# A Comparative Analysis of the Efficiencies of Binary and Linear Search Algorithms

Ghaniyyat Bolanle Balogun
Department of Computer Science,
University of Ilorin, Ilorin,
Nigeria.
*Email: balogun.gb@unilorin.edu.ng*

**ABSTRACT**

***The binary search algorithm is one of the popular search methods for pragmatic purposes. Its efficiency is derived from its technique of 'divide and conquer'. The linear search, on the other hand is a search algorithm that has to go through the entire search to conclude a search. The general view therefore, is that, the binary search is more efficient than the linear search especially when dealing with large data set. This assertion has however been challenged by some researchers based on the fact that the efficiency implicit in the binary search assumes that the items to be searched have been sorted, which may not be true in all cases. Their study shows that when sorting time is added, the linear search proves to be more efficient than the binary search. This conclusion however, could also be said to be hasty as it did not take into cognizance the variations in the number of times a search is to be conducted on a specific data set as the sorting need not be repeated. This led to the aim of this work which is to examine the efficiency of the two search methods; the linear and binary search with sorting time added, when a search is conducted numerous times on a specific data set. The objective is to derive an average sorting time and add it to each search as the search is gradually increased from the very 1st to the 100,000th search. The methodology used in obtaining this is by dividing the sorting time (QS) by the number of searches (N) conducted. The binary search only, binary search with quick sort and linear search algorithms were implemented using the object oriented c_Sharp (C#). The result shows that despite the argument that the linear search is more efficient than the binary search when sorting time is considered, the binary search will eventually recover to prove itself as the more efficient method when a search is conducted multiple times on the same data set.***

**Keywords**: *Sorting, Linear Search, Binary Search, Data.*

**African Journal of Computing & ICT Reference Format:**

Ghaniyyat Bolanle Balogun (2020), A Comparative Analysis of the Efficiencies of Binary and Linear Search Algorithms, *Afr. J. Comp. & ICT*, Vol.13, No. 1, pp. 39 - 51.

© Afr. J. Comp. & ICT, March 2020; P-ISSN 2006-1781

## I. INTRODUCTION

Information retrieval is the most fundamental requirement for any kind of computing application which requires search operation to be performed on massive databases implemented by various data structures. Searching an element from the list is a fundamental aspect in computing world. Numbers of algorithms have been developed for searching an element among which linear search and binary search are the most popular algorithms [21]. Many of the tasks of computer science in general, and artificial intelligence in particular can be phrased in terms of a search for the solution to the problem at hand. Indeed, basic search techniques provide the key to many historically important accomplishments in the area of artificial intelligence [15].

The difference between an efficient program and an inefficient one is the use of a good algorithm for the data set [22]. 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 [6].

An important property of most of the significant search problems studied in artificial intelligence is that they suffer from combinatorial explosion. That is, the number of states which must be searched generally grows very rapidly with the size and complexity of the system studied. Various strategies for effective search have emerged; some of them are; binary search, breadth – first search, depth – first search, hill – climbing heuristic, best – first heuristic and linear search [15].

The choice of search method depends on the amount of data, data type and data structure used [23]. In this study the binary and linear search techniques are considered. These two techniques belong to the class of fundamental algorithms in data structures. But these search algorithms, as with most others, are not without their shortcomings; the binary search algorithm, which is believed to be a very efficient algorithm, requires that the array elements be sorted using any of the sorting algorithms. Writing code for binary search also requires care to ensure that all the special cases are accurately implemented. If there is an error in the code used for the implementation of binary search algorithm, then the search itself becomes ineffective [5].

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; 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 [6].

[3] conducted an experiment, whereby the time complexity of the bubble sort was added to that of the binary search. They made a comparison of the execution time of finding an integer in an array of integers using linear and binary search algorithms. They concluded in their report that if the sorting complexity is added to that of the binary search, then the binary search cannot be said to be more efficient in terms of time complexity. They judged that some existing research works came to a hasty conclusion by stating that the binary search is more efficient without considering the time complexity of the sorting method used in the execution of the search algorithm.

[5] took the research further by considering three sorting techniques; namely: bubble sort, insertion sort and quick sort in the implementation of binary search technique. The three sorting methods were applied one at a time on various sets of data before applying binary search technique. A C# code was developed. The code generates the required random numbers, it also sorts them using bubble sort, insertion sort, and quick sort techniques before applying the binary search algorithm.

