IBBUL
Lapai Journal of Applied and Natural Sciences
LAJANS Vol 4(1): 106-113

RESEARCH PAPER

Performance Analysis of Bubble Sort Based And Insertion Sort Based Searching Techniques On 2-D Array

Oladipupo E.T., Olayemi, D. O., Adeagbo C.O. and Olayemi, T.A.

Department of Computer Science, Federal Polytechnic, Bida, Nigeria

ABSTRACT

Row wise and column wise 2-D array search algorithm is a popular searching technique. Bubble and Insertion sort algorithms are also known for their effectiveness in sorting arrays that contain small data. In this research, bubble and insertion sort algorithms are applied in the sorting of the randomly generated elements of two dimensional arrays. The arrangement is done in such a way that every row is progressively sorted from left to right and the last number in each row is less than the first number of the next row. Row wise and column wise search technique is then applied in searching for any element in the 2-D array. The objective is to measure and compare the time taken by row wise and column wise search technique on 2-D array when sorting method used in the sorting of the element of the array is taking into consideration. A simple program where bubble sort, insertion sort and row wise and column wise matrix search algorithms were implemented was developed using Matrix Laboratory (MatLab) programming Language. The internal system clock is set to monitor the time duration to complete the computation in each of bubble sort based and insertion sort based search methods. The results indicated that the insertion sort based search takes lesser time than bubble sort based search when applied to 2-D arrays.

Keywords: Insertion sort, bubble sort, 2-D arrays, time complexity

INTRODUCTION

In most areas of works of life, searching and sorting activities cannot be ruled out. For instance, a teacher who wants to know the particulars of the student who score the highest mark in a test will have to engage him/herself in searching for highest score among the scores of student that took the test. A dictionary is a sorted list of word definitions. A telephone book is a sorted list of people's names, addresses and telephone numbers. Basic search techniques provide the key to many historically important accomplishments in the area of artificial intelligence. As rightly noted, an important property of most of the significant search problems studied in artificial intelligence is that they suffer from combinatorial explosion (Balogun & Sadiku, 2013).

In view of wide range of application of searching and sorting in different works of life and the quest to have efficiency in the process of searching and sorting, different techniques for searching and sorting have been developed. For example, searching techniques include: linear search, binary search, breadth-first search, Depth-First search, Hill climbing heuristics and Best-First heuristics to mention few. Various sorting algorithm include bubble, insertion and heap sort. Each of the different techniques for searching and sorting has its strength and weaknesses.

This paper reports the results of the performance analysis carried out when bubble and insertion sort algorithms were employed in the sorting of the elements of two dimensional arrays. An efficient algorithm that works in O(n) time was applied as the searching algorithm. The duration for completion of sorting and searching for each of the bubble sort based and insertion sort based searches are measured using system clock and recorded. The results for each sorting algorithms are compared.

Received 18 June, 2019
Accepted 25 September, 2019
Address Correspondence to:
taiwotheophilus@gmail.com

Ashima, Meenu, and Swati (2016) observed that if the elements of a two dimensional sorted array is such that every row is progressively sorted from left to right and the last number in each row is less than the first number of the next row, the matrix can be viewed as a sorted one dimensional array. Contrary to the primitive method of searching an item in such a 2-D array which normally cost O(PQ) time for a 2-D with P number of rows and Q number of columns, a 2-D binary search algorithm whose pseudocode is shown below was proposed.

2-D Binary Search Algorithm

2D_BINARY_SEARCH (mat, start, end, key,mid)

1. SET start = 0, end = rows * cols-1
2. REPEAT
    a. SET mid= start + (end – start)/2
    b. SET Row = mid/cols
    c. SET col = mid%cols
    d. SET value = mat[row,col]
    e. IF value == key THEN GOTO 3

ELSE
IF value > key THEN
SET end = mid -1
ELSE
SET start = mid + 1
END IF
ENDIF
UNTIL start <= end

