Volume 4, Issue 6, June 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

An Efficient and Optimistic Binary Sort algorithm using Divide and Conquer

Love Arora
Department of Computer Science & Engineering
Guru Nanak Dev University, Regional Campus
Jalandhar, India.

Abstract - In the domain of computer science, sorting is the basic fundamental task that each and every procedure and application follow to have data in a discipline and in a convenient form. We have developed an efficient sorting algorithm called binary sort to get optimistic solutions in the minimal time and complexity of O (n log n) using divide and conquer technique. This algorithm offers opportunity to have set of optimistic solutions in one go to a variety of problems in computer science.

Keywords: Binary Sort, Sorting, Efficient Algorithm, Sorting Algorithm, Sort Data.

I. INTRODUCTION

One of the fundamental problems of computer science is ordering a list of items in least count of time and has greater efficiency. There is a plethora of solutions to this problem, known as sorting algorithms. Sorting is a fundamental task that is performed by most computers. It is used frequently in a large variety of important applications. It is about 25% of the total time that computer deals is dedicated to various sorting procedures that the operating system has to perform to achieve required successful functionality. Database applications used by schools, banks, and other institutions all make use of sorting technique to have all the data in discipline. Sorting has been universally accepted in every aspect of science and technology because of its applicability, flexibility and excellent functionality. The various sorting methods are applied in the core to have the sorted entities for getting the optimistic solution to various problems concerned with it along with least complexity, stability and the faster speed. It is important at both secondary and tertiary levels of education and research The common sorting algorithms can be divided into two classes by the complexity of their algorithms. There is a direct correlation between the complexity of an algorithm and its relative efficiency [1]. Algorithmic complexity is generally written in a form known as big-O notation, where the O represents the complexity of the algorithm and a value n represents the size of the set the algorithm is run against. Sorting refers to the arranging of numerical or alphabetical or character data in statistical order (either in increasing order or decreasing order) or in lexicographical order (alphabetical value like addressee key) [3,5-6]. Because of the importance of sorting in various applications, dozens of sorting algorithms have been developed over the decades with varying complexity.

There are several slow sorting algorithms which are simple and intuitive, such as Bubble Sort, Selection Sort and Insertion Sort have a theoretical complexity of O (n^2) that limits its usefulness to small number of elements no more than a few thousand data points. The quadratic time complexity of existing algorithms such as Bubble Sort, Selection Sort and Insertion Sort limits their performance when array size increases. And Of course, if an application needs a faster sorting algorithm, there are certainly many algorithms which are already developed till now , including Quick sort (extremely complicated), Merge sort, and Heap sort to produce lightning-fast results but still there is lack of theoretical complexity to be reduced which is essential to get efficient algorithm. These most of the algorithms have a theoretical complexity of O (n log n) in average case and have O (n^2) in worst case. Since the dawn of computing, the sorting problem has attracted a great deal of research from last decade perhaps to reduce the complexity of sorting algorithm and to increase its efficiency.

Our contribution is to introduction a new algorithm called binary sort, which is an efficient algorithm based on the Divide and Conquer [4] technique. We evaluated the theoretical and empirical complexity of Binary Sort which comes out to be O (n log n) after the careful analysis in both the average case and O (n^2) in worst case. This algorithm is easy to implement, works very well for different types of input data, and is known to use fewer resources as compared to any other sorting algorithm [7]. It has a better running time than the simplest Bubble Sort, Selection Sort and Insertion Sort algorithm and almost comparative to the Quick Sort and Heap Sort but along with more convenience. It yields around 60% performance improvement over the Bubble sort, Insertion sort and Selection Sort [10]. Similar to the most classic non adaptive sorting algorithms like Quick Sort, Heap Sort [2,9], and Merge Sort [2], Binary Sort is also non adaptive and its time complexity is O (n log n) irrespective of the number of inputs. Most of the concluded results are of theoretical in nature and a few practical gains in running time which has been demonstrated for Binary Sort algorithm and evaluated much efficient as compared to already developed non-adaptive algorithms.

In Section 2 some previous works relating to the sorting algorithms is discussed briefly along with the preliminaries of the sorting. Section 3 introduces the classical proposed algorithm followed by working of the proposed algorithm. In section 4 we have the analysis and result of the proposed algorithm and followed by Section 5 for discussion in detail. Section 6 summarizes and concludes this paper after discussion of the results.

