Evaluating the Time Efficiency of the Modified Linear Search Algorithm

G. B. Balogun

Abstract: In this study, a comparative analysis of three search algorithms is carried out. Two of the algorithms; the linear and binary are popular search methods, while the third, 'The Bilinear Search Algorithm' is a newly introduced one; A modified linear search algorithm that combines some features in linear and binary search algorithms to produce a better performing one. These three algorithms are simulated using seventeen randomly generated data sets to compare their time efficiency. A c_Sharp (C#) code is used to achieve this by generating some random numbers and implementing their working algorithms. The internal clock of the computer is set to monitor the time durations of their computations. The result shows that the linear search performs better than the binary search. This agrees with the existing assertion that when a data set is small and the time complexity of sorting method added, the linear search performs better than the binary search. The bilinear search however, proves to be the most efficient of them all as its time efficiency at all levels of the data sets proves to be better than both the linear and binary search algorithms.

Keywords: Linear Search, Binary Search, Bilinear, Simulation.

I. INTRODUCTION

The difference between a fast program and a slow one is the use of a good algorithm for the data set (Prelude, 2011). Any imperfections in search algorithms would undoubtedly affect most if not all computer users as users are likely to engage in search activities at one time or the other while using the system. There is therefore a need to know which search techniques should or should not be used in data processing to minimize the effects of their shortcomings on the output.

Brian, (2017) tried to display the advantages of binary search over linear search. He noted that the more the elements present in the search array, the faster a binary search will be (on average) compared to a linear search. The downside, he continued, is that binary search only operates on a sorted array, which means the data must be pre-sorted using some means. However, some other factors such as the data size can determine the choice of the search technique used (Dalal, 2004).

Furthermore, John Morris, (1998) affirmed that the binary and linear search algorithms as with others are not without their shortcomings. The binary search algorithm which is believed to be a very efficient algorithm (Shield, 1983) requires that the array elements be sorted using
any of the sorting algorithms. Its efficiency therefore depends on the sorting algorithm used.

The linear search on the other hand, neither requires a sorted data to operate nor any special care to write its codes as they are straight forward and relatively simple, but that is not to say it is without its own disadvantages (Trims, 2014); it is inefficient when the array being searched contains large number of elements.

The algorithm will have to look through all the elements in order to find a value in the last element. The average probability of finding an item at the beginning of the array is almost the same as finding it at the end. (Thomas and Devin, 2017).

Nell, Daniel and Chip,(2016), tried to improve on the performance of the linear search by stopping a search when an element larger than the target element is encountered, but this is based on the condition that the elements in the array be sorted in ascending order. The disadvantage of this method is that, it is also dependent on sorting just like in the binary search. Its efficiency would be dependent on the sorting algorithm used for its implementation.

Manuscript received 2 August, 2017.
Balogun Ghaniyyat Bolanle is a lecturer in the Department of Computer Science, University of Ilorin, Kwara State, Nigeria.

II. RESEARCH MOTIVATION AND METHODOLOGY

The need to develop a search algorithm that would be both independent of sorting and efficient brought about the development of the 'bilinear search' algorithm. The 'bilinear search' algorithm is an improved/modified form of the linear search algorithm. For this method, the array is broken and searched in a multi-directional manner. The array need not be sorted to use the bilinear search. The method operates on the idea that when a search is conducted on the raw/unsorted data it should not be conducted in a one directional (top-down) or step by step manner. Rather the search should be conducted in a multi directional manner (top-down, down-top, top-center, center-top, center-down, down-center and so on). That is the search is spread out and comes from different directions. This method increases the probability/chances of finding the search item faster than in the case of the one directional linear search. This search method also allocates position in the form of sectors for the items in the array thereby providing some form of information as to the location of the required item in question. The use of this algorithm to search for a specified element in an array at one stretch is referred to as 'single bilinear search'. Dividing the array into two sectors is referred to as the 'double bilinear'. Dividing into three it is called the 'triple bilinear search'. Dividing the elements into four sectors is called 'quadruple bilinear search' and dividing the elements into five sectors to search for some specified elements, is called 'quintuple bilinear search'.