3. EXIT.

The algorithm was implemented on 2-D array and the analysis of its time complexity was carried out. It was found out that the time complexity of 2-D Binary Search algorithm is 0(n log n) while the time complexity of linear search algorithm when applied to 2-D array was found to be 0(n*n). It has been asserted that binary search algorithm is very efficient for sorted data (Francis, 1987). (Balogun & Sadiku, 2013) however, argued that the assertion does not state categorically the method used in accomplishing the sorting in the first instance, it was therefore opined that not all has been said about the efficiency of the binary search algorithm and that there is the need to take into consideration the efficiency of sorting technique employed before the application of binary search algorithm. In an attempt to improve the performance of binary search algorithm (Karthick, 2016) proposed Odd Even Based Binary search Algorithm where, before comparison starts, the nature of the key (odd or even) has to be determined. If the given key is odd only the odd numbers from the list is searched by completely ignoring the list of even number and vice versa. The sorted list is then presented to Odd Even Based Binary Search Algorithm. It was shown that Odd Even Based Binary Search Algorithm provides efficient results by reducing time complexity and minimizing the usage of computer resources when the list contains combination of even and odd numbers. However, if the list contains completely even or odd numbers, the algorithm takes same time as in case of Binary search. Quadratic Search algorithm developed by (Parveen, 2013) was found to be two times faster than the classical Binary search algorithm. The algorithm takes the advantage of the fact that whereas each step of the Binary Search Algorithm divides the block of items being searched in half Quadratic search divides the block of items being searched into 4 parts at each step. Since a set of n items can be divided in half at most $\log_2$ n times, the running time of a binary search is proportional to log n and can be said to be O ($\log_2$ n). Since a set of n items can be divided into 4 part at most log n/2 time, the running time of Quadratic search algorithm is proportional to log n/2 and it can be said to be O($\log_2$ n/2).

Kazim (2017) carried out a machine independent comparative study of sorting algorithms. The running time of the algorithms considered were purely taken as a mathematical entity and so his analysis was done from a generic point of view. Harish (2014) presented a variant of bubble sort algorithm which was found to have a better runtime complexity than its predecessor.

Ahmed & Zirra (2013) carried out machine specific analysis on selection sort and Quick sort algorithms on integer and character arrays by running on CPU and measure time taken to sort both arrays. Quicksort algorithm was found to require shorter time for sorting array of the same size.

The analysis of bubble, gnome and insertion sort algorithms carried out by Jehad (2015) showed that gnome sort algorithm was the quickest one for already sorted data while

selection sort is quicker than gnome and bubble in unsorted data.

Khalid, Ibrahim, Abdallah, & Nabeel (2013) identified various factors that should be taken into consideration when comparing between various sorting algorithms as: the time complexity, stability and memory space. The time complexity of an algorithm determines the amount of time that can be taken by an algorithm to run. Stability considers whether the algorithm keeps elements with equal values in the same relative order in the output as they were in the input or not (Cormen, Leiserson, Rivest, & Stein, 2009; Goodrich & Tamassia, 2010; Aditya & Deepak, 2008).

METHODOLOGY

Three algorithms: bubble sort, insertion sort and row wise and column wise search algorithm for matrix are implemented. The pseudocode for the three algorithm and the graphical user interface for the implementation of the algorithms in Matrix Laboratory programming language are given below.

Bubble Sort Algorithm

```
Bubblesort(list, size)
REPEAT
        SET  swap = FALSE
        FOR j = 0 TO size
                IF list[j] > list[j+1]     //compare pair of adjacent items
                        SET temp = list[j]
                        SET list[j] = list[j+1]
                        SET list[j+1] = temp
                        SET swapped = TRUE
        NEXT j
UNTIL swapped = FALSE
```

Swap adjacent element if it is in the wrong order

Insertion Sort Algorithm

```
sortedarray = insertionsort(inputarr,row,col)
// convert the matrix to a 1 dimensional array
SET i = 1;
for x = 1 to row

   for y = 1 to col
      SET temparr(i) = inputarr(x,y);
      SET i = i+1;
   END FOR
ENDFOR

//Apply insertion sort to one dimensional array
FOR j = 2 to  row *col
  SET  value = temparr(j)
  SET  k= j
   WHILE ((k > 1) AND temparr(k-1)>value)
    DO
       SET temparr(k)= temparr(k-1)
       SET k= k-1;
   ENDDO
   SET temparr(k) = value;
END FOR
//Convert the 1 dimensional array to 2-dimensional array
i = 1;
FOR  x = 1 TO row
   FOR y = 1 TO col
     SET inputarr(x,y)= temparr(i);
     SET  i = i+1;
   END FOR
```

```
END FOR
SET sortedarray = inputarr
RETURN sortedarray
Row Wise and Column Wise Matrix Search Algorithm
   function MatrixSearch (mat,row,col, key)
     SET  r = 1
      SET c = col
      SET f = 0
       IF mat(1,1)> key OR  mat(row,col)< key
           RETURN r, c f
       ENDIF
       WHILE(r <= row AND r>=1 AND c>1 AND c <= col)
           DO
          IF mat(r,c) < key
            SET r = r+1
          ELSE
             IF mat(r,c) > key
                SET c = c - 1
             ELSE
                IF mat(r,c) == key
                   f = 1;
                   RETURN f, c ,r
                ENDIF
             ENDIF
          ENDIF
ENDDO
```