The system clock was set to know the duration of time taken by the following computations [5].

| | | |
|---|---|---|
| BinBS | = | Bubble sort based binary search |
| BinQS | = | Quick sort based binary search |
| BinIS | = | Insertion sort based binary search |
| BS | = | Binary search only |
| Lin | = | Linear search only |

The authors demonstrated that the subsisting assertion that quick sort is superior to both the bubble sort and insertion sort, stands. They also noted that the insertion sort is more efficient than bubble sort. However, irrespective of the efficiency of one sort method over the other, the authors concluded that linear search may be preferred if the data items to be searched have not been sorted. They believe that the efficiency implicit in the binary search assumes that the items to be searched have been sorted which may not be true in all cases. Finally, they recommended that if there is need to sort data before being searched, then both bubble sort and insertion sort methods should be used for small data sets while quick sort should be used otherwise. They concluded that when sorting time is added, the linear search is more time efficient than the binary search [5].

What these authors did not consider however, is that, when a particular data set is to be searched repeatedly, these assertions may be challenged as a specific data needs not be sorted more than once to carry out as many search incidents on it as possible. This study therefore investigates the average sorting time 'AQS' added to a search from the very first search to the 100, 000th search to identify the impact of sorting time on their efficiency.

## II. LITERATURE REVIEW

Searching and sorting are two fundamental operations in computer science [14]. Sorting refers to arranging data in a particular format. That is, sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order. The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats [11].

### *Bubble Sort*

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are; $O(n*(n-1)) = O(n^2)$ (ignoring the constant), where n, is the number of items [10].

If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on. If we have total n elements, then we need to repeat this process for n-1 times.

The sort algorithm is known as bubble sort, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.

Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required [8].

Sample Algorithm for the Bubble sort on an array [n] [8].

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for

  return list

end BubbleSort
```

***Insertion Sort***

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is also not suitable for large data sets as its average and worst case complexity are $O(n(n-1)/2) = O(n^2)$ (ignoring the constant), where n, is the number of items [13].

Sample Algorithm for the Insertion sort on an array [n] [13].

```
Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the
          value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted
```

***Quick Sort***

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quick sort partitions an array and then calls itself recursively twice to sort the two resulting sub arrays. This algorithm is quite efficient for large-sized data sets as its average and worst case complexity are of $O(n^2)$, where n is the number of items [12].

Sample Algorithm for the Quick sort on an array [n] [12]

```
Step 1 − Choose the highest index value as pivot
Step 2 − Take two variables to point left and right of the list excluding pivot
Step 3 − left points to the low index
Step 4 − right points to the high
Step 5 − while value at left is less than pivot move right
Step 6 − while value at right is greater than pivot move left
Step 7 − if both step 5 and step 6 are not matched swap left and right
Step 8 − if left ≥ right, the point where they met is new pivot
```

Given the vast amount of data available, search algorithms are an inevitable part of programming. Searching broadly refers to the process of finding an item with specific properties in the given data set [2]. Searching is a process whereby data kept on the computer system are retrieved according to some search criterion. Thus, the proper storage of data to enable fast searching is an essential issue in data processing. A search typically answers either true or false as to whether the item is present. On occasion it may be modified to return where the item is found [17].

The two most used algorithms for searching are linear and binary search [16]. The linear search is an algorithm that neither requires a sorted data to operate nor any special care to write its codes. This is because they are straight forward and relatively simple. But that is not to say it is without its own disadvantages; 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, its O notation is of order n; O(n) [6].

Sample Algorithm for the Linear Search on an array [A] [30].

```
Step 1: Set i=1
Step 2: if i > n then goto step 7
Step 3: if A[i] = x then goto step 6
Step 4: Set i to i+1
Step 5: Goto Step 2
Step 6: Print x found at index i and goto step 8
Step 7: Print element not found
Step 8: Exit
```

Binary search on the other hand has been found to be very efficient searching technique for sorted data. The efficiency, given that the data items have been sorted is O(logn). In this method, the array is divided into two parts and in this way number of comparisons is very less as compared to linear search. The time is also less in binary search as compared to linear search [6]. Binary search works in a way that middle element of the array is found and then the middle element is compared with one part of array. There can be three possibilities to find the required item; middle number can be the number that needs to be found or the number that is less than the middle number or the number that is greater than the middle number. The number of comparisons is at most log(N+1). Binary search is an efficient algorithm when compared to linear search, but the array has to be sorted before carrying out the search [16]. If an ineffective sort algorithm is used for its implementation then the binary search algorithm may also become inefficient. Writing code for the binary search also requires care to ensure that all the special cases are accurately implemented. If there is an error in the code used for the implementation of binary search algorithm, then the search itself becomes ineffective [5].

Sample Algorithm for the Binary Search on an array [A] [5]

```
  Procedure BINSRCH (A, n, x, j)
// given an array A(1:n) of elements in nondecreasing order,//
// n>=o, determine if x is present, and if so, set j such that x
=A(j)//
// else j = o.//
Integer low, high, mid, j, n;
low ← 1;    high ← n
while low <= high do
mid ← [(low + high)/2]

