*J. Appl. Environ. Biol. Sci.*, 4(7S)406-410, 2014

© 2014, TextRoad Publication

ISSN: 2090-4274
Journal of Applied Environmental and Biological Sciences
*www.textroad.com*

# Efficient Sort Using Modified Binary Search- A New Way To Sort

**Masood Ahmad, Ishtiaq Wahid, Abdusalam, Nawsher Khan**

Department of Computer Science,
Abdul Wali Khan University, Mardan.

*Received: September 1, 2014*
*Accepted: November 13, 2014*

**ABSTRACT**

In a single iteration of Insertion sort, only one element is inserted to its proper position. In this technique the proper position of the item can be found using linear search algorithm starting from first element and proceed in incremental fashion. The linear search can be applied either in incremental fashion or in reversed order to find the proper position of the element in the sorted part of the array. So it takes a huge amount of time to search an item when the size of the array is very large. In the proposed technique, binary search instead of linear search is used to find the proper place of an item in the array. Binary search is applicable because the part of the array in which the new element is to be inserted is in sorted form. Similarly the average case running time is further reduced such that the new element to be inserted is first compared with last element of sorted part of the array. If it is greater than the last value of sorted part of the array, no need to perform binary search because the element is in its proper place. Similarly if the element of interest is less than last element of sorted part of the array then it is also compared with the very first element of the array. If it is less than the first element, then binary search cannot be performed and the element is inserted to the first position of the array. The proposed algorithm is compared with insertion sort, binary insertion sort and shell sort. Simulation results show that it is very efficient than other techniques.

**KEYWORDS:** sorting, searching applications, modified binary search.

## 1 INTRODUCTION

Sorting is very important in many Commercial institution such as Government organizations, financial institutions, and enterprises. They organize much of this information by sorting it. In these organizations either the information is accounts to be sorted by number or name, mail to be sorted by address or zip code, transactions to be sorted by date or time, files to be sorted by date or name, or whatever else. Sorting has a wide range of applications. First, sorting provides basis for many algorithms like searching, digital filters, pattern matching etc, in addition to numerous applications in pattern matching, data communication, data processing and statistics and database systems have been found Donald EK, (1975), Abdul W, et al (2014), Sumit Get al (2013). The second important role it plays is in the teaching programming, data structure and algorithm design and its analysis. Lastly and particularly, it is one of the demanding dilemma that has been systematically deliberated Donald EK, (1975), Robert S, Charles ARH (1961), Charles ARH (1962, Charles ARH (1992, Robert S (1975, the performance is significantly enhanced Robert S (1975), Charles ARH (1978), Robert S (1980) and measured the lower-bound of complexity has been reached Donald EK (1975), Robert S. (1988), Stubbs DF, Webre NW (1993), Baase S (1978).

Insertion sort Thomas H C (2001), et al is a stable algorithms in which the elements are to be inserted to its proper position one by one. In this technique, linear search is used to find the proper position of an element in sorted part of the array. This algorithm takes O (n) time in worst case. In worst case the shifting of items is done n times and searching also takes O (n) time in worst case. So it takes a total of O ($n^2$) time in worst case.

The average case and worst case running time can be reduced by eliminating linear search and introducing binary search for search purposes. In this technique the average case running time is reduced but the worst case running time is O (n log n). In this technique if the elements are in reverse order then

***Corresponding Author:** Masood Ahmad, Department of Computer Science, Abdul Wali Khan University, Mardan.

searching will take O (log n) time but shifting elements will take O (n) time which results in overall O (n log n) time. It is beneficial if the array elements are randomly located.

In the proposed technique, binary search is used to find the proper position of the element in the sorted part of the array. Before performing binary search, the item of interest is first compared with the first element of the array; if it is less than first element then it is inserted into first location of the array with without binary search. If it is greater than first element then comparison with last element of sorted part of the array is performed. If it is greater than last element of sorted part; then no other computations are required because the element is in its proper place. The proposed algorithm always sorts the array in less than O ($n^2$) steps, even if the array is in reverse order. So this algorithm performs fast in both average and worst cases. Simulation results show that it performs better in all cases compared to existing techniques.

The paper is organized as follows; section II describe proposed algorithm, section III summarize mathematical and simulation analysis. And the paper is concluded in section IV.