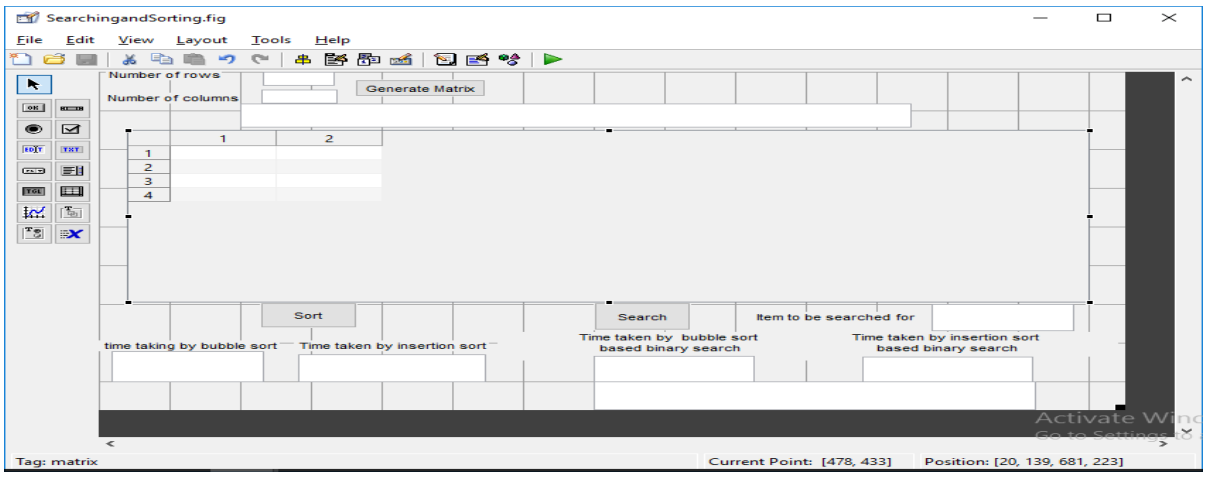

Figure 1: The graphical user Interface of the implementation

RESULTS

The program developed was test run. Sample of the outputs during the test are given in Figure 2 and Figure 3. Table 1 shows the recorded time during the testing for different dimensions of matrix and Figure 4 is the graph of time versus number of elements in the array from where the search was to be made.

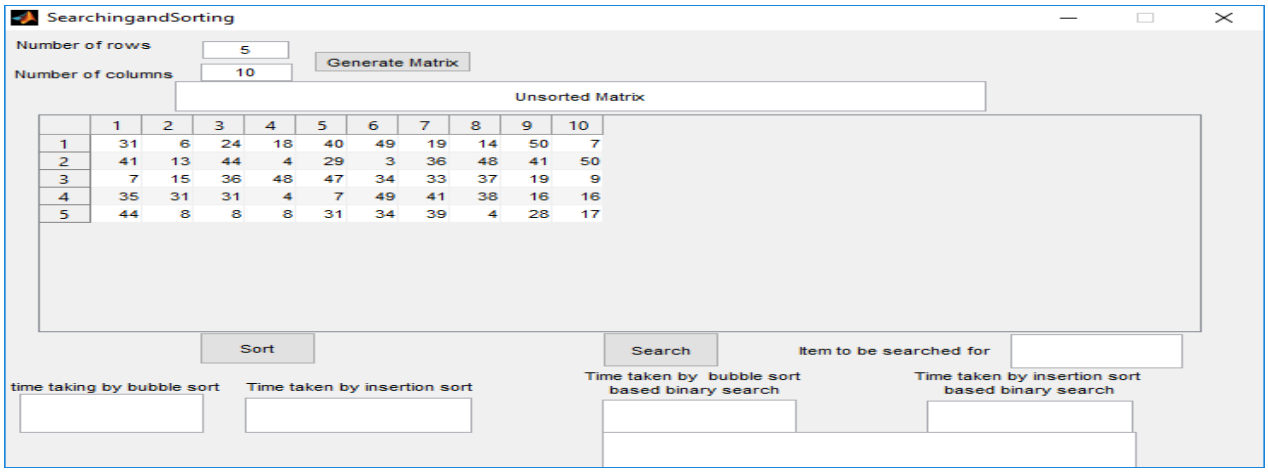

Figure 2: An unsorted 5 by 10 matrix generated as specified in the row and column text boxes

