A DIVIDE AND CONQUER STRATEGY FOR IMPROVING THE EFFICIENCY AND PROBABILITY SUCCESS IN SORTING

S. Muthusundari
Research Scholar
Sathyabama University
Chennai, India
nellailath@yahoo.co.in

Dr. S. Santhosh Baboo
Reader, Dept of Computer Science
D .G. Vaishnav College
Chennai, India
santhos2001@sify.com

Abstract - Any number of practical applications in computing requires things to be in order. 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. One sort algorithm can do sorting of data faster than another. A lot of sorting algorithms has been developed to enhance the performance in terms of computational complexity, memory and other factors. In this paper, the basic idea is to prove the divide and conquer strategy is an efficient technique and probability success for better performance of sorting. In this paper, we do the comparative study the mathematical results of various sorting were verified experimentally on randomly generated unsorted numbers. To have some experimented data to sustain this we compare four different sorting methods were chosen and code was executed and execution time was noted to verify and analyze the performance. At last the divide and conquer strategy performance was found better than other sorting methods.

Keywords: Sorting, Divide and conquer, efficient and probability success.

I. INTRODUCTION

In computer science and mathematics, a sorting algorithm is an algorithm der. The most use orders are numerical order and lexicographical order [1]. Efficient sorting is important to optimizing the use of other algorithms (such as search and merge algorithms) that require sorted list to work correctly; is also often useful for producing human readable output. Sorting algorithms are classified by several other criteria such as computational complexity (worst, average and best number of comparisons for several typical test cases) in terms of the size of the list, stability (memory usage and use of other computer resources), the difference between and average behavior, behaviors' on practically important data sets (completely sorted, inversely sorted and almost sorted).

Sorting is the rearrangement of things in a list into their correct lexicographic order. A number of sorting algorithms have been developed like include heap sort , merge sort, quick sort, selection sort all of which are comparison based sort .There is another class of sorting algorithms which are non comparison based sort. This paper gives the brief introduction about sorting algorithms [2] where it discuss about the class of sorting algorithms and their running times. In this paper, we compare the different sorting methods and its algorithmic approach to be discussed. And its efficiency and their computational performance is analyzed and then we conclude the approach of divide and conquer strategy is better performance than any other technique for the success rate on sorting. The well known existing sorting techniques such as bubble sort, quick sort, insertion sort and Merge sort is considered.

Sorting is one of the basic operations that are performed by many computers since handling the data in a certain order is more efficient than handling the randomized data [3]. There are different kinds of sorting algorithms .Some of the common sorting algorithms are described below.

A. BUBBLE SORT

Bubble sort [5] is the pioneer among all the sorting techniques, the simplest and popular one in fact. It takes a list compare the two items at a time and swaps them if they are not sorted, repeat the steps until the one of the compared item reaches its exact location in that list, these whole steps continue till no swapping is required more and this is the time when sorting is completed. It compares the first two elements and if the first is greater than second, it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass. While simple, this algorithm is highly inefficient and is rarely used except in education. Average and worst case complexity of bubble sort is O (n^2).

B. INSERTION SORT

Insertion sort is a simple sorting algorithm, a comparison sort in which the sorted array is built one entry at a time. This sort is relatively efficient for small lists and mostly sorted list, and often is used as part of more sophisticated algorithms [4]. It works by taking elements from the list one by one and inserting them in their correct positions in to new sorted lists.

C. QUICK SORT

Quick sort is another well known sorting algorithm and base on divide and conquer paradigm. Its worst case running time is O(n^2) having a list of n items. In spite of slow worst case running time, quick sort is often the best practical choice for sorting the lists because it is extremely efficient on the average running time i.e. O (nlogn) [6].

Quick sort is divide and conquer *algorithm which relies on a partition operation;* to partition an array, we choose an element, called a pivot, move all smaller elements before the pivot and move all greater elements after it. This can be done efficiently in linear time and in-place. The pivot then it works in O (n log n).

D. MERGE SORT

Merge sort is divide and conquer algorithm. Merge sort [7] takes advantage of the ease of merging already sorted list into a new sorted list. It starts by comparing 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 list of four, than merges those lists of four and so on, until at last two lists are merged into the final sorted list. The worst case running time is O (n log n).

II. METHODS AND MATERIALS

In this section, we describe some of the existing sorting algorithms and their methods to sort the lists. Let us describe the four famous sorting techniques such as Bubble sort, Insertion sort, Quick sort and Merge sort.

Sorting algorithms used in computer science are often classified by:

- Computational complexity of element comparisons in terms of size of the list.
- Computational complexity of swaps
- Memory usage
- Recursion
- Stability
- General method: insertion, exchange, selection, merging and divide and conquer and partitioning.
- Generally Exchange sort includes bubble sort and cocktail sort. Selection includes selection sort, shaker sort and heap sort. Merging includes merge sort. Partitioning includes quick sort. Divide and conquer includes Merge sort and Quick sort [8].