Case
                    : x<A (mid): high ← mid – 1
                    : x<A (mid): low ← mid + 1
                    : else: j ← mid; return
          endcase
          repeat
          j ← 0
endBINSRCH
```

Binary search is an extremely efficient algorithm when compared to linear search. This technique searches data in minimum possible comparisons [18]. However, the necessary and sufficient condition for this searching technique is that the working array must be sorted [26]. That is, the array should be arranged in ascending or descending order [1]. Which means the data must be pre-sorted using some means. Some factors such as the data size can determine the choice of the search technique used [9].

[7] noted 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 requires that the array elements be sorted using any of the sorting algorithms [25]. Its efficiency therefore depends on the sorting algorithm used. This is not the case with the linear search, but as mentioned earlier it also has its own shortcomings. 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 [29].

[28] conducted a research to figure out the advantages and disadvantages of some sorting and searching algorithms based on time and space complexity. Their result corroborates existing assertions that quick sort is productive for large data items and insertion sort is efficient for small list. It also shows that binary search is suitable for mid-sized data items while the hash search is best for large data items. These authors however did not consider the dependency of the binary search algorithm on the efficiency of the sorting algorithm used for its implementation.

[21] also made their contributions on the comparison of linear and binary search algorithm. These researchers made efforts to compare both algorithms to implement on various data structures and to find out a solution to implementing binary search on linked list. They also analyzed the two algorithms to some extent for applicability and execution efficiency. They concluded that the binary search is more efficient than the linear search but insertion of an element is not efficient in binary search as it requires arranged elements in specific order. That is, sorting is still required for the implementation of the binary search which is a major shortcoming of the search algorithm as any inefficiency in the sorting algorithm used for its implementation would be reflected on the performance of the search.

[4] made a comparative analysis of three sorting algorithms namely; quick, merge and insertion sort implemented in three programming languages (C, Java and Python) using execution time. The comparison of the execution time of these three sorting algorithms was based on programming language, data size and algorithm implementation style (Iterative and Recursive). Their result shows The data size, programming language, the implementation algorithm style are factors that affect execution time of any software and the way software is written and the programming language used are both very important determinants of the execution time of software. This is a very important observation but it does not address the efficiency of the linear search algorithm in comparison to that of the binary search algorithm.