## 2. Modified binary search sorting

In this technique, first the number of computational steps can be reduced by using binary search when the item lies between first and last element of the sorted part of the array. The search and shift operation can also be avoided when the element of interest in greater than the last element of the sorted part of the array or less than first element of the array. The pseudo code of the proposed technique is given as:

**Pseudo code of Algorithm 1**

```
1. Modified Binary InsSort (Array, N)
2. {
3. for i←1 to N
4.       {
5.       if (a[i]<=a[i-1]
6.             {
7.             if(a[i]<=a[0])
8.             Mid=0;
9.             else
10.    Mid=Binary Search (Array, I)
11.            Swap A[i] & A[mid]
12.            } //end if
13.      } //end for
14. } //end Modified Binary InsSort
```

In the above pseudo code the algorithm iterates for N times where N is the no of elements of the array. In this section the proposed algorithm and its steps are discussed with the help of an example.
Suppose we have an array:

| **Index** | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| **Elements** | **5** | 6 | 3 | 1 | 4 | 2 | 8 | 7 |

Step1: No need to apply search and shift because the element at index 111 is greater than index 110 element. The resultant array after step 1 is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **5** | **6** | 3 | 1 | 4 | 2 | 8 | 7 |

Step2: compare index three elements to index 111 so its proper place would be index 110 to index 111. Compare it with first index element; since its less than 5 so it will be inserted at position 1. no need for binary search because the element proper place is known. Just shift the elements and insert 3 into index 1. So the result is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **3** | **5** | **6** | 1 | 4 | 2 | 8 | 7 |

Step3: compare element at index 113 to element at index 112. Index 112 element is greater. Now compare it to index 111. Index 110 element is also greater so no need to search. Just shift elements and insert element at index 113 to index 110.

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **1** | **3** | **5** | **6** | 4 | 2 | 8 | 7 |

Step4: compare index 114 elements to index 113; it is less than index 113 element so compare it with index 110 element; it is greater than index 110 element. The element lies between index110 and index 114 so apply binary search. After shifting and insertion the resultant array is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **1** | **3** | **4** | **5** | **6** | 2 | 8 | 7 |

Step5: compare index 115 elements to index 114; it is less than index 114 element so compare it with index 110 element; it is greater than index 110 element. The element lies between index110 and index 115 so apply binary search. After shifting and insertion the resultant array is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **1** | **2** | **3** | **4** | **5** | **6** | 8 | 7 |

Step6: no need to apply search and shift because the element at index 116 is greater than index 115 element. The resultant array after step 5 is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **1** | **2** | **3** | **4** | **5** | **6** | **8** | 7 |

Step7: compare index 117 elements to index 116; it is less than index 116 element so compare it with index 110 element; it is greater than index 110 element. The element lies between index110 and index 117 so apply binary search. After shifting and insertion the resultant array is:

| Index | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
|---|---|---|---|---|---|---|---|---|
| Elements | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |

## 3. Mathematical and Simulation Analysis

In insertion sort the worst case occurs when the array is in reverse order. The best case is when the array is already sorted. The proposed technique takes O (n) time either when the array is in reverse order or it is already sorted. In the worst case it will take O (n log n) time. There is no need to perform the search as well as shift operation if the array is in reverse order as shown in the above section.