II. PRELIMINARIES & RELATED WORK

A. Sorting-Definition

One of the most common applications in computer science curriculum is sorting, through which data is arranged according to their values. Let A be a list of n elements A_1, A_2…………..A_n in memory. Sorting of A means the operation of rearranging the contents of A so that they are increasing in order (numerically or lexicographically), so that $A_1 <= A_2 <= A_3 <= A_4 \ldots\ldots\ldots <= A_n$.

Since A (list of n elements must be linear array) has n elements, there are n! ways that the contents can appear in A. These ways correspond to the n! permutation of 1, 2, 3……..n. accordingly, each sorting algorithm must take care of these n! possibilities.

B. Classification of Sorting Algorithms

1) Based on data size: Sorts are generally classified as either external sorting or internal sorting. An internal sort is the sort in which all the data is held in the primary memory during the sorting process. An external sort uses primary memory for the data currently being sorted and secondary storage for any data that will not fit in the primary memory.

2) Based on the information about data: Comparison based sorting: A comparison based algorithm orders a sorting array by weighing the value of one element against the value of other elements. Algorithms such as quick sort, merge sort, heap sort, bubble sort, and insertion sort are comparison based. Non-comparison based sorting: A non-comparison based algorithm sorts an array without consideration of pair wise data elements. Bucket sort, radix sort are example of non comparison based.

C. Constraints Of Sorting Algorithms

Efficient sorting is important to optimize the use of other algorithms that require sorted lists to work correctly; it is also often useful for producing human-readable output. More formally, the output must satisfy two conditions:

- The output is in non decreasing order.
- The output is a permutation, or reordering of the input.

D. Classification Of Sorting

1) On Basis Of Complexity: One way to classify sorting algorithm is according to their complexity. There are so many other ways in which one of them is based on internal structure of the algorithm.

- Swap-based sort: These types of sort begin with entire list and exchange between a pair of elements and move till the list become sorted like Bubble Sort.
- Merge-based sort: In these types of sort, it creates initial "naturally" or "unnaturally" sorted sequence and then add either one element or merge two already sorted sequences.
- Tree based sort: It consists of two different types of approaches, one is to create a heap of the given list either minimum heap or maximum heap according to the required list in ascending order and descending order respectively and other is to sort on the basis of search trees.
- Parallel sort: There are several sequential sorting algorithms that use to sort the data points by using the concept of parallelized processing to increase the efficiency of the algorithm.

2) On Basis Of Other Criteria: Another way to classify the sorting algorithm based on other criteria as described.

- Computational complexity: (best, average and worst case) in terms of the size of list (n). Good average number of comparisons is O (n log n) and bad is $O(n^2)$.
- Stability: Stable sorting algorithm maintains the relative order of record with equal keys while unstable sorting algorithm does not maintain the relative order of record with equal keys.
- Memory usage / Use of the computer resource: Some sorting algorithm are "in place" such that O(1) or O(n log n) memory is needed beyond the items being sorted, while others need to create auxiliary locations for data to be temporarily store.
- Recursion: Some algorithms are either recursive or non recursive while other may be both (like merge sort).
- Comparison-based: Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator.

Since the dawn of computing, the sorting problem has attracted a great deal of research from last decade perhaps to reduce the complexity of sorting algorithm and to increase its efficiency. There are several already developed sorting algorithms which are slow, simple and intuitive, such as Bubble Sort developed in 1956 by Knuth [11], Selection Sort, Radix Sort and Insertion Sort have a theoretical complexity of O (n^2) that limits its usefulness to small number of elements no more than a few thousand data points. The quadratic time complexity of existing algorithms such as Bubble Sort, Selection Sort and Insertion Sort, Radix Sort limits their performance when array size increases. And Of course, if an application needs a faster sorting algorithm, there are certainly many algorithms which are already developed till now , including Quick sort (extremely complicated), Merge sort, and Heap sort to produce lightning-fast results but still there is lack of theoretical complexity to be reduced which is essential to get efficient algorithm. These most of the algorithms like Quick Sort have a theoretical complexity of O (n log n) in average case and have O (n^2) in worst case. These all already developed algorithms still are not able to satisfy some of the problems in computer science efficiently due to more complexity and run time of the Sorting Algorithm. In the various Sorting Techniques that already exists are very much complex.