Some authors have 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 [20]. 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.

Some other comparisons between these two algorithms include the fact that the time complexity of binary search has O(log2N) while the time complexity of linear search happens to be O(N). The best case scenario for time in linear search c program is for the 1st element, which is **O(1).** In comparison, in case of binary search, the search is for the middle element, which is O(1). Linear searches may be implemented in array and in linked lists. On the other hand, binary searches are not capable of being implemented on linked lists directly [27].

[24] highlighted the advantages and disadvantages of some sorting algorithms, such as; the quick, selection and bubble sort. They tried to deduce which sorting techniques take less time and space for the given inputs. The binary and linear search algorithms are also explained and compared.

[19] studied the linear search and binary search algorithm and made a comparison on the basis of their time complexity. They concluded that the linear search is simpler than the binary search but the binary search is more efficient than the linear search because it takes fewer amounts of comparisons to find target element as compared to linear search. This assertion has however been challenged by some researchers who believe that when sorting time is added to binary search, the linear search will prove to be more time efficient than the binary search.

This research therefore, investigates the time efficiency of the linear and binary search when data needs to be sorted only once and multiple searches carried out on it.

## III. METHODOLOGY

Figure 1 is a flowchart indicating how sorting time is gradually added to binary search to find its time efficiency when compared to linear search.

The flowchart in Figure 1 shows the steps taken to add the average sorting time to binary search before making a comparison with the linear search.

This research investigates whether the efficiency implicit in the linear search as stated by some researchers is maintained when a search is conducted multiple times on a data. The methodology used in achieving this, is to select a specific data set, search it with the linear search and record the execution time 'LIN'. The same data set is then sorted using a sorting algorithm and the execution time used for the sorting 'QS' noted. The sorted data is then subjected to the binary search and the execution time for the search 'BIN' also noted. The total execution time for binary search 'BIN' in this research includes the sorting time 'QS' making it 'BINQS'. However, with all other factors remaining constants, the number of times a search is repeated 'N' is gradually increased. The total sorting time is divided among this number of times the binary search is repeated on the data set. That is; rather than have BIN + QS= BINQS, it would be; BIN + AQS. Where AQS = QS/N. When the search is conducted once; AQS = QS/1 = QS, twice; AQS = QS/2, thrice; AQS = QS/3 and so on.

Take the example of Tables 1.0 and 2.0.