In this study, a comparative analysis of the bilinear search algorithm with the linear and binary search algorithms when subjected to some experimental platforms is conducted. A c_Sharp (C#) code is developed. The code generates some random numbers, implements the working algorithms of the bilinear searches, the linear search, and the binary search algorithms. It sorts the random numbers using bubble sort, insertion sort, and quick sort techniques before applying the binary search algorithm. The internal clock of the computer is set to monitor the time durations of the computations.

The linear search and binary search together with the single bilinear search, double bilinear search, triple bilinear search, quadruple bilinear search and quintuple bilinear search methods are simulated using seventeen different data sets. The data are all numeral and generated randomly.
The sizes of the data sets used in the simulation are 50, 100, 200, 500, 1000, 2000, 5000, 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000 and 100,000
In summary, the C# code does the following:

- Generate random numbers for the 17 data sets listed above
- Implements the five levels of the bilinear search algorithms
- Implement bubble sort
- Implement insertion sort
- Implement quick sort
- Implement linear search
- Implement binary search

The code also sets the system clock to know the duration of time taken by different computations.
The following computations are done and the time used on them by the computer noted.

- BinBS = Bubble sort based binary search
- BinQS = Quick sort based binary search
- BinIS = Insertion sort based binary search
- Lin = Linear search only
- Single BL = Single Bilinear
- Double BL = Double Bilinear
- Triple BL = Triple Bilinear
- Quadruple BL = Quadruple Bilinear
- Quintuple BL = Quintuple Bilinear

The outcome of the study was to determine whether to recommend one of the existing algorithms or opt for the newly introduced one.
Below is the C# code for bubble sort based binary search, quick sort based binary search, insertion sort based binary search and the linear search:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace TestCompareNew
{class Program
   { static void Main(string[] args)
      {Stopwatch s = new Stopwatch();
         int MaxData = 100000;
Repeat: Console.Clear();
int[] DataSet = { 50, 100, 200, 500, 1000, 2000, 5000, 10000,
20000, 30000, 40000, 50000, 60000, 70000, 80000,
90000,MaxData };
Random random = new Random();
Console.WriteLine("|-------|---------------|----|---------------|------------
---|");
Console.WriteLine("|-------|---------------|--This will return only
time spent in millisecond--|---------------|---------------|");
Console.WriteLine("|-------|-----------------------|-----------------------
|-----------------------|-----------------------|");
Console.WriteLine("|Set\t|\tBinBS\t\t|\tBinQS\t\t|\tBinIS\t\t|\tLin\t\t
|");
Console.Write("|-------|-----------------------|-----------------------|------
-----------------|-----------------------|");
int[] NumArray = new int[MaxData];
NumArray = RandomNumber(MaxData);
for (int i = 0; i < DataSet.Length; i++)
{
int ind = DataSet[i] - 1;
/////////////Bubble sort and Binary Search starts/////////
 int[] NumArray1 = PickRandomNumber(NumArray, 0,
DataSet[i]);
            s.Start();

            int[] ArrayNum = BubbleSort(NumArray1);
            int SearchedVal = BinarySearch(ArrayNum,
NumArray1[ind]);
s.Stop();
Console.Write("\n|" + DataSet[i]);
Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds+"\t");

