Left-Right Mid Way Linear Searching Algorithm

Lakshay Goyal
School of Computer Science
UPES Dehradun

Kunal Sharma
School of Computer Science
UPES Dehradun

Tanu Rai
School of Computer Science
UPES Dehradun

Bharat Gupta
School of Computer Science
UPES Dehradun

Nitin Arora
Electronics & Computer Discipline
IIT Roorkee

ABSTRACT
Searching on an element from a large set of data elements is always a time taking task. It becomes more challenging if the data set is random and is very large, having millions of items. Many searching techniques already exist like linear search, two-way linear search, etc. All the exiting algorithms have their best and worst performances depending upon the input cases. This paper proposed a new technique Left-Right midway (LRMW) linear search algorithm, which is based on the midway and then two-way method. The proposed algorithm works well in many types of input cases. The proposed algorithm, linear search algorithm, and two-way linear search algorithm is implemented in C language and tested on different size of input elements and time of execution is calculated for all the three algorithms for all input elements. Results show significant improvement if the data set is extensive and the required element is present nearby the middle element. The time complexity analysis of the proposed algorithm is also discussed in the paper.

Keywords
Linear Search; Binary search; Two Way Linear Search; Time Complexity Analysis.

1. INTRODUCTION
Searching is a technique that is used to find a particular element in the array. When the iterator finds an element that is equal to the key, then the search stops, and it returns the index of the element. If the key is not present in the array, then the size of the array will be returned. There exist many variants of the linear search algorithm. In [1], the author discussed many searching and sorting algorithms. Without searching, we would have to look at each item of data individually, to see whether it is what you are looking for. In this, we discussed three types of searching methods: Mid Two Way Linear Search(proposed algorithm), Two Way Linear Search [4], and Linear Search. We also discussed the time complexity analysis of these three searching algorithms. These searches are used to find a particular element in the array. In linear search algorithms, the array doesn't need to be sorted.

The rest of the paper is divided as follows: Related work is discussed in section 2. Section 3 represents the existing algorithms related to the article. The proposed algorithm is discussed in Section 4. Time complexity analysis is done in Section 5. Results are shown in section 6, and the paper has been concluded along with future scope in section 7.

2. RELATED WORK
Many researchers work on searching for algorithms. In this section, we discussed the work associated with searching algorithms. In [1], the author discussed many sorting and searching algorithms and also did the comparative analysis. In [2] author compared the searching process of linear, binary, and interpolation algorithms and also mentioned that in binary search, a sorted array is required. The two-way linear search algorithm is discussed in [4]. In this, the author searched the key element from two sides left and right and found a remarkable improvement in the time of execution for particular cases. A new algorithm two-way counting position sort is discussed in [3], which is based on a non-comparative sorting technique. Double hashing technique in closed hashing search is used in [5]. The author also quoted that this is one of the best methods. In [6], Odd-Even based binary search algorithm is proposed by the author. Odd-Even based binary search algorithm is better than a binary search algorithm as this algorithm searches the key element in less time. A ternary search algorithm is proposed in [7], which is based on binary search. The only difference is that the input array is divided into three parts instead of two parts. In [8], a novel sorting algorithm is discussed by the authors. They proposed sorting algorithms and compared them with bubble and insertion sort. In [9], the authors deploy all the sorting and searching algorithms. An iterative method for recreating a binary tree from its traversals is discussed in [10] and used linear search technique in this method. All the basics searching and sorting algorithm with time complexity analysis is presented in [11-12]

3. EXISTING ALGORITHM
There are many variants of the linear search algorithm. In this section, we discussed some of the existing options of linear search algorithms.

3.1 Linear search
Linear Search is used to find a particular element in an array. There is not the compulsion of a collection to be sorted. Linear Search starts consecutive searching the items in the array, and if the element has been found, it will display the particular element and the index value of the element. The performance of linear search improves if the required value is present near the beginning of the list than to its end.

Algorithm

Input*: Array L, Key x and total number n*

Output*: If Key belongs to L return 1 else return -1*

```
1: i = 0
2: for (i = 0; i< n; i++)
3: if (L[i] == x)
4: then return 1
5: if (i==x)
6: then return -1
```

Let the input array be as follows (figure 1), and the key element is 55. Now, to search a key component of the input array using the linear algorithm, four comparisons are required.

