© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

A Modified Linear Search Algorithm

G. B. Balogun
Department of Computer Science,
University of Ilorin, Ilorin,
Nigeria.
Email: balogun.gb@unilorin.edu.ng,

ABSTRACT
In this study, a modified linear search algorithm, called the bilinear search is introduced. The aim is to address some of the shortcomings found in the linear search algorithm. For the linear search algorithm, a search may go through an entire array to get a desired result. It conducts the search in a one-directional manner which starts from the very first item to the last one. The efficiency of this search method depends on the size and the number of times a search is to be performed on a specific data set. The bilinear search on the other hand attempts to conduct its search in a multidirectional manner where a search is conducted from different directions in an organised manner. Five forms of the introduced bilinear search algorithm together with the linear search are used to search for some specific numbers in a set of array consisting of 50 elements. The objective is to find the number of steps required to locate each number in their position, thus testing for the effectiveness of the bilinear search when compared with the linear search algorithm. Another objective was to find out if the variation in the number of search directions affects the efficiency of the bilinear search. The result shows that the bilinear search algorithm in its various forms is more efficient than the linear search. It also shows that there are variations in the performance of the bilinear search when the numbers of search directions are varied. It is therefore recommended that the bilinear search algorithm be considered as a reliable search algorithm in data structures.

Keywords: *Array, algorithm, Bilinear, Linear Search, Binary Search*

African Journal of Computing & ICT Reference Format:

G. B. Balogun (2019), A Modified Linear Search Algorithm,
Afr. J. Comp. & ICT, Vol.12, No. 2, pp. 43 - 54.

© Afr. J. Comp. & ICT, June 2019; ISSN 2006-1781

I. INTRODUCTION

Searching and sorting are two fundamental operations in computer science [10]. 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 [8]. Searching for a keyword or value is the basis of many computing applications, whether on an internet search engine or looking up a bank account balance [6]. The difference between a fast program and a slow one is the use of a good algorithm for the data set [13]. 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. Some factors such as the data size can determine the choice of the search technique used, but the disadvantages found in the algorithms still remain; the linear search, an algorithm where every item is checked and if a match is found then that particular item is returned otherwise the search continues till the end of the

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

data collection [17] has the time complexity of order O(n). That is 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 [16].

The binary search algorithm on the other hand, is an algorithm which looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned, if the middle item is greater than the item then the item is searched in the sub-array to the left of the middle item otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the sub-array reduces to zero [17]. This algorithm is believed to be more efficient than the linear search. A major disadvantage of this search technique is that it is dependent on sorting. 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 [2]. This made the study come up with a modified form of linear search called the bilinear search method. An algorithm where its efficiency would not depend on the sorting as in the case of binary search and at the same time would be more efficient than the linear search.

The term 'Bilinear' is used to describe this algorithm because the search is conducted on the elements/sectors from two ends of an array/sectors to finish up in the middle unlike in the case of the linear search where the search is conducted from only one end to finish up at the other end. This search method operates on the idea that when a search is conducted on the data it should not be conducted in a one directional (top-down) or step by step manner as in the linear search. 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). The search is conducted by picking the first item followed by the last item in the array. Where the number of items in the array is of the odd number, the last single item in the middle would be the last to be searched. What makes this modified search algorithm multi directional is the fact that the array can be broken into sectors and each sector searched from two ends. That is; the first sector is searched and then the last. The second sector and then the second to the last and so on. Hence, conducting the search from different points in a single array. The array need not be sorted to use the bilinear search. That is, it can be employed in the search of raw data.

Taking an array with finite elements as an example, the bilinear search can be used to find some specific elements in the array at one stretch. The elements in the array can also be broken into two sectors and each of the sectors searched in a 'bilinear' manner. That is, the first sector is searched followed by the last sector. Going further, the elements in this array can be broken into three, four, five etc. sectors before searching for some specified element in each of the sectors contained in the array. The search on each sector is carried out in the manner described above until the middle sector(s) is/are reached and searched.