/////////////Bubble sort and Binary Search ends/////////

/////////////Quick sort and Binary Search starts/////////
int[] NumArray2 = PickRandomNumber(NumArray, 0,
DataSet[i]);
s.Restart();
ArrayNum = quickSort(NumArray2, 0, NumArray2.Length - 1);
SearchedVal = BinarySearch(ArrayNum, NumArray2[ind]);
            s.Stop();
Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds + "\t");
/////////////Quick sort and Binary Search ends/////////
/////////////Insertion sort and Binary Search starts/////////
int[] NumArray3 = PickRandomNumber(NumArray, 0,
DataSet[i]);
s.Restart();
ArrayNum = InsertSort(NumArray3);
SearchedVal = BinarySearch(ArrayNum, NumArray3[ind]);
s.Stop();
/////////////Insertion sort and Binary Search ends/////////
Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds + "\t");
/////////////Linear Search starts/////////
int[] NumArray4 = PickRandomNumber(NumArray, 0,
DataSet[i]);
s.Restart();
SearchedVal = LinearSearch(NumArray4, NumArray4[ind]);
s.Stop();
Console.WriteLine("\t|\t" + s.Elapsed.TotalMilliseconds + "\t\t|");
Console.Write("|-------|-----------------------|-----------------------|------
-----------------|-----------------------|");
/////////////Linear Search ends/////////
         }
      Console.WriteLine("\nDo you want to repeat the
simulation? 1 -- Yes; 0 -- No ");
      string resp = Console.ReadLine();
      if (resp == "1")
      {
         goto Repeat;
      }
      else if (resp == "0")
      {
         Console.WriteLine("Good Bye");
         Console.ReadLine();
      }
   }
   public static int[] RandomNumber(int ListNum)
   {
      Random randomNew = new Random();
      int[] Number = new int[ListNum];
 for (int i = 0; i < ListNum; i++)
      {
            int num = randomNew.Next();
            Number[i] = num;
        }
        return Number;
    }

    public static int[] PickRandomNumber(int[] FromArray, int
init, int ListNum)
    {
        Random randomNew = new Random();
        int[] Number = new int[ListNum];
        int icount = 0;
        for (int i = init; i < ListNum; i++)
        {
            Number[icount] = FromArray[i];
            icount++;
        }
        return Number;
    }

    //////// BubbleSort
    public static int[] BubbleSort(int[] array)
    {
        int temp;
        int j;
        int i = array.Length - 1;
        while (i > 0)
        {
            int swap = 0;
            for (j = 0; j < i; j++)
            {
                if (array[j].CompareTo(array[j + 1]) > 0)
                {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swap = j;
                }
            }
            i = swap;
        }
        return array;
    }

    //??/////Insertion Sort
    static int[] InsertSort(int[] array)
    {
        int i, j;

        for (i = 1; i < array.Length; i++)
        {
            int value = array[i];
            j = i - 1;
while ((j >= 0) && (array[j].CompareTo(value) > 0))
            {
                array[j + 1] = array[j];

                j--;
            }
            array[j + 1] = value;
        }
        return array;
    }
    //////// QuickSort
    private static void quickSwap(int[] Array, int Left, int Right)
    {
        int Temp = Array[Right];
        Array[Right] = Array[Left];
        Array[Left] = Temp;
    }
    public static int[] quickSort(int[] Array, int Left, int Right)
    {
        int LHold = Left;
        int RHold = Right;
        Random ObjRan = new Random();
        int Pivot = ObjRan.Next(Left, Right);
        quickSwap(Array, Pivot, Left);
        Pivot = Left;
        Left++;

        while (Right >= Left)
        {
            int cmpLeftVal = Array[Left].CompareTo(Array[Pivot]);
            int             cmpRightVal              =
Array[Right].CompareTo(Array[Pivot]);

            if ((cmpLeftVal >= 0) && (cmpRightVal < 0))
            {
                quickSwap(Array, Left, Right);
            }
            else
            {
                if (cmpLeftVal >= 0)
                {
                    Right--;
                }
                else
                {
                    if (cmpRightVal < 0)
                    {
                        Left++;
                    }
                    else
                    {
                        Right--;
                        Left++;
                    }
        quickSwap(Array, Pivot, Right);
        Pivot = Right;
        if (Pivot > LHold)
        {
            quickSort(Array, LHold, Pivot);
        }
        if (RHold > Pivot + 1)
        {
            quickSort(Array, Pivot + 1, RHold);
        }
        return Array;
    }
    public static int BinarySearch(int[] array, int value)
    {
        int low = 0, high = array.Length - 1, midpoint = 0;

        while (low <= high)
        {
            midpoint = low + (high - low) / 2;

            // check to see if value is equal to item in array
            if (value == array[midpoint])
            {
                return midpoint;
            }
            else if (value < array[midpoint])
                high = midpoint - 1;
            else
                low = midpoint + 1;
        }
        // item was not found
        return -1;
    }
```