Tables 1.0 and 2.0 [5] shows that the time efficiency of the linear search outperforms that of the binary based quick sort method (BinQS) which was assumed to be the better of the three binary based sorting methods.

Consider Table 2.0, the time in millisecond used to execute the linear search, binary search and binary search with sorting time added for the 100,000 data set is analysed from the very first search to the 100,000th search. The data used for analysis is derived from the table. The sorting time added to each level of the binary searches is derived from the execution time 'QS' divided by the number of searches 'N' carried out. That is, Execution time for Quick sort divided by the number of searches 'N' at each level.

BinQS = Execution time for Binary (Bin) + Quick sort (QS)
BS = Execution time for Binary search only
QS = BinQS – BS = Execution time for the Quick sort
AQS = QS/N. Average Execution time for Quick sort
BinAQS = Execution time for Binary (Bin) + Average Quick sort (AQS)
Lin = Execution time for Linear search only

Computations from three points of the search; the 1st, 1000th and 100,000th search are considered below.

**- 1st Search**
QS = 819.4650
AQS = 819.4650
BS = 0.0032
BinAQS = 819.4682
Lin = 1.67

**- 1000th Search**
QS = 819.4650
AQS = 0.8194650
BS = 0.0032
BinAQS = 0.8227
Lin = 1.67

**- 100,000th Search**
QS = 819.4650
AQS = 0.008194650
BS = 0.0032
BinAQS = 0.0114
Lin = 1.67

## IV. RESULT

The results of this paper are presented in Table 3.0, Table 4.0, Table 5.0 and Table 6.0, as well as in Figure 2 and Figure 3. The results relate to average time used for sorting and time efficiency of search algorithms.

### 4.1 Analysis of Results

From the results obtained in Table 3.0 and Figures 2 to 4, it is observed that, as the number of repeated binary searches conducted on a specific data increases from the very first search to the 10th search, and then to the 1000th search and finally to the 100,000th, so does the impact of the sorting time on the total time efficiency of the binary search decrease from very strong to very weak so much so that it approaches zero. It can be concluded that, as the number 'N' for searches tends to infinity, N $\rightarrow \infty$, so does the average time complexity AQS of the quick sort (or any other sort method) tend to zero AQS $\rightarrow$ 0. In fact, by the 1000th search upwards to the 100,000th search in Table 2.0, the value of AQS drops dramatically, thereby making the binary search overthrow the linear search as the more efficient search algorithm. Based on this, it can be deduced that, the smaller the data set D, the larger the number of times the search N has to be repeated to make the binary search more efficient than the linear search and vice versa. That is, D $\alpha$ 1/N**.** Conclusively, the type of search method that should be used is not only dependent on the size of data set D but also on the number of times the data D is to be searched; N.