The use of the bilinear search to search for a specified element in an array at one stretch is referred to in this work as 'single bilinear search'. Dividing the array into two sectors is referred to here as the 'double bilinear'. Dividing into three 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 our specified element, is called 'quintuple bilinear search'.

This method increases the probability/chances of finding the search item faster than in the case of the one directional linear search. It combines some of the constructive features found in binary and linear search algorithms to produce a hybrid that can take care of the disadvantages in the linear search algorithms. For this research, the sector division stops on the fifth one.

II. LITERATURE REVIEW OF SEARCH ALGORITHMS

The linear search is an algorithm that 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, its O notation is of order n; O(n) [15].

Sample Algorithm for the Linear Search on an array [A] [17]:

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

```
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 elrment  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 [15]. The efficiency, given that the data items have been sorted is O(logn).

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

```
 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
```

Many authors have tried to improve on the Performance of these two fundamental algorithms in data structure. [14] tried to improve the linear search algorithm with model based approaches for MAXSAT solving using models found by the SAT solver to partition the relaxation variables. [5] also made attempts to come up with an algorithm that has the same time complexity as that of the linear search O(n), but better in terms of execution time. There have been efforts trying to improve on the performance of binary search algorithm as well. [3] tried to enhance binary search algorithm by ensuring that the search is performed if and only if the search key is within feasible search space. Thus, claiming a better time complexity than the binary search algorithm. [9] tried to improve on the efficiency of the linear and binary search algorithms by proposing the Hash based searching technique for searching elements in a given array. Most of these proposed techniques however, do not use the simple logical means of searching for items as in linear and binary search algorithms.

[1] also made a modification on the binary search algorithm to optimize the worst case of this search algorithm. They tried to achieve this by comparing the input element with the first & last element of the data set along with the middle element. They also tried to check whether the input number belong to the range of numbers present in the given data set at each iteration there by reducing the time taken by the worst cases of binary search algorithm. The shortcoming of this algorithm as with most other variations of binary search however, is its dependency on sorting. If the sorting algorithm used for its implementation is inefficient it would affect the overall performance of the modified binary search algorithm [2].

[11] worked on the performance of the linear search algorithm by stopping a search when an element larger than the target element is encountered. However, this method 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 algorithm just like in the binary search. That is, its efficiency would be dependent on the sorting algorithm used for its implementation.

[4] applied a modified linear search technique by comparing the search item K by one increasing index from the first element to the middle element and the another increasing index from the middle to the last element simultaneously. This check was continued until either a match with the search key K is found or the list is exhausted. These authors went further by dividing the array list in three equal halves and the search key K was compared with the list of elements by increasing index from first element to the first middle element, second one by increasing index from first middle element to the second middle element and the last one by increasing index from second middle element to the last element of the list. This check was continued until a match with the search key K is found or the list become exhausted. The modification proposed in this research work however, varies from the above as the search algorithm is conducted from two ends of a list and terminates in the middle. This proposed technique enables the list to be further broken into as many parts as possible. When this is done, the parts are also searched from two ends.

[7, 12] also came up with methods of searching for elements from two different points in an array of numbers. What these authors however, did not do is to

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

break the array down into parts (sectors) and search each sector and the elements contained within from two ends. The search would consequently not come from only two ends of an array but from different directions in the array. Hence, making this proposed search method a multi directional one.

III. RESEARCH MOTIVATION AND METHODOLOGY

To tackle some of the shortcomings found in the linear search algorithm, a new algorithm that still maintain simplicity in its search technique; the bilinear search and its five variations is introduced. The number of steps taken to locate some specific numbers in an array, noted. The objective is to determine if the bilinear search is practicable and if it is more efficient than the linear search.

Sample Algorithm for a bilinear search in an Array [A].