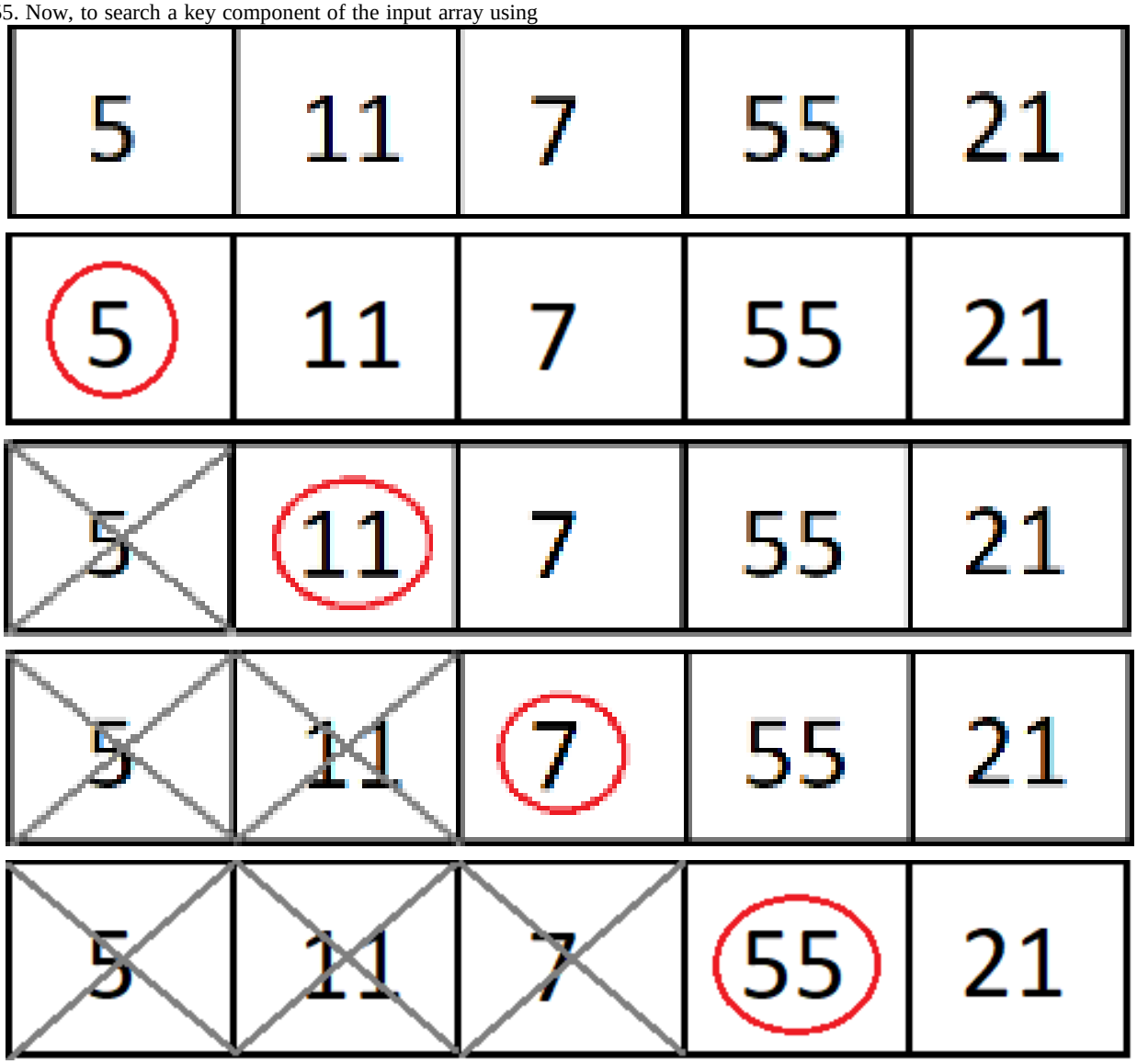


Fig1. Processing of linear search

3.2 Two way linear search

In Two Way Linear Search, we initialize the array with the values. In this searching will start to search the array from index 0 as well as from the index n-1. If the value is found at its position from the left or the right, then it will return 1. If the values are not found, the value returned is -1. Is it much faster than Linear Search as linear search start searching from index 0 but two-way linear search both index 0 and index n-1?The proposed algorithm executes in less time if the searching element is present at the last positions or somewhere in the first mid part of the array.