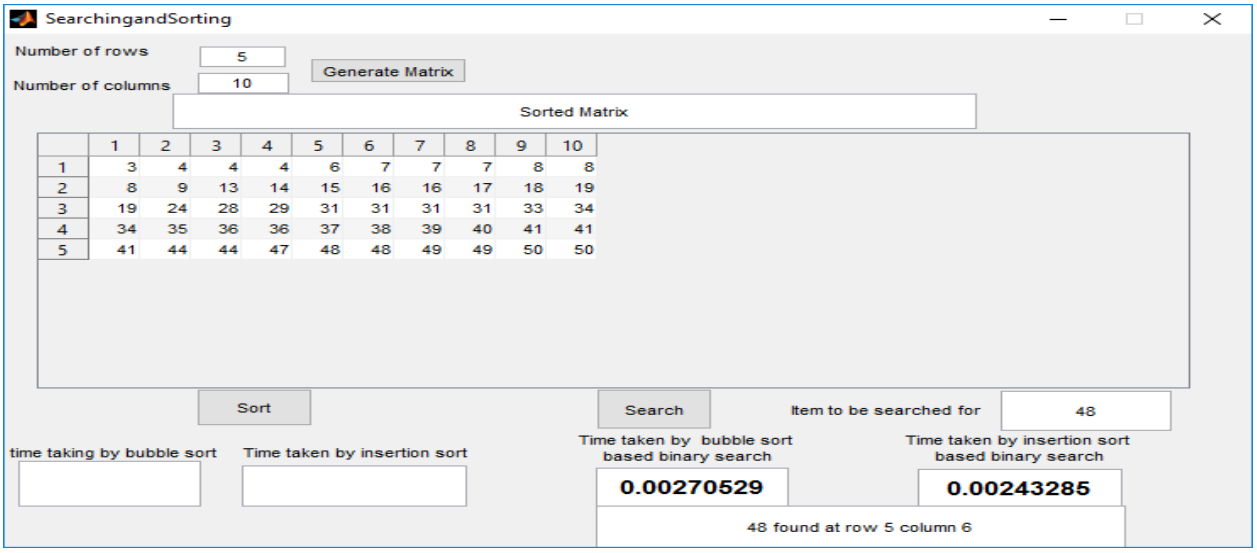

Figure 3: sample output showing the sorted matrix and the time taken for both bubble sort based and Insertion sort based search

Table 1: Measured execution time for both bubble and insertion sort based search

Matrix Dimension	Time Taken For Bubble Sort Based Search (s)	Time Taken For Insertion Based Search (s)	Quantity Of Data In The Matrix
5 X 10	0.003	0.002	50
10 X 10	0.004	0.003	100
10 X 15	0.005	0.006	150
15 X 15	0.008	0.007	225
15 X 20	0.014	0.009	300
20 X 20	0.021	0.011	400
20 X 25	0.031	0.018	500
25 X 25	0.048	0.025	625
30 X 25	0.071	0.034	750
30 X 30	0.096	0.05	900
30 X 35	0.130	0.067	1050
35 X 35	0.176	0.087	1225
35 X 40	0.220	0.106	1400
40 X 40	0.292	0.141	1600
100 X 100	10.383	4.454	10,000
200 X 200	102.192	51.743	40,000
200 X 300	295.84	132.172	60,000
300 X 300	680.015	347.014	90,000