```
Step 1    start
Step 2    define a list of elements i such that i = A[1….n]
Step 3    search for an element 'k' by comparing with the
          left most element i=1
Step 4    and then compare with the rightmost element
i=n
Step 5    if element k is found then end. else
Step 6    goto i=2 and i=n-1
Step 7    if element is found then end. else goto i=3 AND
i=n-2
Step 8    if element is found then end. else continue until
          midpoint of n or last element i=n/2 is searched
Step 9    if element is not found then return k = 0
Step 10   end
```

Take a sample array [A] of 50 elements. The linear search and the various levels of the bilinear search are used to find some specific elements (pivots) in the array, as shown in Table 1.

(a) Single Bilinear Search Algorithm

This is a situation whereby all the number in the array are put together and searched 'bilinearly' in one stretch, Using this technique to find some elements in the above sample array we have Figure 1 etc.

Assuming two numbers; 32 and 27 are to be searched for in the array of numbers above, using the linear search, number 32 is found after the 50th search and the number 27 is found after the 5th, 32nd and 49th search respectively. Using the bilinear search, the element 32 is found at point 50 after the first comparison. And the number 27 is found at point 49 after the second comparison. But searching further for the remaining numbers 32 and 27 in the array, after the 25th comparison it is discovered that there is no other 32 in the array. And for the number 27 in the array, it is found at two others points; 5 and 32 after the 5th and 19th comparison respectively. Should this method of searching be applied to an array of odd number say 51, then the search at the middle point would pick only one element in the middle of the array for comparison

b) Double Bilinear Search