## V. CONCLUSION

There are many factors affecting the choice of search methods used in computer science. This choice is usually made based on the performance, efficiency or shortcomings of a search algorithm. Another major factor given consideration is the data size. However, this work has shown that not only is the efficiency of a search method important and not only is the size of data to be used important but the number of times a specific data set would be subjected to a particular search method while other factors remain constant is crucial in making a decision as to the type of search method to be adopted in a particular situation. The linear search may prove to be more efficient than the binary search when sorting time is added to the binary search, but the binary search would eventually recover to prove to be more efficient when the search is conducted multiple times on a specific data set.

## REFERENCES

[1] O. Ancy and P. Chanchal. Binary Search Algorithm. *International Journal of Innovative Research In Technology*. Vol. 1, No.5, pp 800-803, 2014.

[2] R. C. Ankit, M. Rishikesh and M. Tanaya. Modified Binary Search Algorithm. *International Journal of Applied Information Systems* (IJAIS). Vol. 7, No. 2, pp. 37- 40, 2014.

[3] P. O. Asagba, E. O. Osaghae and E. E. Ogheneovo. 'Is binary search technique faster than linear search technique?' *Scientia Africana*, Vol. 9 (2) pp 83-92, 2010.

[4] O. S. Ayodele and B. Oluwade, 'A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms using Three Programming Languages I: Execution Time Analysis', *African Journal of Management Information System*, Vol. 1, Issue 1, pp. 1-18, 2019.

[5] B. G. Balogun and J. S. Sadiku. 'Simulating Binary Search Technique Using Different Sorting Algorithms'. *International Journal of Applied Science and Technology*. Volume 3, No 6, pp. 67-75, 2013.

[6] G. B. Balogun. A Modified Linear Search Algorithm, *African Journal of Computing & ICT*, Vol.12, No.2, pp. 43 – 54, 2019.

[7] M. Brad and R. David. Problem Solving with Algorithms and Data Structures, 2017. https://www.cs.auckland.ac.nz/compsci105s1c/resources/ProblemSolvingwithAlgorithmsandDataStructures *(accessed 2017).*

[8] Bubble Sort Algorithm.https://www.studytonight.com/data-structures/bubble-sort (accessed 2019).

[9] A. C. Dalal. Searching and Sorting Algorithms. Supplementary Lecture Notes, 2004. http://www.cs.carleton.edu/faculty/adalal/teaching/f04/117/notes/searchSort.pdf (accessed 2016).

[10] Data Structure - Bubble Sort Algorithm. https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm (accessed 2019).

[11] Data Structure - Sorting Techniques. https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm (accessed 2019).

[12] Data Structure and Algorithms - Quick Sort. https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm (accessed 2019).

[13] Data Structure and Algorithms Insertion Sort. https://www.tutorialspoint.com/data_structures_algorithms/insertion_sort_algorithm.htm (accessed 2019).

[14] Difference Between Linear Search and Binary Search. https://www.stechies.com/differencebetween-linear- search-binary-search/ (accessed 2019).

[15] M. W. Firebaugh. *Artificial Intelligence; A knowledge–Based Approach*. World Scientific Publishing Co. Ltd, 1992.

[16] G. Harold, Linear Search Vs. Binary Search. https://diffzi.com/linear-search-vs- binarysearch/. (accessed 2019).

[17] K. P. Komlesh and P. Narendra. 'A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure'. *International Journal of Computer Science and Mobile Computing*. IJCSMC. Volume 3, issue 7, pp 751-758, 2014.