Algorithm

Input*: Array L, Key x and total number n*

Output*: If Key belongs to L return 1 else return -1*

```
1: first=0
2: last=n-1
3: while (first <= last)
4: if ((L[first] == x) or (L[last] == x))
5: then return 1    //end of while loop
6: else first = first+1; last = last-1;
7: if (first > high)
8: return -1
```

Let the input array be as follows (figure 2), and the key element is 55. Now, to search for a key element in the input array using the linear algorithm, two iterations are required.

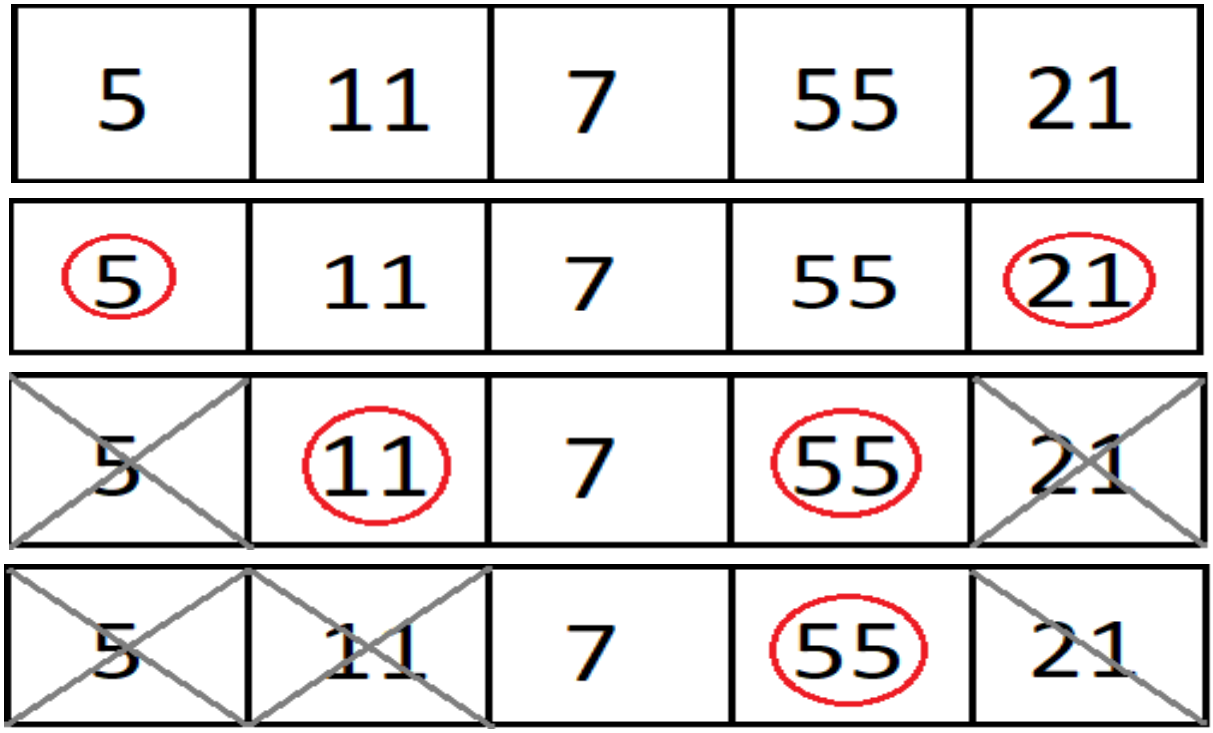


Fig. 2: Processing of Two way linear search

4. PROPOSED ALGORITHM

In the proposed linear search, we initialize the array with the values. In this searching will start to search the variety from the middle. In these, the array is divided into halves, and the key is compared with the middle element of the array. If the match is found then, the location of the middle element is returned; otherwise, we will apply the two-way linear search in both the left and right subarray. The proposed algorithm executes in less time if the searching element is present in the middle position or last positions or somewhere in the first mid part of the array.