III. PROPOSED ALGORITHM (BINARY SORT)

A. Algorithm

```
BinarySort (a , n)
{
//a is defined as an global array a[1:n];
//p is defined as a global sub array p [1: n];
//q is defined as a global sub array q [1: n];
//n is the total number of the elements in the defined array;
If n is less than or equal 2
{
            If first element is greater than second and n is 2 then do
    {
            Interchange a[1] & a[2];
}
End if
Get sorted element pairs here; //Put in array a or Print the elements;
}
Else
{
Assign value to min and max by very first element of array.
for i : = 1 to n do
{
    If (a[i] > max) then max = a[i];
    Else if (a[i] < min) then min = a[i];
//Get maximum and minimum element from the array.
}
End for
Calculate MID value of the MAXIMUM & MINIMUM element found.
For i : = 1 to n do
{
    If(a[i] < mid) then {  Increment Count1 by 1;  and  P[Count1]=a[i] }
    Else if (a[i] > mid) then { Increment Count2 by 1; and Q[Count2]=a[i] }
    //Divide the major array to sub-arrays;
    //Count1 and Count2 are counters to make check on the size of sub-arrays generated.
}
End for
BinarySort (P, Count1);
  BinarySort (Q, Count2);
}
End if
}
```

B. Working Procedure Of Proposed Algorithm

In this algorithm, in the complex array list first we find the maximum and minimum element and find the mid value and taking mid value as reference we split the complex array into two sub arrays in which one sub array contains all the elements smaller than the mid value but they are inserted in the sub array without altering the sequence of their index number and similarly we develop the another sub array that have all the elements greater than the mid value without altering their sequence of index number and afterwards we apply the same algorithm on the sub arrays recursively till it reach to the base case defined that is sub array of only two elements and then if the first index numbered element is greater than the second one then swapping takes place else they are a sorted sub list. This procedure follows till the recursive call ends or we can say that the stack containing the recursive copies of the sub arrays eliminates and stack[top] = -1 as defined and the whole list is sorted using divide and conquer technique with minimal time and complexity as shown in figure 3.2.1a.