DIVIDE AND CONQUER

In Computer science, divide and conquer (D&C) is an important algorithm design paradigm. It works by recursively breaking down a problem into two or more sub – problems of

the same type, until these become simple enough

NAME OF THE SORT	RECURRENCE RELATION	METHOD
BUBBLE SORT	O(n^2)	Exchanging
INSERTION SORT	O(N^2)	Insertion
QUICK SORT	O(N LOG N)	Partitioning, divide & conquer
MERGE SORT	O(n log n)	Divide & conquer, merging
Selection Sort	O(n^2)	selection
HEAP SORT	O(n log n)	selection
Cocktail sort	O(n^2)	Exchanging

to be solved directly. The solutions to the sub – problems are then combined to give a solution to the original problem. A divide and conquer is closely tied to t type of recurrence relation between functions of the data in question: data is "divided" into smaller partitions and the result calculated thence.

This technique is the basis of efficient algorithms for all kinds of problems, such as sorting (quick sort, merge sort) and the discrete Fourier transform (FFTs). Some of the advantages of divide and conquer are as follows:

- Solving difficult problems
- Algorithm efficiency
- Memory access
- Parallelism

There are a number of general and powerful computational strategies that are repeatedly used in computer science. It is often possible to phrase any problem in terms of these general strategies. These general strategies are Divide and Conquer, Dynamic Programming.

The most widely known and often used of these is the divide and conquer strategy.

The basic idea of divide and conquer is to divide the original problem into two or more sub-problems which can be solved by the same technique. If it is possible to split the problem further into smaller and smaller sub-problems, a stage is reached where the sub-problems are small enough to be solved without further splitting. Combining the solutions of the individuals we get the final conquering. Combining need not mean, simply the union of individual solutions.

SOME LIST OF SORTING ALGORITHM WITH ITS TECHNIQUES

II. PSEUDO CODE

Divide-and-Conquer Algorithm

Divide-and-conquer is a top-down technique for designing algorithms that consists of dividing the problem into smaller sub problems hoping that the solutions of the Sub problems are easier to find and then composing the partial solutions into the Solution of the original problem.

Little more formally, divide-and-conquer paradigm consists of following major phases:

• Breaking the problem into several sub-problems those are similar to the original

problem but smaller in size,

• Solve the sub-problem recursively (successively and independently), and then

• Combine these solutions to sub problems to create a solution to the original problem. Divide and Conquer involves four steps
1. Divide

2. Conquer [Initial Conquer occurred due to solving]

3. Combine

4. Conquer [Final Conquer].

In precise, forward journey is divide and backward journey is Conquer. A general binary divide and conquer algorithm is :

Procedure D&C (P,Q) //the data size is from p to

q
{
If size(P,Q) is small Then

Solve(P,Q)

Else

M ¬ divide(P,Q)

Combine (D&C(P,M), D&C(M+1,Q))

}

IV.COMPARATIVE ANALYSIS

In this previous section, Bubble sort, Insertion sort, Quick sort and Merge sort algorithms and their methods were described. The performance metric in all the experiments is the total execution time taken and the number of comparisons taken by the sorting operation [9].

The experiments were conducted in two categories

Category 1:

Number of Comparisons made for a random data set

Category 2:

Average computing time taken for a random data set

For the experiments integer numbers have been used which were generated randomly. To obtain results data files were used. To study the performance of the algorithms generated data sets with 50 to 500 items were used and code was executed 50 times and average execution time and number of comparisons were recorded in C language.

Table 1
NUMBER OF COMPARISONS MADE FOR RANDOM DATA SET

Sort/elements	50	100	200	300	400	500
Bubble sort	1410	5335	20300	45650	79866	126585
Insertion sort	1391	5399	20473	44449	78779	123715
Quick sort	399	990	1954	3384	5066	6256
Merge sort	500	1140	2154	3685	6543	7349

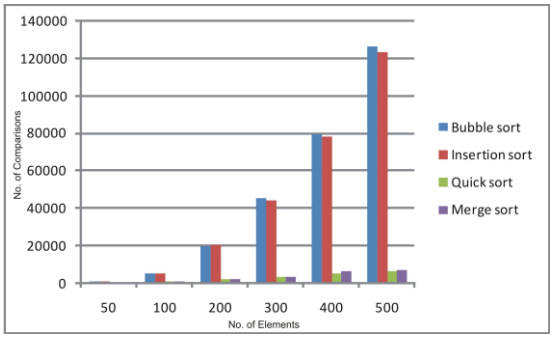
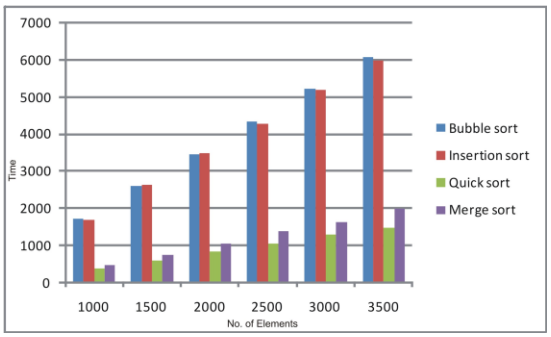