Algorithm

Input*: Array L, Key x and total number n*

Output*: If Key belongs to L return 1 and return -1*

1. *low=0;*

2. *mid=(low+high)/2;*

3. ***while*** *((low<=mid-1) && (mid+1<=high))*

4. ***if*** *(array[mid]==search)*

a. *4.* ***then*** *return 1 //end of while loop*

5. ***if*** *(((array[low]==search) || (array[mid-1]==search)) || ((array[mid+1]==search) || (array[high]==search)))*

6. ***then*** *return 1*

7. ***else***

8. *low=low+1;*

a. *mid-1 =mid-1 -1;*

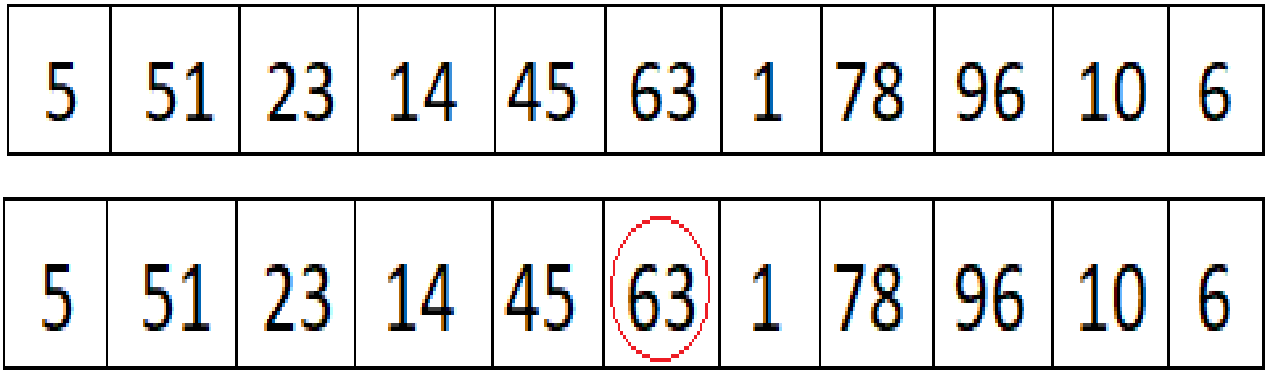
b. *mid+1 =mid+1 +1;*

9. *high=high-1;*

10. ***if*** *((low>x) || (y>high))*

11. ***return -1***

Let the input array be as follows (figure 3), and the key element is 45. Now, to search for a key element in the input array using the proposed algorithm, only one comparison is required. Here 63 is the middle element. First, we will compare the key to the central element. Then the algorithm will apply a two-way search algorithm in both the halves.



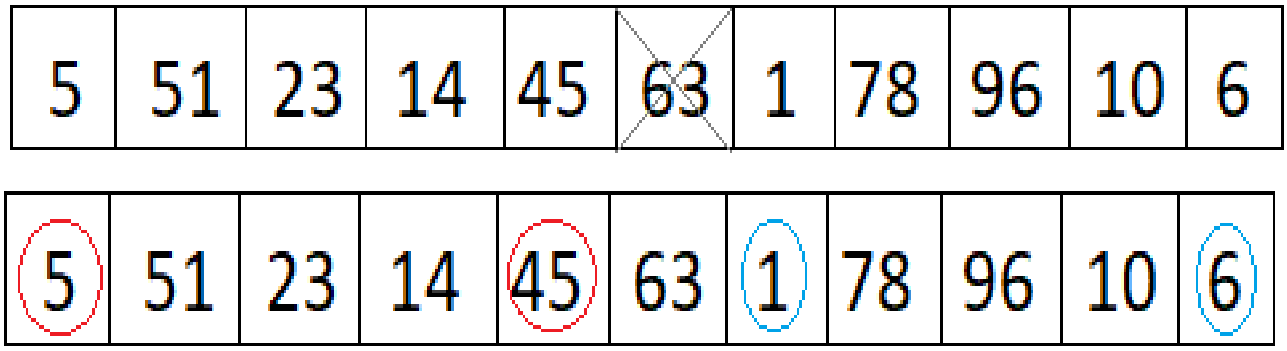


Fig. 3: Processing of proposed algorithm

5. TIME COMPLEXITY ANALYSIS

In this section, we computed the time of execution of the three algorithms for the different sizes of the input array. The details are shown in table 1.