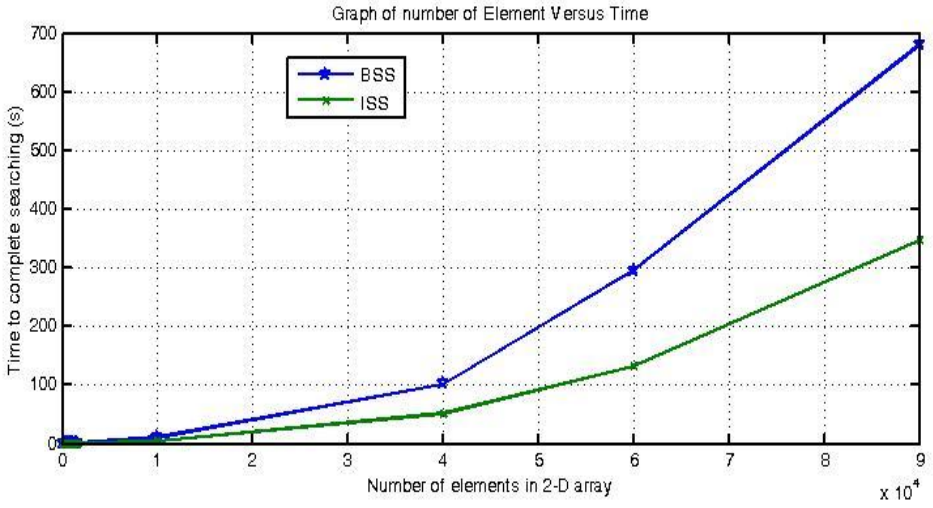

Figure 4: The graph showing the performance of bubble sort based search and insertion sort based search on 2-D array

DISCUSSION

From smaller dimensions to the higher dimensions, insertion sort based search (ISS) requires lesser time than bubble sort based search (BSS). This result is contrary to the asymptotic results where both algorithm is said to have O(n) for best case and $O(n^2)$ for average and worst case (Kavitha, Vijay, & Sakeh, 2016). However, the result agrees with the result obtained in Balogun & Sadiku (2013). Richard & Marcus (2004) gave a convincing explanation of the noticed superiority of insertion sort algorithm over bubble sort algorithm when it was explained that the big Oh give no regard to constant as a result both O(n/2) and O(n/4) are regarded as O(n).

Conclusion

In this study bubble sort, insertion sort and row wise and column wise 2-D array search algorithm are implemented. The performances (in terms of time required to complete a search) of bubble sort based search and insertion sort based search are measured for matrix of different dimensions. The superiority of insertion sort based search over bubble sort based search despite that both have the same asymptotic time complexity is upheld. If there is therefore the need to choose between bubble sort and insertion sort for search of 2-D array element, insertion sort is recommended.

REFERENCES

Aditya, D. M., & Deepak, G. (2008). Selection of Best Sorting Algorithm. *International Journal of Intelligent Information Processing*, 363-368. Retrieved from http://academia.edu/1976253/Selection_if_Best_Sorting_Algorithm

Ahmed, M., & Zirra, P. (2013). A Comparative Analysis of Sorting Algorithm on Integer and Character Arrays. *The International Journal of Engineering and Science (IJES)*, 25-30.

Ashima, G., Meenu, V., & Swati, G. (2016, Jan-April). Implementation and Application of Binary search in 2-D Array. *International Journal of Institutional and Industrial Research, 1*(1), 30-31.

Balogun, B., & Sadiku, J. (2013, August). Simulating Binary Search Technique Using Different Sorting Algorithms. *International Journal of Applied Science and Technology, 3*(6), 67-75. Retrieved October 1, 2018, from http:// www.ijastnet.com

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms* (Third ed.). McGraw Hill.

Francis, S. (1987). *Theory and Problems of Computers and Programming*. Lagos, Nigeria: Schaum's Outline Series.

Goodrich, M., & Tamassia, R. (2010). *Data Structures and Algorithms in Java* (4th ed.). John wiley & sons.

Harish, R. M. (2014). Runtime Bubble sort - an Enhancement of Bubble Sort. *International Journal of Computer Trends and Technology (IJCTT), 14*(1).

Jehad, H. (2015). A Comparative Study between Various Sorting Algorithms. *International Journal of Computer Science and Network Security, 15* (3).

Karthick, S. (2016, September-October). Odd Even based Binary Search. *International Journal of Computer Engineering & Technology (IJCET), 7*(5), 40-55. Retrieved October 2, 2018, from http://www.iaeme.com/ijcet/issues.asp?JType=JJCET&VType=7&IType=5

Kavitha, S., Vijay, V., & Sakeh, A. (2016). Matrix Sort- A Parallelizable Sorting Algorithm. *Internatioal Journal of Computer Applications, 143*(9), 1-6.

Kazim, A. (2017). A Comparative Study of Well Known Sorting Algorithms. *International Journal of Advanced Research in Computer Science, 8*(1), 277-280.

Khalid, S. A.-K., Ibrahim, M. A., Abdallah, M. I., & Nabeel, I. Z. (2013). Review on Sorting Algorithms A Comparative Study. *International Journal of Computer Science and Security, 7*(3), 120-126.

Parveen, K. (2013, March). Quadratic Search: A new and Fast Search Algorithm (An Extension of Classical Binary Search Strategy). *International Journal of*

Computer Applications, *65*(14), 43-46. Retrieved October 2, 2018

Richard, J., & Marcus, S. (2004). *Algorithms.* USA: Prentice Hall.