Table 2
COMPARING THE AVERAGE COMPUTING TIME FOR FOUR SORTING ALGORTIHMS

Sort/elements	1000	1500	2000	2500	3000	3500
Bubble sort	1741	2612	3484	4354	5225	6096
Insertion sort	1691	2650	3500	4300	5200	5987
Quick sort	400	600	850	1050	1300	1500
Merge sort	500	750	1050	1400	1650	2000



Results of Category 1 are shown in Table 1. It shows the number of comparisons of all the four algorithms for the no. of data items ranging from 50 to 500. It is observed that the Quick sort takes very less comparisons than other sorting algorithms. The next efficient is observed that Merge sort takes less than the other sorting algorithms. From the table 1 the greater number of comparison takes Bubble sort and the next is Insertion sort. The methods in the sorting algorithms were already described in section III. The algorithm uses the divide and conquer technique which yields efficient performance than the other methods in sorting algorithms.

Results of Category 2 are shown in Table 2. It shows the execution times of all the four algorithms for no. of elements ranging from 1000 to 3500. It is observed that Quick sort takes less time than other sorting algorithms. The next place takes merge sort for less time. The greatest time taken by Bubble sort and the next place takes Insertion Sort**.** Hence it is observed that the sorting algorithms which uses the divide and conquer technique that shows efficient and probabilistic success than any other methods in sorting.

V. CONCLUSIONS

This paper presented the comparative performance study of four sorting algorithms. To study the effectiveness of the algorithms, we have implemented all the four sorting methods. From the above graph is mentioned, we prove that the performance of Quick sort and merge sort is more efficient than the other two existing sorting algorithms. As we have described the methods used for all the four sorting algorithms in section III.

The result of this paper shows that the Quick sort and merge sort techniques are efficient one. The worst performance algorithms are considered as Bubble sort and Insertion sort. Normally the efficient of the algorithms which have already proven by using divide and conquer technique in section III described. Bubble sort uses the Exchange method and Insertion sort uses Insertion method for the algorithmic approach.

Quick sort and merge sort uses the divide and conquer method to order the list. Hence the efficient also proven in this paper. The result also proven Quick sort and merge sort are the efficient sorting methods because of using the approach Divide and conquer. Hence the efficient sorting techniques with only by obtained using divide and conquer method.

Though there are other sorting algorithms but the graph of total time and number of comparisons taken by different sorting algorithms confirms the methods used as divide and conquer technique is superiority of all the other existing similar algorithms. Hence we have also proved the efficiency and probabilistic success by using the divide and conquer technique in sorting. Sorting takes a vital role in many other applications in database management systems, file retrieval systems and networking theory wherever we have to consider all possible data sets.

Our future work comprises of presenting more sorting algorithms which includes divide and conquer techniques for decreasing its time bound and specific applications [10].

REFERENCES

[1]ThomasH.Cormen, Charles, E.Leiserson, RonaldL.Rivest, CLIfford Stein, *Introduction to Algorithms*, Second Edition, Prentice-Hall, NewDelhi, 2004

[2]Ananth Grama,Anshul Gupta,George Karypis,Vipin Kumar,*Intoduction to Parallel computing*,Second Edition,Addison-Wesley

[3]Wikipedia, *SortingAlgorithm*, http://en.wikipedia.org/wiki/Sorting_algorithm, Current: 2007-04-24

[4]MichaelLamont, *SortingAlgorithms*, http://linux.wku.edu/~lamonml/algor/sort/sort.html, Cuttent:

2007-04-24

[5]Wikipedia,Bubble*Sort*, http://en.wikipedia.org/wiki/Bubble_sort, Current: 2007-04-24

[6]Wikipedia,QuickSort, http://en.wikipedia.org/wiki/Quick_sort, Current: 2007-04-24

[7] Robert Lafore, Data Structures and Algorithms in Java, Second Addition, 2002.

[8] McCauley, P.B., Sorting methods and Apparatus, U. S. Patent 4, 809, 159, 1989

[9] Wainwright, R. 1985. A class of sorting algorithm based on Quick sort. ACM 28(4), 396-402.

[10] Cook, R. and Kim, J. 1990. Best sorting algorithms for nearly sorted list ACM, 23, 620 – 624.