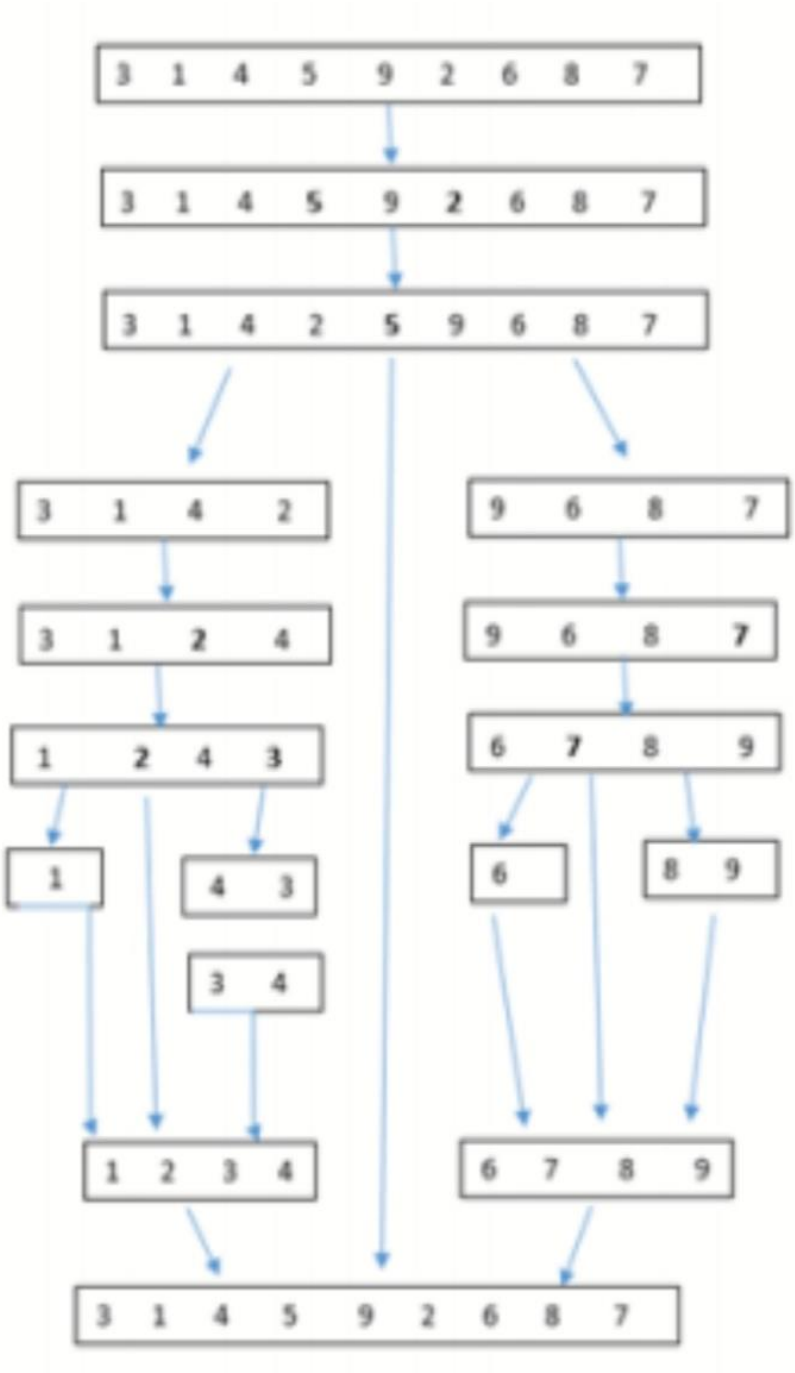

Fig. 4. Quick Sort Pictorial Presentation.

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 themidpointeach 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(n2) sort. According to above algorithm the base case will be O(nlogn), average case O(nlogn) and worst case O(n2)
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(n2)
- 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].

COMPLEXITY	Merge	Quick
1.Best Case	O(nlogn)	O(nlogn)
2. Average Case	O(nlogn)	O(nlogn)
3. Worst Case	O(nlogn)	O(n2)

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(n2). 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.