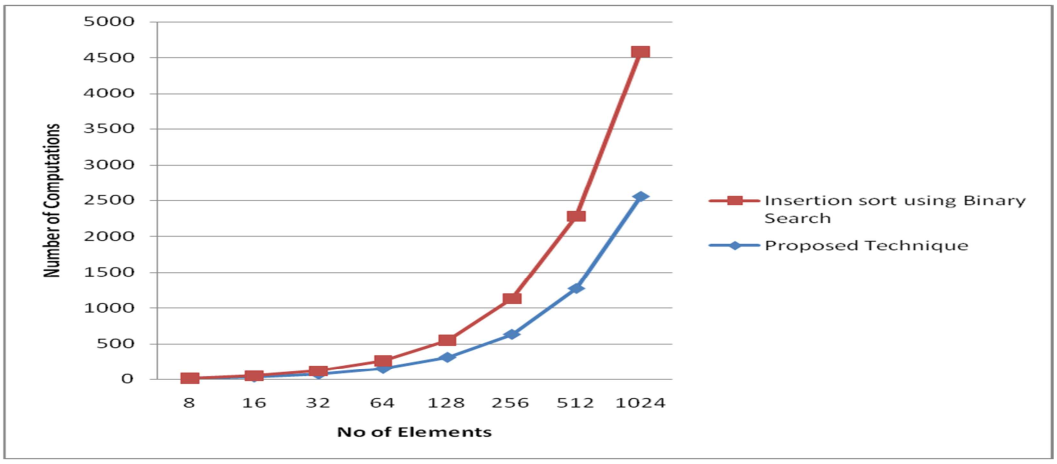

Figure 1: Comparison of binary insertion sort and proposed technique

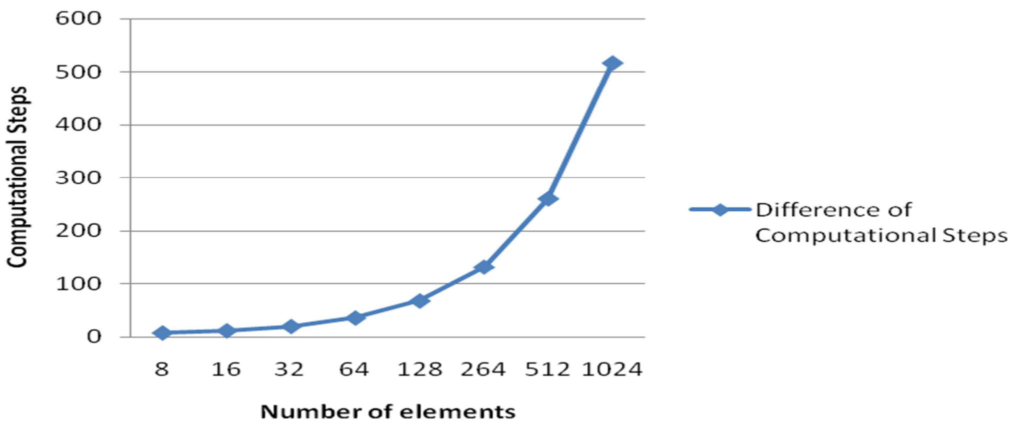

Figure 2: Number of computational steps reduced with each input

The simulation results as shown in Figure 1 demonstrate that the difference between the numbers of computations is increasing as the size of the input increases. The proposed technique is very helpful when the number of elements is very large. The number of computational steps will be dramatically very less when apply to large amount of data as compared to existing techniques such as insertion sort, binary insertion sort, shell sort and other sorting algorithms.

As shown in Figure 2, the differece of computational steps of proposed tecnique and binary insertion sort algorithm increases as the number of elements are increased. This show a significant advantage with large amount of data.

## 4. Conclusion

Sorting is very important for a variety of applications such as databases. In this paper an efficient algorithm is proposed and the simulation results are compared with other techniques. The array is sorted using binary search and is further optimized by comparing the element of interest with first and last element of the sorted part of the array. This result in reducing the complexity to O (n) when either the array is sorted are in reverse order and will take O (n log n) time in worst case.

## REFERENCES

1. Donald EK, (1975) The Art of Computer Programming. 2nd ed. Boston, MA: Addison-Wesley.
2. Robert S. Algorithms (1988), 2nd ed. Boston, MA: Addison-Wesley Series in Computer Science, pp. 111-113.
3. Charles ARH (1961) Algorithm 64: Quicksort. Communications of the ACM 1961; 4: 321.
4. Charles ARH (1962). Quicksort. BCS Computer Journal 5: 10-15.
5. Charles ARH (1992). Essays in Computer Science. Upper Saddle River, New Jersey: Prentice-Hall.
6. Robert S (1975). Quick sort. PhD, Stanford University.
7. Charles ARH (1978). Implementing Quicksort Programs. Communications of the ACM 21: 847-856.
8. Robert S (1980). Quicksort. New York, NY, USA: Garland publishing.
9. Huang BC, Donald EK(1986). A one-way, stackless quicksort algorithm. BIT Numerical Mathematics 26: 127-130.
10. LindermanJP(1984). Theory and Practice in the Construction of a Working Sort Routine.Bell System Technical Journal 63: 1827-1843.
11. Stubbs DF, Webre NW (1993), Data Structures with Abstract Data Type and Ada. PWS-Kent Pub Co. pp.301-341.
12. Baase S (1978) Computer Algorithms -Introduction to Design and Analysis. Edinburgh Gate Harlow CM20 2JE, UK: Addison-Wesley pp. 58-132.
13. Thomas H C, Charles E L, Ronald L R, Clifford S (2001). Introduction to Algorithms 2nd ed. London, W1W 6AN, UK: MIT Presspp. 323–69.
14. Abdul W, et al (2014) Cognitive Storage Model And Mapping With Classical Data Structures, VAWKUM Transactions on computer sciences, Vol 3, No 1 pp.7-9.
15. Sumit Get al (2013), "machine learning cascade algorithm for analyzing shelf life of processed cheese", VAWKUM transactions on computer sciences, Vol 2, No 1. Pp.