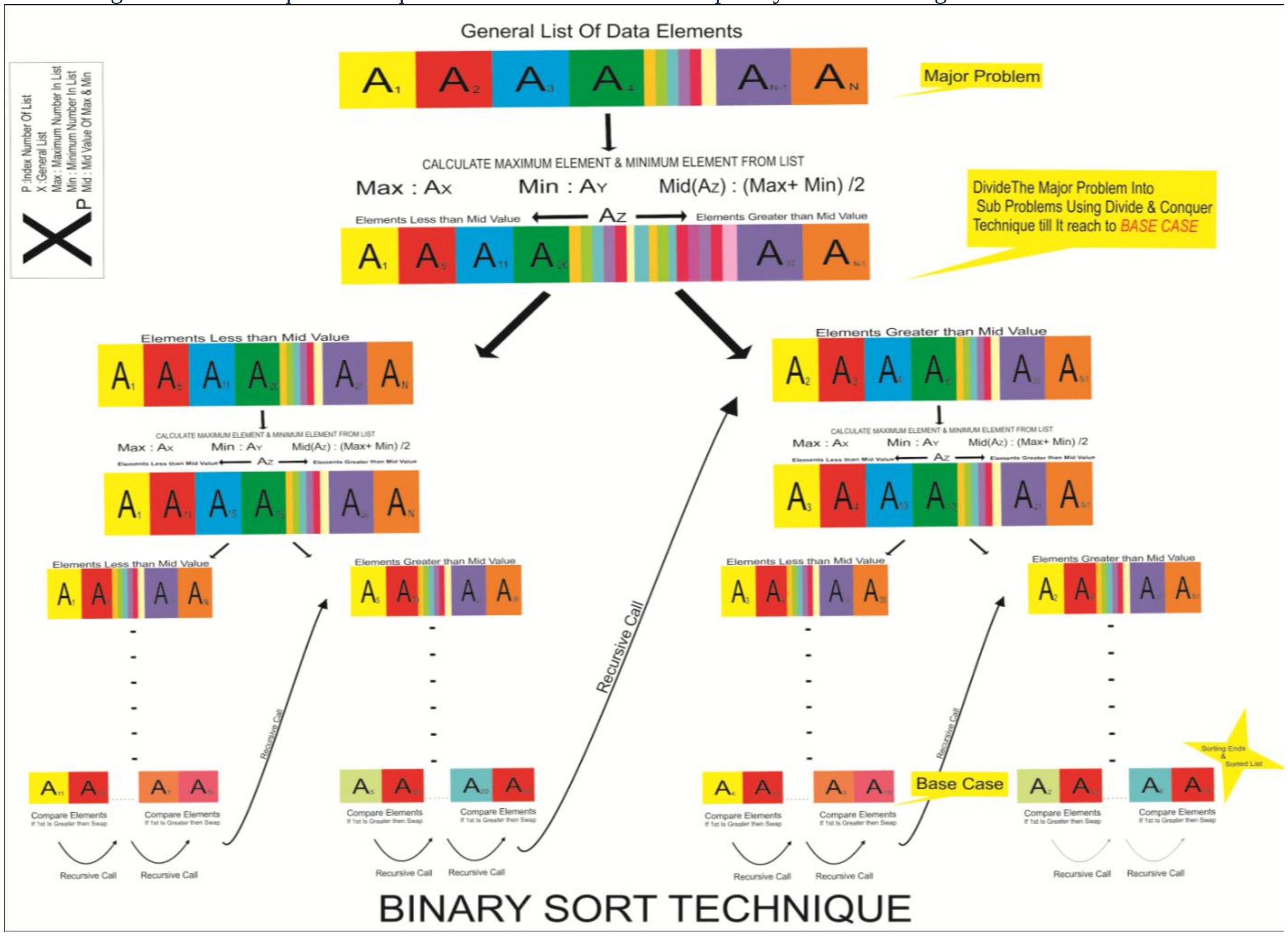

General List Of Data Elements
Major Problem
P :Index Number Of List
X :General List
Max : Maximum Number In List
Min : Minimum Number In List
Mid : Mid Value Of Max & Min
CALCULATE MAXIMUM ELEMENT & MINIMUM ELEMENT FROM LIST
Max : Ax
Min : Ay
Mid(Az) : (Max+ Min) /2
Elements Less than Mid Value
Elements Greater than Mid Value
DivideThe Major Problem Into Sub Problems Using Divide & Conquer Technique till It reach to BASE CASE
Recursive Call
Compare Elements
If 1st Is Greater then Swap
Base Case
Sorting Ends & Sorted List
BINARY SORT TECHNIQUE

Fig.1 General working procedure of binary sort

1) *Practical Example:* Considering a Data array with elements 9,3,14,2,7,11 and using the proposed protocol(binary sort) the implementation is shown in figure 3.2.2a.

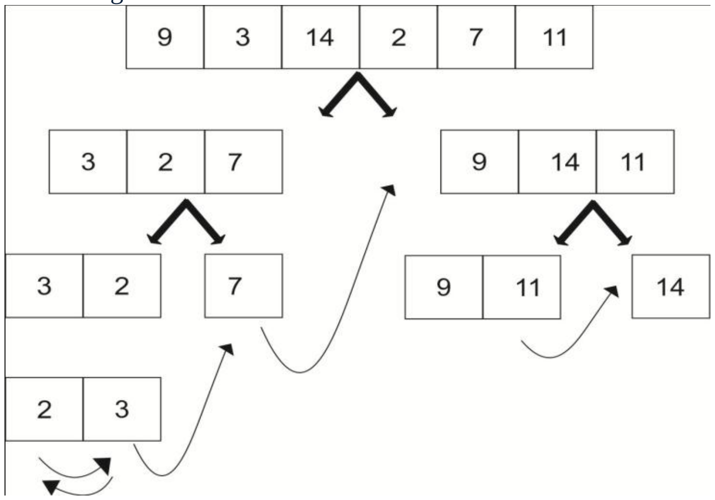


Fig. 2 Example of a binary sort

Step 1: Max number = 14 and Min Number = 2
Mid= (Max +Min)/2
= 8
Step 2: Array P generated with all the elements Less than or equal to 8 without altering index number.
Step 3: Array Q generated with all the elements Greater than 8 without altering index number and place in the stack.
Step 4: Array p again follow the algorithm by recursive call.
Max number = 7 and Min number = 2
Mid= (Max +Min)/2
= 4.5
Step 5: Again Array P generated with all the elements Less than or equal to 4.5 without altering index number.
Step 6: Array Q generated with all the elements Greater than 4.5 without altering index number and Place in the stack.
Step 7: Now array P satisfy the condition of the base case so check for swapping if needed and after wards all the sub arrays on the stack pop one by one follow the same procedure.