```
        public static int LinearSearch(int[] array, int item)
        {
            int searchItem = item;
            int len = array.Length;
            for (int j = 0; j < len; j++)
            {
                if (array[j] == searchItem)
                {
                    return j;
                }
                if (j == len - 1)
                {
                    return -1;
                }
            }
            return -1;
        }
```

C# Program used in the implementation of the bilinear search algorithm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace LinerSearch
{
    class BiLinear
    {
        static void Main(string[] args)
        {


            //Console.WriteLine("Enter the number of elements you
want to add in the array ?");
           // string s =Console.ReadLine();
           // int x = Int32.Parse(s);

            //no of array elements
            int[] x_array = { 50, 100, 200, 500, 1000, 2000, 5000,
10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000,
100000, 200000, 300000, 400000, 500000, 600000, 700000,
800000, 900000, 1000000 };

            //peform the iteration 15x with different no of array size

            Console.WriteLine("======================    Time
in Miliseconds ========================");
            Console.WriteLine("Array Size "+ "\t"+"Linear: " +"\t" +
"Bi-Linear: "+ "\t" + "Double Bi-Linear: " );
            Console.WriteLine("----------------------------------------------
--------------------------------");
            for (int k = 0; k < x_array.Length; k++)
            {
                Random rdmn = new Random();

                //// Console.WriteLine("Enter the Search element\n");
                // string s3 =Console.ReadLine();
                // int x2 = Int32.Parse(s3);

                int x2 = rdmn.Next(500);
                int x = x_array[k];
                int[] a = new int[x];
                int[] anew = new int[x];
                int[] bnew = new int[x];

                a = RandomNumber(x);
                //for the first partion
                for (int j = 0; j < (x / 2); j++)
                {
                    anew[j] = a[j];
                }

                //for the second partion
                for (int j = (x / 2); j < x; j++)
                {
                    bnew[j] = a[j];
                }

                //search using linear method
                var watch = Stopwatch.StartNew();
                Linear ln = new Linear();
                ln.findLinear(a, x2, x);
                watch.Stop();
                var l_elapsedMs = watch.ElapsedMilliseconds;

                //Search begins with bi linear method
                watch = Stopwatch.StartNew();
                Bi_linearClass nn = new Bi_linearClass();
                nn.findbiliner(x, x2, a, anew, bnew);
                //end search
                //Calculate time taken
                watch.Stop();
                var bil_elapsedMs = watch.ElapsedMilliseconds;

                //Search begins with double bi linear method
                watch = Stopwatch.StartNew();
                Double_bilinear dbli = new Double_bilinear();
                dbli.find_double_biliner(x, x2, a, anew, bnew);
                //end search
                //Calculate time taken
                watch.Stop();
                var dbl_elapsedMs = watch.ElapsedMilliseconds;

               Console.WriteLine(x+"\t        " + l_elapsedMs + "\t        "
+ " " + bil_elapsedMs + "\t        " + "       " + dbl_elapsedMs);
               Console.WriteLine("--------------------------------------------
-----------------------------------");
            }

          Console.ReadLine();
        }

        public static int[] RandomNumber(int ListNum)
        {
            Random randomNew = new Random();
            int[] Number = new int[ListNum];
           // Console.WriteLine("Generated Numbers below: ");
            for (int i = 0; i < ListNum; i++)
            {
                int num = randomNew.Next(500);
                Number[i] = num;
                //Console.Write(num+",  " );
            }
            return Number;
```

III. RESULTS

The performance of the search algorithms is presented in Table I. It is also summarised diagrammatically in the diagrams below. The time values (Millisecond) of the search algorithms are embedded in a clockwise manner in the diagrams.