In searching for the same two numbers in the array, numbers 32 and 27 using the double bilinear search, the first sector is searched, and after the first 13 comparisons, no number 32 was found. But on the very first comparison in the second sector, (14th comparison), the number 32 is found. Going through the remaining 12 comparisons in the second sector, no other 32 was discovered. Finding the number 27, on the 5th comparison in the first sector, it is found at point 5. And in the second sector, it is found after the second search (15th comparison) at point 49. And continuing the search, the last number 27 was found after the 7th search in the sector (19th comparison).

c) Triple Bilinear Search Algorithm

It should be noted that when the array is broken down into sectors, the sectors are also searched from two directions as it is done for the array of numbers. Hence, picking the first sector, the last one before falling back to the middle one. This is reflected in the numbering of the sectors in the diagram above as in the other diagrams in this work.

Now searching for the two numbers mentioned above, 32 and 27 using the triple bilinear algorithm, the first sector is searched, after eight comparisons no number 32 was found. Going over to the third sector, the first search (9th comparison) yields the number 32. Conducting the remaining 16 comparisons in the second and third sectors, no other 32 was found. For the number 27, searching the first sector, the number 27 was found after the 5th comparison. Switching over to the third (last) sector, the number 27 is found on the second comparison (10th comparison). The last number 27 is found on the first comparison (18th comparison) in the second sector.

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

d) Quadruple Bilinear Search Algorithm