Table 1: Time of execution for different exiting algorithms and proposed algorithms for the different sizes of inputs and various types of input cases.

Searching Algorithm / Input	1000		10000		50000		100000		500000	
	EPSP	EPEP	EPSP	EPEP	EPSP	EPEP	EPSP	EPEP	EPSP	EPEP
Two Way Linear Search	0	0	0	0	0	0	0.02	0	0.068	0.052
Linear Search	0	0	0	0	0	0.016	0.021	0.061	0.092	0.125
Proposed Algorithm	0	0	0	0	0	0	0	0	0.01	0.01

EPSP: Element Present at Starting present EPEP: Element Present at ending Position

5.1 Linear Search Algorithm

The time complexity analysis is done using a linear method. For the linear search algorithm as the size of the problem is reduced to half in each iteration. The recurrence relation for the linear search is:

$$T(n) = \begin{cases} T(n-1) + 1, & n > 0 \\ 0, & n = 0 \end{cases} \quad (1)$$

on solving eq. 1 for T(n)

$$T(n) = T(n-1) + 1 \quad (2)$$
$$T(n-1) = T(n-2) + 1 \quad (3)$$
$$T(n-2) = T(n-3) + 1 \quad (4)$$
$$T(n-3) = T(n-4) + 1 \quad (5)$$
....
....
$$T(1) = T(0) + 1 \quad (n+1)$$

Adding all the eq. (2) to eq. (n+1)

$$T(n) = T(0) + n = O(n)$$

We can say that the time complexity of the linear search algorithm is O(n)

5.2 Two way linear search Algorithm

In Two Way Linear Search, the array doesn't need to be sorted. In this, we will search the element in the array from index 0 as well as from the index n-1. If the searched element is found at its position from the left or the right, then it will return 1. If the values are not found, the value returned is -1. Is it much faster than Linear Search as linear search start searching from index 0 but two-way linear search both index 0 and index n-1.

5.3 Proposed Algorithm

In Mid Two Way Linear Search, firstly, we will search the middle element if the searched element found it will return 1. Then we will divide the array into two parts, and we will apply the Two Way Linear Search in both the left and right sub array. If the value is found, it will return 1; otherwise, it will return -1. .This array doesn't need to be sorted. It is much faster than Two Way Linear Search as it starts searching from both index 0 and index n-1 in both sub arrays.

6. RESULTS AND DISCUSSION

In this section, we computed and compared the time complexity of the proposed algorithm (mid-two-way linear search), two-way linear search, and linear search algorithms. The time of execution vs. size of the input array is shown in figure 1. The x-axis represents the size of the array, and the y-axis represents the time of execution. All three algorithms are tested for the random array of sizes one lakh, two lakh, three lakh, four lakh, and five lakhs elements. The time of execution is computed in milliseconds. The results show that in starting the time execution is almost the same, but by gradually increasing the

size of the array two-way linear search takes very less time as compared to linear search as it searches the values from both the index.