[18] A. Mehta, A. Saxena and A. Thana. Review of Comparison of Binary Search and Linear Search. *International Journal of Engineering Sciences & Management Research*, (IJESMR) Vol.2, No.10 pp 85 – 88, 2015.

[19] J. Morrice and K. Palak. 'Comparison Between Linear Search and Binary Search Algorithms', KITE/NCISRDC/IJARIIT/2018/CSE/107, pp. 63 – 66, 2018.

[20] N. Dale, D. T. Joyce and C. Weems. Object-Oriented Data Structures Using Java. Jones & Bartlett Learning, 2016.

[21] Vimal P Parmar and C. K. Kumbharana. 'Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List', *International Journal of Computer Applications*, 121(3), pp. 13 – 17, 2016.

[22] Prelude. Linear Search, Binary Search and other Searching Techniques, 2011, http://www.cprogramming.com/discussionarticles/sorting_and_searching.html (accessed2016).

[23] R. Robbi, N. Saiful, R. Mukhlis, A. Siti and P. Windania (2017). Comparison Searching Process of Linear, Binary and Interpolation Algorithm. *International Conference on Information and Communication Technology* (IconICT). J. Phys.: Conf. Ser. 930 012007. pp1-6,2017.

[24] K. Roopa and J. Reshma. 'A Comparative Study of Sorting and Searching Algorithms'. *International Research Journal of Engineering and Technology* (IRJET), Volume 05, Issue 01, pp. 1412 – 1416, 2018.

[25] F. Shield. *Theory and Problems of Computers and Programming, Schaum's Outline Series in Computers*. McGraw-Hill Book Company, 1983.

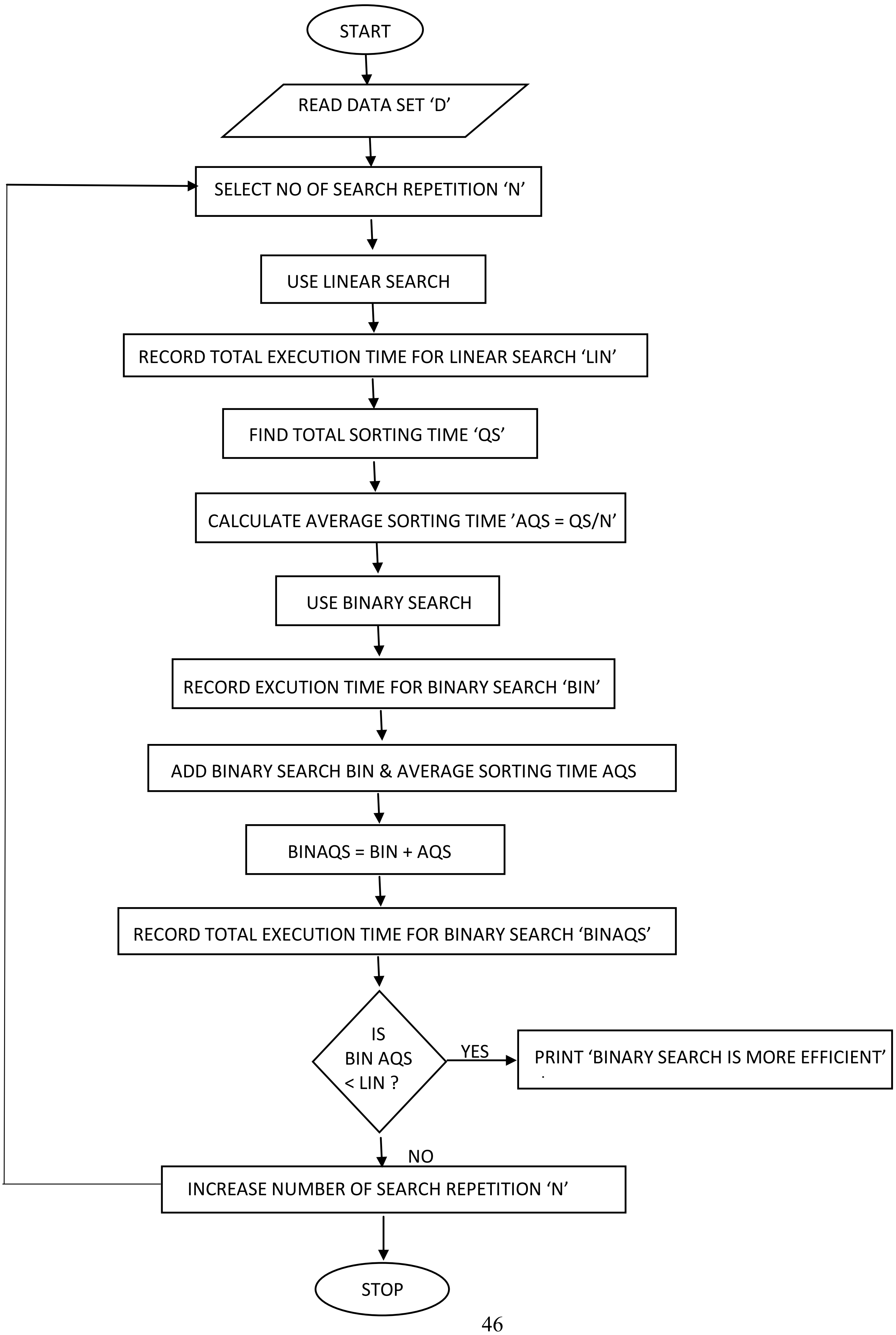
[26] D. Shubham, and M. Kirti. Comparative Performance Analysis of Binary Search in Sequential and Parallel Processing. International Journal for Research in Applied Science & Engineering Technology (IJRASET). Volume 5, Issue X, pp 1-9, 2017.

[27] STechies. (2013) Key Difference between Linear Search and Binary Search with Comparison Chart, 2013. https://www.stechies.com/difference-between-linear-search-binary-search/ (accessed July, 2019).

[28] B. Subbarayudu, L. Lalitha Gayatri, P. Sai Nidhi, P. Ramesh, R. Gangadhar Reddy and C. Kishor Kumar Reddy. 'Comparative Analysis on Sorting and Searching Algorithms'. *International Journal of Civil Engineering and Technology*, Volume 8, Issue 8, pp. 955 – 978, 2017.

[29] C. Thomas and B. Devin. Binary Search. Khan Academy computing curriculum Team, 2017. https://www.khanacademy.org/computing/computerscience/algorithms/binary-search/a/implementing-binary-search-of-an-array (accessed 2018).

[30] Tutorialspoint. Data Structure and Algorithms Linear Search, 2018. http://www.tutorialspoint.com/data_structures_algorithms/linear_search_algorithms.htm (accessed 2018).


START
READ DATA SET 'D'
SELECT NO OF SEARCH REPETITION 'N'
USE LINEAR SEARCH
RECORD TOTAL EXECUTION TIME FOR LINEAR SEARCH 'LIN'
FIND TOTAL SORTING TIME 'QS'
CALCULATE AVERAGE SORTING TIME 'AQS = QS/N'
USE BINARY SEARCH
RECORD EXCUTION TIME FOR BINARY SEARCH 'BIN'
ADD BINARY SEARCH BIN & AVERAGE SORTING TIME AQS
BINAQS = BIN + AQS
RECORD TOTAL EXECUTION TIME FOR BINARY SEARCH 'BINAQS'
IS BIN AQS < LIN ?
YES
PRINT 'BINARY SEARCH IS MORE EFFICIENT'
NO
INCREASE NUMBER OF SEARCH REPETITION 'N'
STOP

ST

Figure 1: Flowchart for Search Comparison.