The sectors above are also search from two directions, starting with the first, the last, the second and then the second to the last, hence, the numbering of the sectors above. Searching the first sector, no number 32 was found. Going over to the last sector (4th), the very first search (7th comparison), yields the number 32. The search is over. But assuming confirmation is needed as to whether another number 32 occur in the array, the remaining comparison in this sector is checked, and then the second and finally the third sector are checked. All the sectors yields no other 32. Searching for number 27, it is found after the 5th comparison in the first sector. Going to the last sector, another 27 is found on the second search (8th comparison). Searching on, no number 27 found in the second sector, but finally found one at position 5 in the last (third) sector after 24th comparison.

e) Quintuple Bilinear Search Algorithm

The sectors are searched from two directions. as indicated in the numbering of the sectors above. Searching for the two numbers 32 and 27 using Quintuple Bilinear Search Algorithm, the first sector is searched and no 32 found. Moving to the last sector, the first search in this sector (6th comparison), yields the number 32. Searching further, the remaining numbers in this sector are searched. The second sector, the fourth sector, and finally the fifth sector, are then searched but no other 32 was found. Searching for number 27, after the 5th comparison in the first sector, it is found. Searching further, another 27 is found on the second search (7th comparison) in the last sector. Moving to the second sector, no 27 is found. In the fourth sector, on the second search (17th comparison), a third 27 is found. Searching the remaining numbers in this sector, and the remaining in the third sector, yields no other 27.

IV. RESULTS AND ANALYSIS

4.1 Result

Table 2 below shows the number of searches used by the linear, and the five bilinear variations to manually find four different elements in the 50 elements array listed above.

4.2 Analysis of Result