Table 1: Time Spent in Millisecond for the Execution of 50 to 100,000 Data Set

Set	BinBS	BinQS	BinIS	Lin	Single BL	Double BL	Triple BL	Quadruple BL	Quintuple BL
50	0.9674	1.6193	0.5427	0.3727	0.003	0.0036	0.0031	0.0031	0.003
100	0.4997	0.6948	0.0538	0.0032	0.0031	0.003	0.003	0.0033	0.003
200	0.5023	1.5506	0.1751	0.0051	0.0045	0.0039	0.0038	0.0036	0.0035
500	4.0547	3.8436	1.072	0.0083	0.0063	0.0041	0.0046	0.0041	0.0039
1000	12.6108	8.0684	4.4294	0.0147	0.0108	0.0087	0.0088	0.008	0.0079
2000	50.6395	16.099	17.1159	0.0256	0.0191	0.0121	0.0132	0.011	0.0108
5000	321.5405	37.1292	131.5038	0.0603	0.0421	0.0301	0.0291	0.0298	0.029
10000	1285.501	71.8507	436.5314	0.1174	0.0961	0.0951	0.0751	0.0801	0.0742
20000	5150.955	168.4701	1741.123	0.2328	0.2003	0.174	0.1759	0.1603	0.1602
30000	11638.39	226.806	4145.625	0.6781	0.3671	0.3123	0.3182	0.3137	0.4001
40000	20414.13	292.9878	6933.484	0.4709	0.462	0.4001	0.3901	0.3854	0.4012
50000	31203.49	372.8973	10917.26	0.5787	0.4778	0.3982	0.4217	0.4115	0.4215
60000	45536.24	468.4171	16241.34	0.6961	0.5183	0.518	0.4687	0.4821	0.4827
70000	63380.1	505.5233	21143.43	0.818	0.6249	0.6318	0.6219	0.6365	0.6255
80000	82410.09	622.7206	29061.69	0.9232	0.5437	0.5563	0.5321	0.5612	0.5327
90000	112526.9	700.1806	36341.18	1.0643	0.6781	0.6417	0.6511	0.6153	0.6104
100000	126516.8	774.3558	45896.85	1.158	0.9721	0.7543	0.8256	0.8129	0.8116

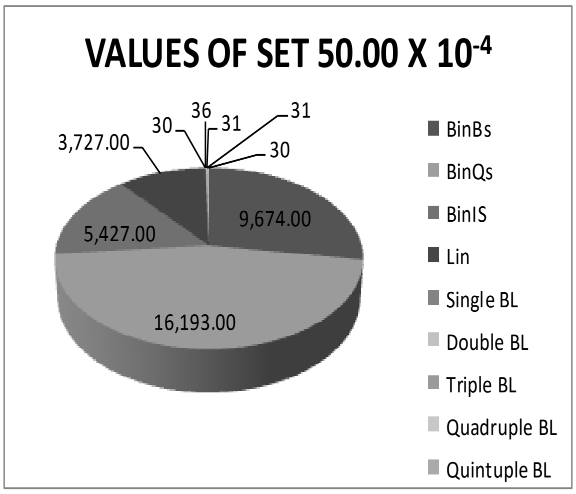

Figure 1: Pictorial analysis of the Search Algorithms used on Data Set 50.

