COMPARATIVE EXECUTION TIME ANALYSIS FOR BINARY, JUMP, AND INTERPOLATION SEARCH ALGORITHMS IN OBJECT ORIENTED LANGUAGES

ISOLA Esther O.[1], OGUNDOYIN Ibrahim K.[2], OMOTOSHO Lawrence O.[3], ADIGUN Adepeju A.[4]
OSUN STATE UNIVERSITY, OSOGBO, OSUN STATE
[1]esther.isola@uniosun.edu.ng, [2]ibraheem.ogundoyin@uniosun.edu.ng,
[3]lawrence.omotosho@uniosun.edu.ng, [4]adigunaa@uniosun.edu.ng

Keywords: Time, searching algorithm, Execution time, Comparative Analysis, object-oriented languages

Abstract: *In computer, execution time is critical, and knowing which algorithm and programming language will perform better, among other things will save programmers time and effort. The purpose of this work is to compare three searching algorithms: Binary, Jump, and Interpolation, which are implemented in three object-oriented programming languages: C++, Java, and Python using execution time. Data size, algorithm style, and programming languages were used to compare the execution times of these searching methods. A time stamp was utilized to determine the search algorithm's execution time. The results demonstrate that among the programming languages tested, C++ is faster than Java and Python in terms of execution. Interpolation search is also the algorithm with the shortest execution time for vast amounts of data, followed by Binary search and the Jump search method. It was discovered that the size of the data and the programming language employed have an impact on the execution time. As a result, this paper offers guidance on selecting a search algorithm and the type of programming language to be used. In the current age of explosive increase in the use of searching for data or information on a device, this is done in order to minimize the execution time of a searching algorithm.*

1. INTRODUCTION

The most basic requirement for any computing application is information retrieval, which necessitates search operations on vast databases implemented by diverse data structures. A key feature of the computing industry is finding an element from a list. For searching an element, a variety of algorithms have been developed, including linear search, binary search, jump search, and interpolation search algorithms [1].

Green computing, which focuses on developing energy and power efficient devices, using non-toxic materials, and minimizing e-waste in such design, has received a lot of attention recently. As a result, the hardware part of green computing and green IT has received more attention than the software aspect, despite the fact that green computing involves the study and practice of optimally employing all computing resources. As a result, determining and enhancing the efficiency of any algorithm requires a thorough understanding of its execution time and energy usage/consumption. Many of the computer's tasks are likely to involve search activities at some point while using the system. As a result, it's important to understand which search techniques should and shouldn't be utilized in data processing to reduce the impact of their flaws on the final product [2].

The longer an algorithm takes to execute, the more energy it consumes, which has a direct impact on the environment. The elements that affect the execution time of searching algorithms are examined in this work, and the findings are compared and contrasted. Binary, jump, and interpolation algorithms were used in the search, and they were implemented in C++, Java, and Python computer languages.

2. RELATED WORKS

[3] did a comparative study to compare the performance of three algorithms in terms of execution time: comb, cocktail, and count sorting algorithms. On the same platform, Java programming was utilized to implement the algorithms that used numeric data. The cocktail method was discovered to have the quickest execution time of the three algorithms, while counting sort is in second place. Furthermore, in terms of execution time, Comb is ranked lowest. Different sorting algorithms were utilized in the study, but only Java was used to compare them, and the energy usage of the methods was not taken into account. [4] used the C programming language to conduct experiments to see how different sorting algorithms affect energy consumption. It was revealed that time and energy consumption had an impact on the efficiency of these sorting algorithms, with rapid sort, merge sort, and shell sort being in the same time and energy consumption range, followed by insertion and selection sort which is far better than Bubble sort. The implementation, however, is limited to the C programming language and a non-varying small data size of 10,000.
[5] used execution time to compare three sorting algorithms: quick, merge, and insertion sort, which were implemented in three computer languages (C++, Java, and Python). The three sorting algorithms were compared in terms of execution time based on programming language, data size, and method of implementation. Their findings revealed that data size, programming language, and implementation method style are all elements that influence software execution time, with the way software is written and the programming language used to be two of the most important predictors of software execution time [6][7]. This is an essential point, but it does not address the linear search's efficiency.

2.1 Search algorithms

Any algorithm that solves the search problem, namely retrieving information contained within some data structure or calculated in the search space of a problem domain, with discrete or continuous values, is referred to as a search algorithm [8]. A search algorithm is a set of formulas and commands that are used to solve problems and provide possible answers to problems that people are faced with, whereas searching is the universal way of addressing problems in AI and the foundation on which search engines are created. Binary, Jump, and Interpolation search algorithms are some of the algorithms that will be examined in this study.

2.2. Binary Search

The Binary Search algorithm is used on a big sorted array or list. Its O(log n) time complexity makes it extremely quick when compared to other searching methods [9]. The array or list of elements must be sorted in order for the binary search method to work. The procedure is as follows:
1. Pick a value from the (sorted) array in the middle.
2. If the value matches what we're looking for, we're done.
3. If the value is smaller than what we're looking for, go back to step one with the left subarray.
4. Alternatively, if the value is bigger than what we're looking for, go back to step one with the correct subarray.

2.3. Interpolation Search

The binary search is essentially improved by the interpolation search. With each step, the algorithm calculates where in the remaining search space the target element might be based on the value of the limits compared to the target element, similar to how one might search a telephone book for a name. If the elements are evenly distributed, the time complexity is O(log (log n)). In worst cases it can take up to O(n).