Analysing the results from Table 2 and Figures 1 to 5, it can be seen that finding the four pivot elements took the linear search a total of 136 searches. For the single bilinear search, a total of 27 searches were used to find them. But as the number of elements is further segmented to double bilinear, the number of searches rises to 53. Going further to the triple bilinear the number of searches drops a bit to 42 only to rise back to 44 for the quadruple bilinear Search. And getting to the quintuple bilinear search, the number of searches drops back to 35. The reason behind the rise and fall of the number of searches in the varied bilinear searches can be attributed to the low number of elements in the array. The time efficiency however, of all the variations of the modified linear search algorithm, proves to be better than that of the linear search algorithm. The time complexity of the modified linear search algorithm may be expressed as O(n/2) but with no consideration given to constants, the complexity still remains O(n).

V. CONCLUSION

The binary search algorithm is listed among the very efficient search method for pragmatic purposes. But the advantage implicit in it assumes that the items to be searched would always be in sorted form, which may not be true in all cases [2]. Most data collected in their raw form are not sorted. This remains one of the major drawbacks of the binary search. The linear search on the other hand comes in handy as one of the popular search methods that can be used in this situation, but with the shortcomings associated with linear search; its O notation is of order N; O(N), most of the time doesn't become an option. This calls for a need to develop another algorithm to take care of the shortcomings.

The development of a new searching algorithm called the 'bilinear search algorithm' that does not search from only one direction as in linear search and allocates 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] R. C. Ankit, M. Rishikesh and M. Tanaya, International Journal of Applied Information Systems (IJAIS), Volume 7, No. 2, April 2014.