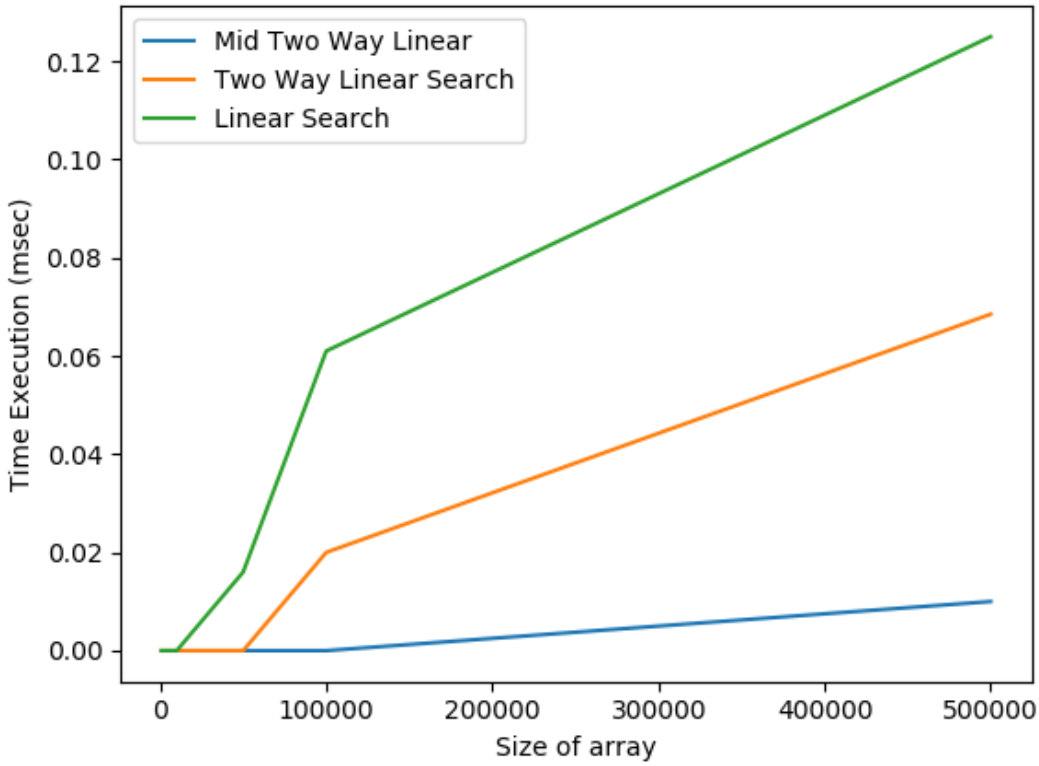


Fig. 4: Time of execution vs. size of input array for all the three algorithms

7. CONCLUSION AND FUTURE SCOPE

Searching is a technique that is used to find a particular element in an array. In the proposed linear search algorithm, two-way linear search, and linear search algorithms, it's not necessary for the array to be sorted unlike in case of binary search and ternary search. By increasing the size of the array from 100 elements to 10,000 elements and more, the time of execution increases from less time to consider time. There can be any random input elements. The result shows that the proposed searching algorithm is working well for all input values, and it takes lesser time for specific types of inputs as compared to the other searching techniques. In the future, many algorithms can be proposed related to big data analysis.

8. REFERENCES

[1] Subbarayudu, B., et al. "Comparative Analysis on Sorting and Searching Algorithms." *International Journal of Civil Engineering and Technology (IJCIET)* 8.8 (2017): 955-978.

[2] Rahim, Robbi, et al. "Comparison Searching Process of Linear, Binary and Interpolation Algorithm." *J. Phys. Conf. Ser*. Vol. 930. No. 1. 2017.

[3] Arora, Nitin, Anil Kumar, and Pramod Mehra. "Two Way Counting Position Sort." *International Journal of Computer Applications* 57.12 (2012).

[4] Arora, Nitin, Garima Bhasin, and Neha Sharma. "Two way linear search algorithm." *International Journal of Computer Applications* 107.21 (2014).

[5] Rahim, Robbi, Iskandar Zulkarnain, and Hendra Jaya. "Double hashing technique in closed hashing search process." *IOP Conference Series: Materials Science and Engineering*. Vol. 237. No. 1. IOP Publishing, 2017.

[6] Karthik, S. "Odd Even Based Binary Search." *International Journal of Computer Engineering & Technology (IJCET)* 7.5 (2016): 40-55.

[7] Arora, Nitin, Mamta Martolia Arora, and Esha Arora. "A Novel Ternary Search Algorithm." *International Journal of Computer Applications* 144.11 (2016).

[8] Arora, Nitin, Suresh Kumar, and Vivek Kumar Tamta. "A novel sorting algorithm and comparison with Bubble sort and Insertion sort." *International Journal of Computer Applications* 45.1 (2012): 31-32.

[9] ALAJEELI, Adnan Saher Mohammed. *Development of Sorting and Searching Algorithms*. Diss. Ankara Yıldırım Beyazıt Üniversitesi Fen Bilimleri Enstitüsü, 2017.

[10] Arora, Nitin, Pradeep Kumar Kaushik, and Satendra Kumar. "Iterative method for recreating a binary tree from its traversals." *International Journal of Computer Applications* 57.11 (2012): 6-13.

[11] Lipschutz, Seymour. "Data Structure with C, Schaum Series." (2009).

[12] McGraw-Hill, Herbert Schildt Tata. "The Complete Reference C." (2005).