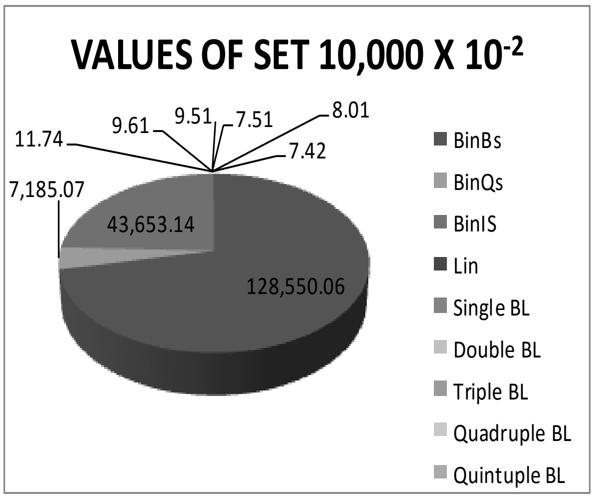

Figure 3: Pictorial Analysis of the Search Algorithms used on Data Set 10,000

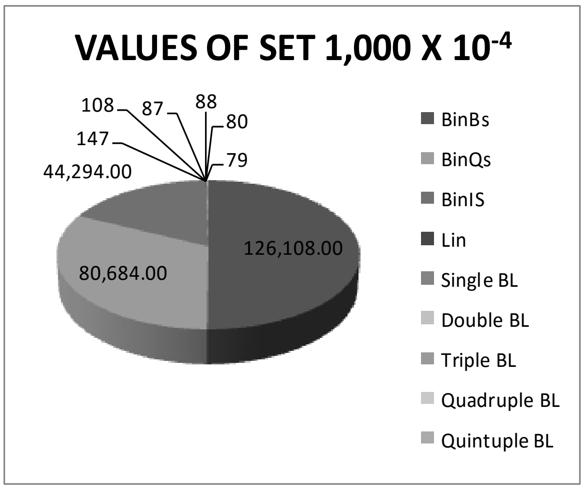

Figure 2: Pictorial analysis of the Search Algorithms used on Data Set 1,000.

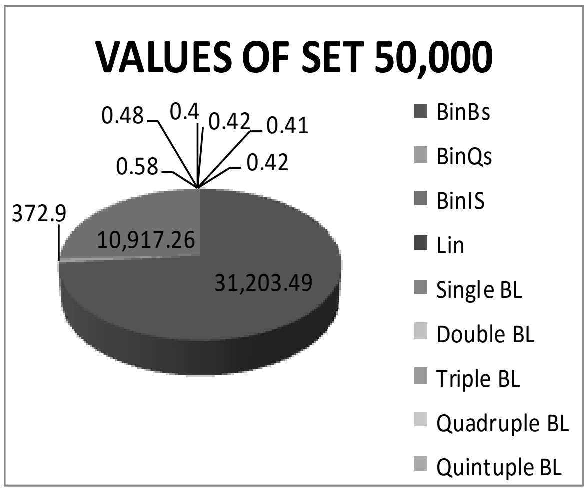

Figure 4: Pictorial Analysis of the Search Algorithms used on Data Set 50,000.

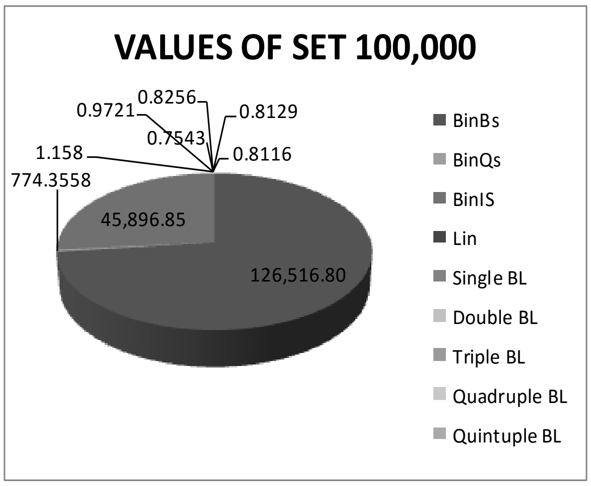

Figure 5: Pictorial Analysis of the Search Algorithms used on Data Set 100,000

IV. ANALYSIS OF THE RESULTS

From figure 1, it is observed, that for the data set 50, both the single bilinear search and the quintuple bilinear search algorithms return as the most efficient algorithm in terms of time complexity. The other bilinear searches also prove to be more efficient than the linear and binary searches. The quick sort based binary search is found to be the least efficient. This is to be expected as the quicksort is known to be inefficient where small data are concerned (Shield, 1983).

As the data set increases from 50 to 10,000, so does the performance of the quicksort based binary search improve. But better as the quicksort based binary sort may get, the linear search still proves to be more efficient at this stage. All the levels of the bilinear searches however, prove to be the most efficient of them all. There are slight fluctuations in the level of efficiency among the bilinear searches in this interval of 50 to 10,000, but the quintuple bilinear search still maintains its position as the most efficient.

The inefficiency of the bubble sort explodes as the data set increases from 10,000 to 50,000 and then gets worse as the data set increases to 100,000. This is to be expected as bubble sort has been proved to be efficient only for small data sets. The insertion sort though also efficient for only small data set, proves to be more efficient than the bubble sort at all stages of the data sets except for a single data set; 100.