Table 1.0:Time Spent in Millisecond for the Execution of 50 to 100,000 Data Set [5].

--This will return only time spent in millisecond--

| Set | BinBS | BinQS | BinIS | Lin |
|---|---|---|---|---|
| 50 | 0.9674 | 1.6193 | 0.5427 | 0.3727 |
| 100 | 0.4997 | 0.6948 | 0.0538 | 0.0032 |
| 200 | 0.5023 | 1.5506 | 0.1751 | 0.0051 |
| 500 | 4.0547 | 3.8436 | 1.072 | 0.0083 |
| 1000 | 12.6108 | 8.0684 | 4.4294 | 0.0147 |
| 2000 | 50.6395 | 16.099 | 17.1159 | 0.0256 |
| 5000 | 321.5405 | 37.1292 | 131.5038 | 0.0603 |
| 10000 | 1285.5006 | 71.8507 | 436.5314 | 0.1174 |
| 20000 | 5150.9551 | 168.4701 | 1741.1228 | 0.2328 |
| 30000 | 11638.3872 | 226.806 | 4145.6249 | 0.6781 |
| 40000 | 20414.1276 | 292.9878 | 6933.4839 | 0.4709 |
| 50000 | 31203.4899 | 372.8973 | 10917.2642 | 0.5787 |
| 60000 | 45536.2483 | 468.4171 | 16241.34 | 0.6961 |
| 70000 | 63380.0997 | 505.5233 | 21143.4313 | 0.818 |
| 80000 | 82410.0914 | 622.7206 | 29061.6939 | 0.9232 |
| 90000 | 112526.9173 | 700.1806 | 36341.1778 | 1.0643 |
| 100000 | 126516.8047 | 774.3558 | 45896.8454 | 1.158 |

Table 2.0 Time Spent in Millisecond for the Execution of 100,000 to 400,000 Data Set [3].

|--Return only Quick Sort with Binary Search, Binary Search Only and Linear Search--|

| Set (D) | BinQS | BS | Lin |
|---|---|---|---|
| **100000** | **819.4682** | **0.0032** | **1.67** |
| 200000 | 1637.5814 | 0.0032 | 2.3353 |
| 300000 | 2477.2522 | 0.0057 | 3.5299 |
| 400000 | 3267.3536 | 0.0032 | 4.6533 |

Table 3.0: Average Time used for Sorting with Repeated Binary and Linear Search on the 100,000 Data Set (D).

| No of searches(N) | AQS | BS | BinAQS | Lin |
|---|---|---|---|---|
| 1st search | 819.4650 | 0.0032 | 819.4682 | 1.67 |
| 2nd search | 409.7325 | 0.0032 | 409.7357 | 1.67 |
| 3rd search | 273.1550 | 0.0032 | 273.1582 | 1.67 |
| 4th search | 204.8663 | 0.0032 | 204.8695 | 1.67 |
| 5th search | 163.893 | 0.0032 | 163.8962 | 1.67 |
| 6th search | 136.5775 | 0.0032 | 136.5807 | 1.67 |
| 7th search | 117.0664 | 0.0032 | 117.0696 | 1.67 |
| 8th search | 102.4331 | 0.0032 | 102.4363 | 1.67 |
| 9th search | 91.0517 | 0.0032 | 91.0549 | 1.67 |
| 10th search | 81.9465 | 0.0032 | 81.9497 | 1.67 |
| 1000th search | 0.8195 | 0.0032 | 0.8227 | 1.67 |
| 100,000th search | 0.008195 | 0.0032 | 0.0114 | 1.67 |

Table 4.0: Time Efficiency of the Search Algorithms after the First Search.

| 1st Search | | | |
|---|---|---|---|
| BS | BinAQS | Lin | $X10^{-4}$ |
| 32 | 8194682 | 16700 | |

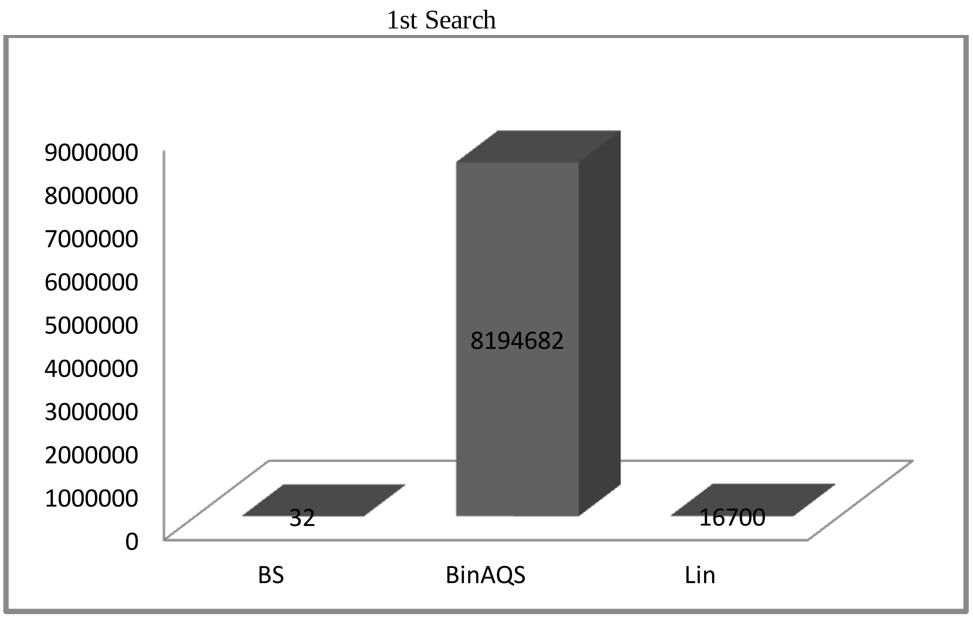


Figure 2: Pictorial Analysis of the Search Algorithms after the First Search.

Table 5.0 : Time Efficiency of the Search Algorithms after the 1,000th Search.

| 1,000th Search | | | |
|---|---|---|---|
| BS | BinAQS | Lin | $X10^{-4}$ |
| 32 | 8227 | 16700 | |

**1,000th Search**

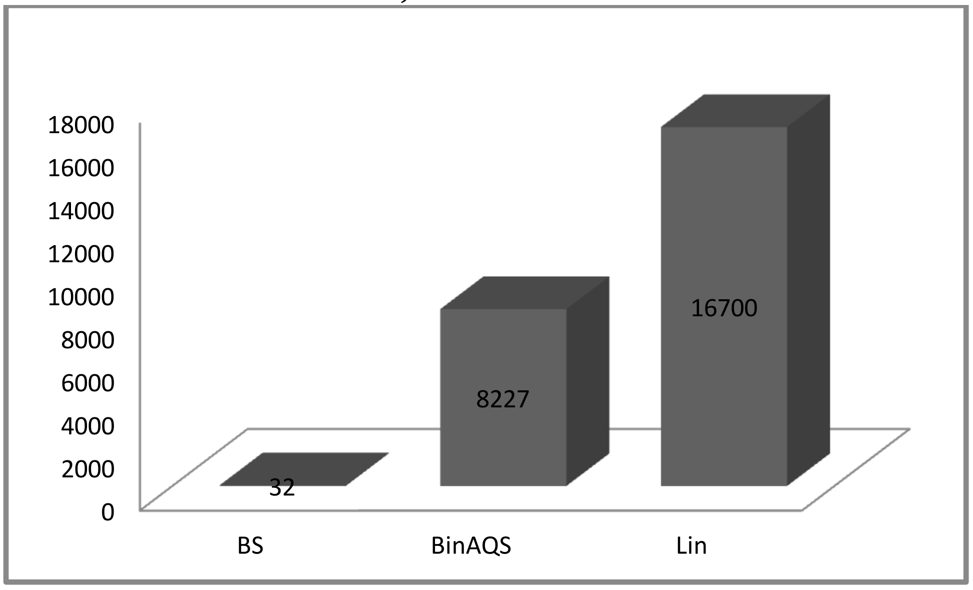


Figure 3 : Pictorial Analysis of the Search Algorithms after the 1,000th Search.

Table 6.0: Time Efficiency of the Search Algorithms after the 100,000th Search.

| 100,000th search | | | |
|---|---|---|---|
| BS | BinAQS | Lin | X10$^{-4}$ |
| 32 | 114 | 16700 | |

100,000th Search

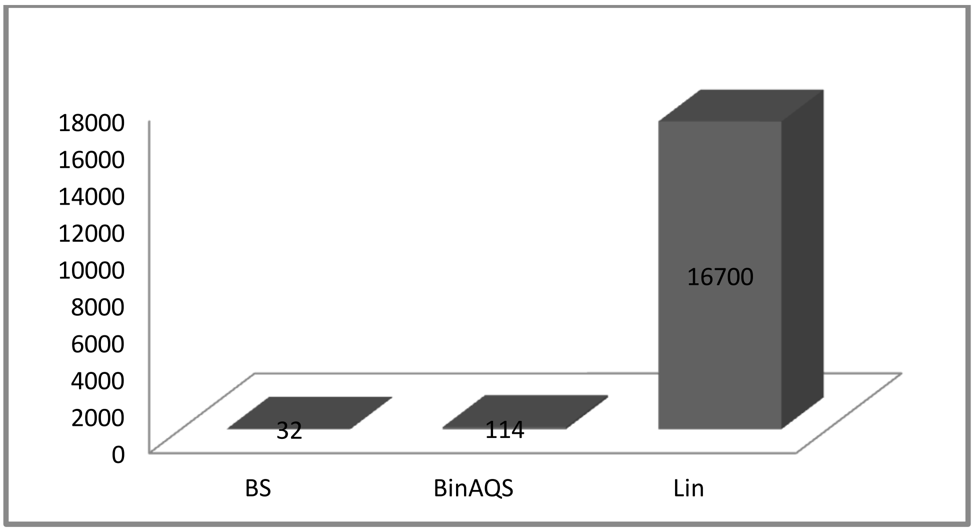


Figure 4: Pictorial Analysis of the Search Algorithms after the 100,000th Search.