IV. ANALYSIS & FORMATIVE-EVALUATION OF RESULT

A. Analysis On Various Constraints

1) Based on data size: The Binary Sort Algorithm is a non adaptive algorithm which does not depend upon the data size that is to be sorted. Using the Binary Sort, there is no variation in the complexity and resulting time for the sorting of an array list having thousands of data elements. This technique is very useful to solve the major problems in the domain of computer science.

2) Based on the information about data: The Binary Sort is a Comparison based sorting. A comparison based algorithm orders an array by weighing the value of one element against the value of other elements. It is similar to the algorithms such as quick sort, merge sort, heap sort which are also categorized in comparison based sorting.

3) Constraints Of Sorting Algorithms: The Binary Sort Algorithm satisfies both of the conditions as mentioned in preliminaries ,Binary Sort algorithm's complexity and results do not vary if the data elements are already in decreasing or increasing order and the resultant of the elements that are to be sorted is obtained by the rearranging elements as least as possible.

B. Analysis On Run Time

The run-time complexity for the worst-case/average case scenario of a proposed algorithm can be evaluated by examining the structure of the algorithm and making some simplifying assumptions. A given computer will take a discrete time to execute each of the instructions involved with carrying out this algorithm. The specific amount of time to carry out a given instruction will vary depending on which instruction is being executed and which computer is executing it, but on a conventional computer, this amount will be deterministic. Say that the actions carried out in step 1 are considered to consume time T_1, step 2 uses time T_2, and so forth.

In all the phases of Binary Sort algorithm is invariant to no. of inputs. It is always divide the major array into two sub arrays of equal size recursively .Thus we have $k = n/2$ and $n - k = n/2$ for the original array and consider the size of the array list to be of size n. While analyzing run time of binary sort,we assume that all the elements are distinct in the the data list.

The reoccurrence relation we derived for the binary sort is given by:

$$T(n) = \begin{cases} \Delta & \Delta \text{ is constant for } n=1 \\ \mu & \mu \text{ is constant time for swapping when } n=2 \\ 2T(n/2) + \zeta n & \text{Otherwise} \end{cases}$$

Fig. 3 Function defined for run time analysis

Here in analyzing the run time ,the constant time required for n=1 i.e Δ and the constant time required for n=2 i.e μ can be neglected as compared to splitting and sorting for $n \rightarrow \infty$.

$T(n) = 2T(n/2) + \zeta n + \Delta + \mu$
On neglecting Δ and μ we obtain
$T(n) = 2T(n/2) + \zeta n$
$T(n) = 2(2T(n/4) + \zeta n/2) + \zeta n$
$T(n) = 2^2 T(n/4) + 2\zeta n$

$T(n)= 2^2(2T(n/8) + ζn/4) + 2ζn$

$T(n)= 2^3T(n/8) + 3ζn$

:

:

$T(n)= 2^kT(n/2^k) + kζn$

Here this recurrence will continue only until $n = 2^k$ (otherwise we have $n/2k < 1$) ; until $k = \log n$. Thus, by putting $k = \log n$, we have the following equation:

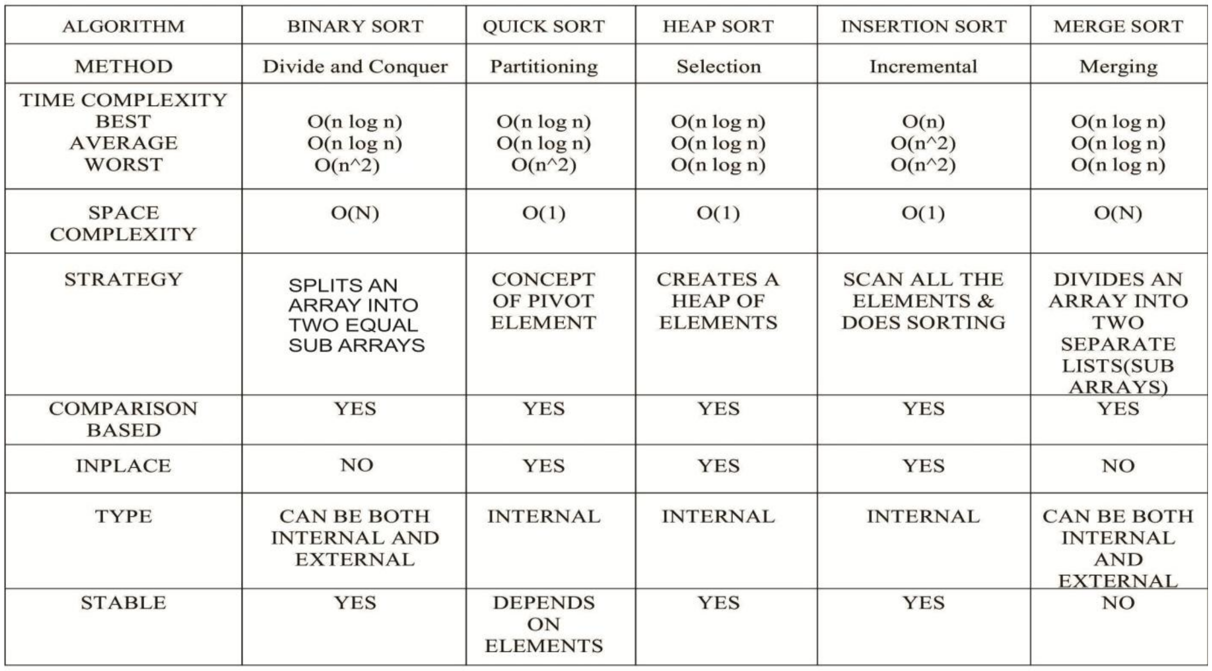
$$T(n) = nT(1) + ζn \log n$$
$$T(n)= O (n \log n)$$

where ζ is the constant value. This is valid in average for binary sort and which comes out to be stable and comparative to the already developed non adaptive algorithms. For worst case we have comparative to quick sort is $O (n^2)$.

C. Analysis On Space:

On the analysis of space complexity, the binary sort utilizes the space of O (n) on the theoretical and formal analysis.

D. Comparison With Algorithms & Performance



ALGORITHM	BINARY SORT	QUICK SORT	HEAP SORT	INSERTION SORT	MERGE SORT
METHOD	Divide and Conquer	Partitioning	Selection	Incremental	Merging
TIME COMPLEXITY BEST AVERAGE WORST	O(n log n) O(n log n) O(n^2)	O(n log n) O(n log n) O(n^2)	O(n log n) O(n log n) O(n log n)	O(n) O(n^2) O(n^2)	O(n log n) O(n log n) O(n log n)
SPACE COMPLEXITY	O(N)	O(1)	O(1)	O(1)	O(N)
STRATEGY	SPLITS AN ARRAY INTO TWO EQUAL SUB ARRAYS	CONCEPT OF PIVOT ELEMENT	CREATES A HEAP OF ELEMENTS	SCAN ALL THE ELEMENTS & DOES SORTING	DIVIDES AN ARRAY INTO TWO SEPARATE LISTS(SUB ARRAYS)
COMPARISON BASED	YES	YES	YES	YES	YES
INPLACE	NO	YES	YES	YES	NO
TYPE	CAN BE BOTH INTERNAL AND EXTERNAL	INTERNAL	INTERNAL	INTERNAL	CAN BE BOTH INTERNAL AND EXTERNAL
STABLE	YES	DEPENDS ON ELEMENTS	YES	YES	NO

Fig.4 Comparison of binary sort with other sorting techniques

We evaluated the theoretical and empirical complexity of Binary Sort which comes out to be O(n log n) after the careful analysis in both the average and worst case. This algorithm is easy to implement, works very well for different types of input data, and is known to use fewer resources as compared to any other sorting algorithm [8].On comparison with other already existing algorithms Binary Sort is equivalent to merge sort but stable in nature rather than merge sort is not stable. It has a better running time than the simplest Bubble Sort, Selection Sort and Insertion Sort algorithm and almost comparative to the Quick Sort and Heap Sort. On ground of the above Binary Sort is defined in all the major projects where an efficient and optimistic sorting technique is required. The proposed sorting technique works in a similar way independent of the number of input data elements and other factors effecting the performance of sorting.The run time formal analysis comparison is discussed as shown in Section 4.6

TABLE I
COMPARISON ON THE BASIS OF RUN TIME ANALYSIS

Sort Type	**n=100**	**n=1000**	**n=10000**	**n=100000**	**n=1000000**
Binary Sort	3.9	7.6	62	1922	2089
Quick Sort	4	8	65	1984	2136
Bubble Sort	8	73	697	54963	71402
Insertion Sort	11	42	227	21407	23647
Selection Sort	12	63	940	34513	39214