[2] 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.

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

[3] E. O. Bennet, V. E. Ejiofor and R. I. Akpan, Efficient Algorithm for Binary Search Enhancements. Journal of Computer Science and its Application, Volume 22, No.1, 2015.

[4] M. Bhaumik, S. Saha and S. Das, A new modified linear search. Working paper, 2016. https://www.researchgate.net/publication/309609322_A_new_modified_linear_search. Retrieved 2018.

[5] C. B. Bishnu, A Search Algorithm. Journal of Applied and Computational Mathematics (JAPPL Computat Math.), Volume 5, Issue 4. pp 322. August, 2016.

[6] Computer Science unplugged, Searching, 2014. http://csunplugged.org/searching-algorithms. Retrieved, 2015.

[7] K. Divya and B. L. Pal, International Journal of Computer Science and Mobile Computing (IJCSMC), Volume 4, Issue 12, pp 148-155, December 2015.

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

[9] R. Jyotirmayee and K. Raghvendra, Hash Based Searching Algorithm. International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET), Volume 2, Issue 2. February, 2013.

[10] 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, pp751-758. July, 2014.

[11] D. Nell, T. Daniel and W. Chip, Weems Object-Oriented Data Structures Using Java, Jones & Bartlett Learning LLC, 2016.

[12] A. Nitin, B. Garima and S. Neha, Two way Linear Search Algorithm International Journal of Computer Applications, Volume 107, No. 21, December, 2014.

[13] Prelude. (2011). Linear Search, Binary Search and other Searching Techniques, 2011, http://www.cprogramming.com/discussionarticles/sorting_and_searching.html, Downloaded 2016.

[14] M. Reuben, M. Vasco and L. Ines, Knowledge Representation and Automated Reasoning. Journal of Experimental and Theoretical Artificial Intelligence. Volume 27, Issue 5, pp 673 -701. 2015.