The efficiency of the quick sort based binary search improves as the data set moves to 100,000. This is also expected as the larger the data, the more efficient the quick sort. Therefore at this stage, the quicksort based binary search performs better than the bubble sort based binary search and the insertion sort based binary search.

The linear search, at the 100,000 data set, still proves to be overwhelmingly more efficient than the entire sort based binary searches. This is because of the fact that when the data to be searched is not very large, and the sorting complexity is added to that of the binary search complexity, the linear search proves to be more efficient.

For all the levels of the bilinear searches however, the results indicate that they are more efficient than all the other search methods even though there are slight fluctuations in the level of efficiency amongst them as the data set increases from 50 to 100,000. With the larger share of efficiency going to the quintuple bilinear search algorithm.

V. SUMMARY OF FINDINGS

The result confirms the fact that the so called acclaimed efficiency of the binary search algorithm cannot be generalised. Its dependence on the type of sorting method used in its implementation and the size of the data to be searched makes its efficiency vulnerable (Balogun and Sadiku, 2013). If the statuses of these factors are not properly verified before employing the binary search method, the search may become inefficient. Variations in the level of efficiency of the binary search using different sorting algorithms can be seen as the data set is gradually increased from 50 to 100,000.

The bubble and insertion based binary searches performed better at the lower data set stage than the quicksort based binary search. But as the data set increases to a larger size, the bubble sort based binary search collapses, that of the insertion based binary search drags on slowly while that of the quicksort based binary search gains momentum to become the better performing option. But overall, the linear search performed better than the sort based binary searches. This corroborates the idea that linear search outperforms the binary search if sorting time required in binary search is taken into consideration.(Glenn, 2007).

Introducing and simulating the bilinear search algorithm which is a modified linear search algorithm along with the linear and binary search algorithms, the result obtained shows that this modified linear search algorithm outperforms both the linear and binary search algorithms. This can be associated to the fact that the search algorithm combines some good features found in both the linear and binary searches, to consequently make it a better performing search algorithm.

VI. CONCLUSION

Most research works conducted on the efficiency of linear and binary search algorithms revolve around the comparison of one search algorithm against the other (Nell et al., 2016), while trying to establish which of the two algorithms come out tops as the better performing one, thus, the discrepancy among some researchers as to which algorithm performs better than the other. Balogun and Sadiku, (2013) however, demonstrated that none of the two; linear or binary search algorithms can be said to outperform the other as their level of efficiency is determined by the size of the data set and the efficiency of the sorting algorithms used in the implementation of the binary search.

The ‘bilinear search algorithm’ which is a modified linear search that does not search from only one direction and can allocate some form of address for the data in the form of sector location, comes as a solution for having a search method that is not only independent of sorting but at the same time performs a search effectively well.

REFERENCES

[1] Balogun B.G. and Sadiku J.S. (2013). Simulating Binary Search Technique Using Different Sorting Algorithms. International Journal Of Applied Science And Technology. Volume 3, No 6 pp. 67-75.August, 2013.
[2] Brian. F. (2016). Computer Programming, C Programming. What are the Advantages of binary search on linear search in c? Copyright © answers.com/Q.
[3] Dalal. A.C. (2004). Searching and Sorting Algorithms. Supplementary Lecture Notes. Copyright © cs.carleton.edu/faculty.

[4] Glenn J. (2007): Computer Science; an Overview Tenth Edition, Pears on Education, New Jersey, U.S.A.
[5] John. M. (1998). Data Structures and Algorithms. Copyright © John Morris j.morris@auckland.ac.nz (1998).
[6] Nell D, Daniel T., Chip (2016). Weems Object-Oriented Data Structures Using Java Copyright © Jones & Bartlett Learning.
[7] Prelude, (2011). Linear Search, Binary Search and other Searching Techniques. Copyright © 1997-2011 Cprogramming.com. All rights reserved.
[8] Shield. F (1983): Theory and Problems of Computers and Programming, Schaum's Outline Series.
[9] Thomas. C and Devin. B (2017). Binary Search. Khan Academy computing curriculum team. Copyright © CC-BY-NC-SA.
[10] Trims. (2014). What are advantages and disadvantages of linear search? Java Programming, C Programming. Eishet Eilon Industries.