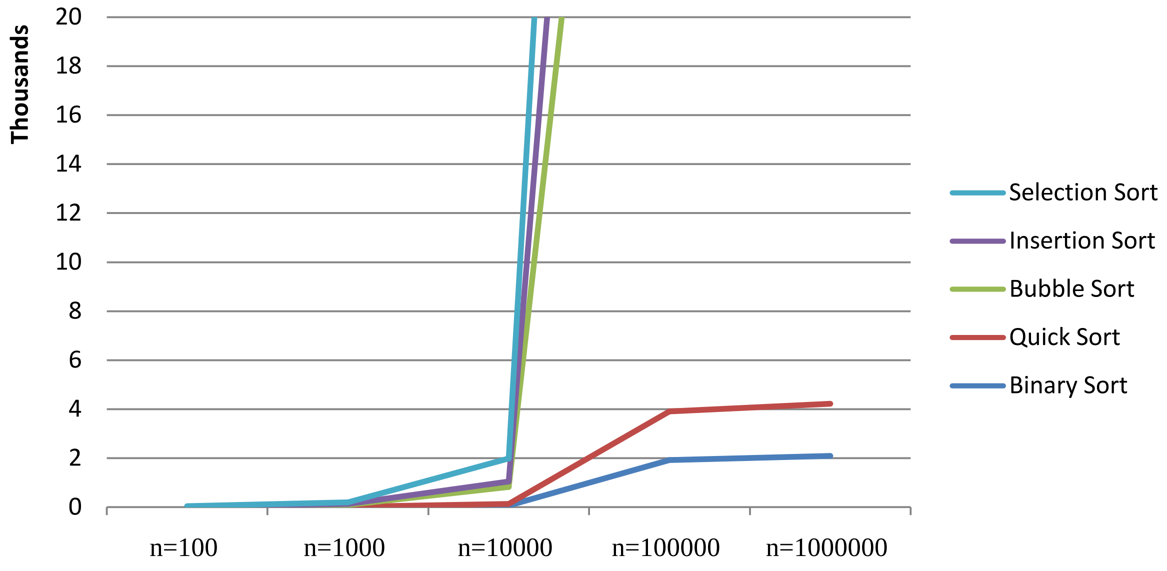
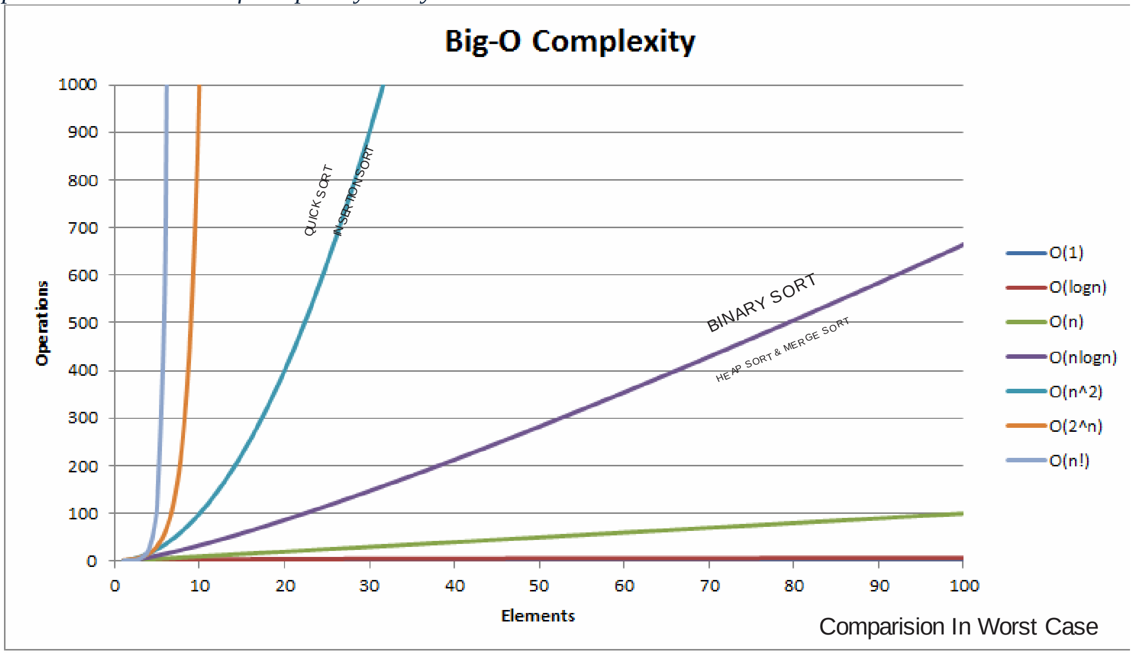


Fig. 5 Graphical Representation Of the Comparision Of Run Time Analysis

F. Comparison on the basis of complexity analysis



V. DISCUSSION

In this paper, a new sorting technique has been introduced which is very much efficient in each and every domain of the computer science and can be implemented in any of the practical region to get best results.