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

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

[17] Tutorialspoint, Data Structure and Algorithms Linear Search, 2018, http://www.tutorialspoint.com/data_structures_algorithms/linear_search_algorithms.htm. Retrieved September, 2018.

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

Table 1: List of elements in an array**.**

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Element	5	4	6	18	27	41	3	22	30	8	25	1	16	39	13	7	10	15	40	37

Index	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Element	48	6	3	18	31	29	9	50	23	13	17	27	3	6	12	19	1	22	5	35

Index	41	42	43	44	45	46	47	48	49	50
Element	3	43	1	20	3	28	3	48	27	32

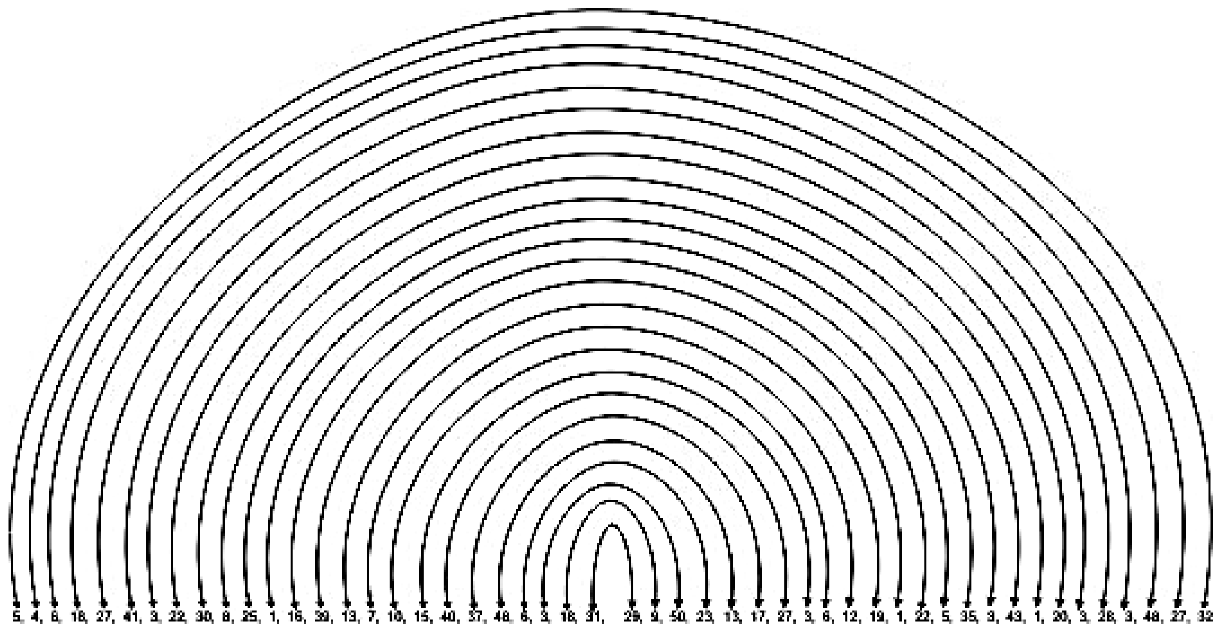

Figure 1: Single Bilinear Search Algorithm

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

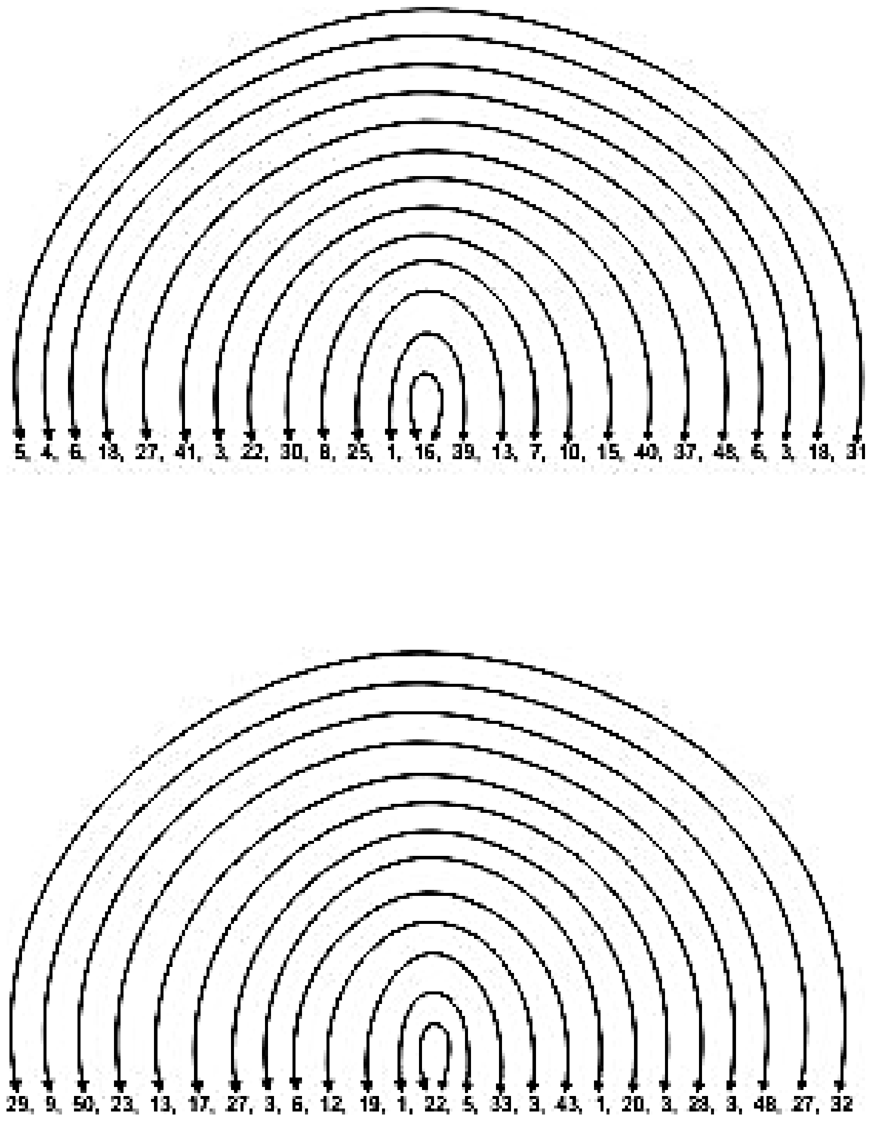

Figure 2: Double Bilinear Search Algorithm

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

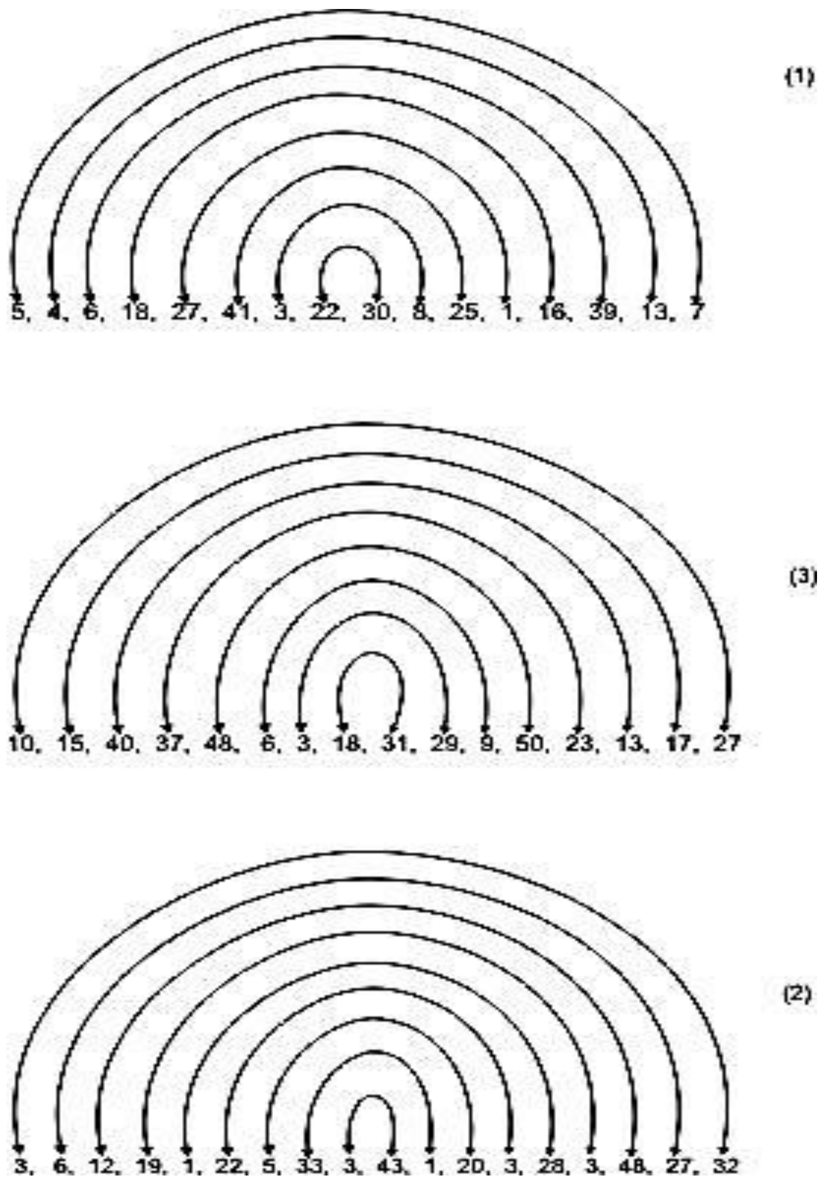

Figure 3: Triple Bilinear Search Algorithm

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

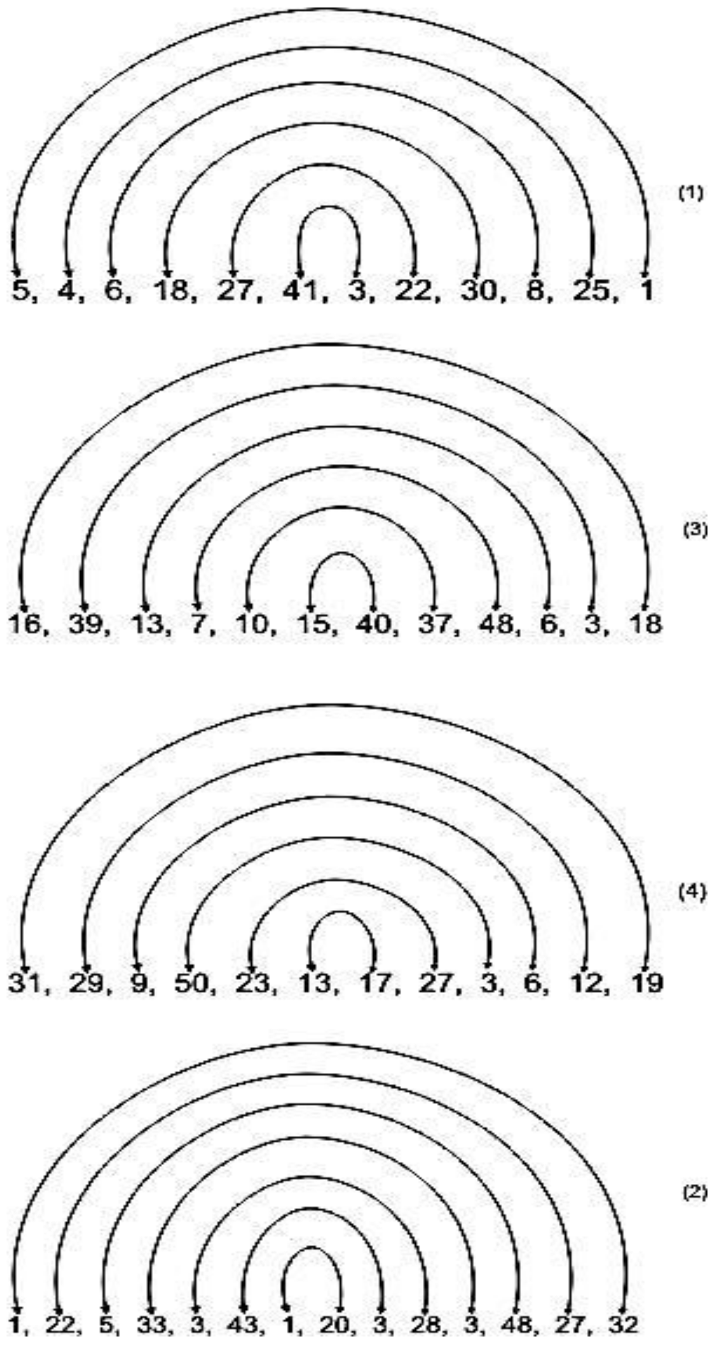

Figure 4: Quadruple Bilinear Search Algorithm

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

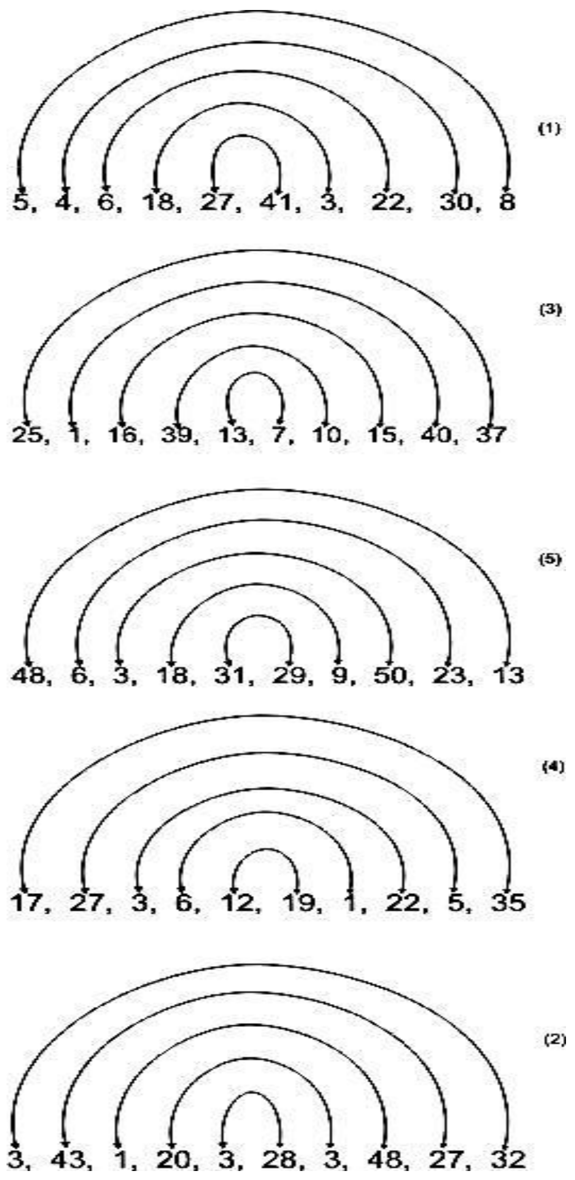

Figure 5: quintuple Bilinear Search Algorithm

© 2019 *Afr. J. Comp. & ICT* – All Rights Reserved
https://afrjcict.net

Table 2: Number of search steps taken using each search method

SEARCHED NUMBERS	LINEAR	SINGLE BILINEAR	DOUBLE BILINEAR	TRIPLE BILINEAR	QUADRUPLE BILINEAR	QUINTUPLE BILINEAR
32	**50**	**1**	**14**	**9**	**7**	**6**
FIRST 27	**5**	**2**	**5**	**5**	**5**	**5**
SECOND 27	**32**	**5**	**15**	**10**	**8**	**7**
THIRD 27	**49**	**19**	**19**	**18**	**24**	**17**
TOTAL NO OF SEARCHES	**136**	**27**	**53**	**42**	**44**	**35**