The algorithm of Binary Sort is based on the one of the best technique to deal with the complex problems called Divide and conquer. Using this technique we divide the complex problems into the sub problems and soon till we get the destination of the base case which returns us our first sub solution and afterwards combining all the results we get a solution to the problem. This algorithm is developed to have stability in the algorithm which can handle variety of situations in any environment. More specifically, it is an attempt to have a new technique of sorting which is quite similar to the binary search in an array list and reduce the complex structure of the algorithms and the pseudo code already developed and make it more users friendly and easy to understand for the developers. Binary Sort, which is an efficient algorithm based on the Divide and Conquer [4] technique. We evaluated the theoretical and empirical complexity of Binary Sort which comes out to be O (n log n) after the careful analysis in both the average case and O (n^2) worst case. This algorithm is easy to implement, works very well for different types of input data, and is known to use fewer resources as compared to any other sorting algorithm [9]. It has a better running time than the simplest Bubble Sort, Selection Sort and Insertion Sort algorithm and almost comparative to the Quick Sort and Heap Sort. It yields around 60% performance improvement over the Bubble sort, Insertion sort and Selection Sort [11]. Similar to the most classic non adaptive sorting algorithms like Quick Sort, Heap Sort [7, 10], and Merge Sort [2], Binary Sort is also non adaptive and its time complexity is O (n log n) irrespective of the number of inputs. Most of the concluded results are of theoretical in nature and a few practical gains in running time which has been demonstrated for Binary Sort algorithm and evaluated much efficient as compared to already developed non-adaptive algorithms.

VI. CONCLUSION

In most of the practical situations to squeeze the efficient results we must have to use the algorithm which satisfies all the constraints and can be adapted in any of the provided valid syntax by the standard (e.g., Java, C, or C++) libraries. However, general sorts do not work in all situations that vary in the domain but Binary Sort can be utilized in any of the way for approximate all of the problems till defined as it can be implemented with the accordance of any of the standard language library. It is most efficient as approx 60% more efficient than basic algorithms with the complexity of O (n log n).

REFERENCES

[1] C. A. R. Hoare, Algorithm 64: Quick sort. Comm. ACM, vol. 4, no. 7 (1961), pp. 321.
[2] D. E. Knuth. The Art of Computer Programming, Volume III: Sorting and Searching. Addison-Wesley, Reading, MA, 1973.
[3] D. Knuth, “The Art of Computer programming Sorting and Searching”, 2nd edition, Addison-Wesley, vol. 3, (1998).
[4] E. Horowitz, S. Sahni and S. Rajasekaran, Computer Algorithms, Galgotia Publications.
[5] G. Franceschini and V. Geffert, “An In-Place Sorting with O (n log n) Comparisons and O (n) Moves”, Proceedings of
44th Annual IEEE Symposium on Foundations of Computer Science, (2003), pp. 242-250.
[6] I. Flores, “Analysis of Internal Computer Sorting”, ACM, vol. 7, no. 4, (1960), pp. 389- 409.
[7] J. W. J. Williams. Algorithm 232 Heap sort. Communications of the ACM, 7(6):{347,348}1964.
[8] Knuth, D. The Art of Computer Programming, Sorting and Searching, 2 ed., vol. 3. Addison-
Wesley, 1998.
[9] R. Sedgewick, *Algorithms in C++*, 3rd edition, Addison Wesley, 1998.
[10] R. W. Floyd. Algorithm 245: Treesort3. Communications of the ACM, 7(12):701, 1964.
[11] Selection Sort, http://www.algolist.net/Algorithms/Sorting/Selection_sort.
[12] Shahzad B. and Afzal M., “Enhanced Shell Sorting Algorithm,” Computer Journal of Enformatika, vol. 21, no. 6, pp. 66-70, 2007.