2.4. Jump Search

The Jump Search Approach is a new algorithm for finding a specific element in a sorted array [10]. When compared to a linear search algorithm, the basic principle behind this searching strategy is to search a smaller number of elements (which scans every element in the array to check if it matches with the element being searched or not). This can be accomplished by skipping a set number of array elements, or jumping ahead by fixed number of steps in every iteration.

3. MATERIALS AND METHOD

Three different but related searching algorithms namely Binary, Jump, and Interpolation search,

were implemented in three different object-oriented programming languages. The languages used are C++, Java, and Python, in order to determine which of the search algorithms execute faster and which programming language performed better for implementing each of the algorithms. The algorithms were chosen for their resemblance to those found in a sorted list or array.

3.1. Implementation of Selected Searching Algorithm

The Searching Algorithms were implemented on a computer system utilizing the NetBeans development IDE and Python 3.7, which is loaded on a Lenovo Ideapad 110 laptop with Windows 10, Intel(R) Core (TM) i3-4005U CPU @ 1.70GHz, Radeon(tm) HD Graphics, 1.70GHz (processing speed), and 4GB of RAM space.

3.2. Binary Search Algorithm

Step 1: Input an array A of n elements and ―data to be search
Step 2: LB = 0, UB = n; mid = int ((LB+UB)/ 2)
Step 3: Repeat step 4 and 5 while (LB <= UB) and (A[mid]! = data)
Step 4: If (data < A[mid]) UB = mid–1
Step 5: Else LB = mid + 1
Step 6: Mid = int ((LB + UB)/ 2)
Step 7: If (A[mid] == data) Display ―the data found.
Step 8: Else Display ―the data is not found‖
Step 9: Exit

3.3. Interpolation Search Algorithm

Following are the steps of implementation that we will be following:
Step 1: Input a sorted array of n elements and the key to be searched
Step 2: Initialize low = 0 and high = n – 1
Step 3: Repeat the steps 4 through 7 until if (low < high)
Step 4: Mid = low + (high – low) × ((key – A[low]) / (A[high] – A[low]))
Step 5: If(key < A[mid]) high = mid–1
Step 6: Elseif (key > A[mid]) low = mid + 1
Step 7: ElseDisplay ― The key is not in the array‖
Step 8: STOP

3.4. Jump Search Algorithm

Steps for Jump Search Algorithms:
Step 1: Set i=0 and m = √n.
Step 2: Compare A[i] with item. If A[i] != item and A[i] < item, then jump to the next block. Also, do the following:
1. Set i = m
2. Increment m by √n
Step 3: Repeat the step 2 till m < n-1
Step 4: If A[i] > item, then move to the beginning of the current block and perform linear search.
1. Set x = i
2. Compare A[x] with item. If A[x]== item, then print x as the valid location else set x++
Repeat Step 4.1 and 4.2 till x < m
Step 5: Exit

3.5. Measurements

3.5.1 Time Stamp

A time stamp was inserted exactly above the called search function, and another time stamp was placed directly below the called sorting function. This is done to ensure that the execution time recorded is only for the purpose of capturing execution time. The execution time was calculated by subtracting the start and end times.

3.5.2 Execution Time

The time it takes for an algorithm to execute is called the Execution Time T. The search time will be calculated using a system clock imported from the programming language's libraries.
Execution The time was calculated by subtracting the start and end times. The times were recorded in seconds. The algorithm for execution time is given below:
Start_Time: Invoke _System_clock
Call_searchingAlgorithm_class/method
End_Time <----Invoke_System_Clock
Execution_Time = End_Time - Start_Time

4. RESULTS AND DISCUSSION

Table 1 shows the execution time and data size for the Binary search, Jump search, and Interpolation search algorithms implemented in C++, Java, and Python programming languages. Figures 1 to 6 show the graph for comparing the algorithms, which shows the data size and execution time. Table 1. shows that when the data size is 100 and 500, jump search algorithm has the highest value of 27.25 and 32.5 respectively for the three selected programming languages. The execution

time for the same method differs depending on the programming language. Because programming languages differ in terms of design and specifications. The execution time of the specified Python algorithms is much longer than that of the Java and C++ algorithms. It was discovered that the programming language used for any algorithm will have effect on the execution time.

Fig. 1 to 6 provide a comparison graph of execution time and data size for the Binary search, Jump search, and Interpolation search algorithms implemented in the C++, Java, and Python programming languages. The graphs depict the data size and average execution time of Binary, Jump, and Interpolation Search Algorithms implemented in C++, Java, and Python, respectively. Execution time increases as the data size grows, regardless of the programming language employed. This is shown in the graph's trend. Fig. 1 to 3 showed the trend that algorithm implemented in C++ programming language has the least execution time. Fig 4 to 6 also showed that out of the three-search algorithm considered, interpolation algorithm has the least execution time.

Table 1: Comparison of Average Execution Time of Binary, Jump, and Interpolation Search Algorithms Implementations in JAVA, C++ and Python.

DATA Size ('000)	**Java**			**C++**			**Python**		
	Binary	**Jump**	**Inter-polation**	**Binary**	**Jump**	**Inter-polation**	**Binary**	**Jump**	**Inter-polation**
100	15.75	27.25	15.5	12.75	20.25	12.50	40.75	46.25	46.25
200	19.25	31.50	19.25	16.25	25.50	16.25	45.25	55.00	47.00
300	23.25	31.75	19.50	20.25	25.75	17.50	52.00	58.50	47.25
400	27.5	32.25	23.50	23.50	28.25	20.50	55.25	59.75	48.75
500	34.75	32.50	24.75	30.55	30.50	20.75	57.25	60.25	54.25

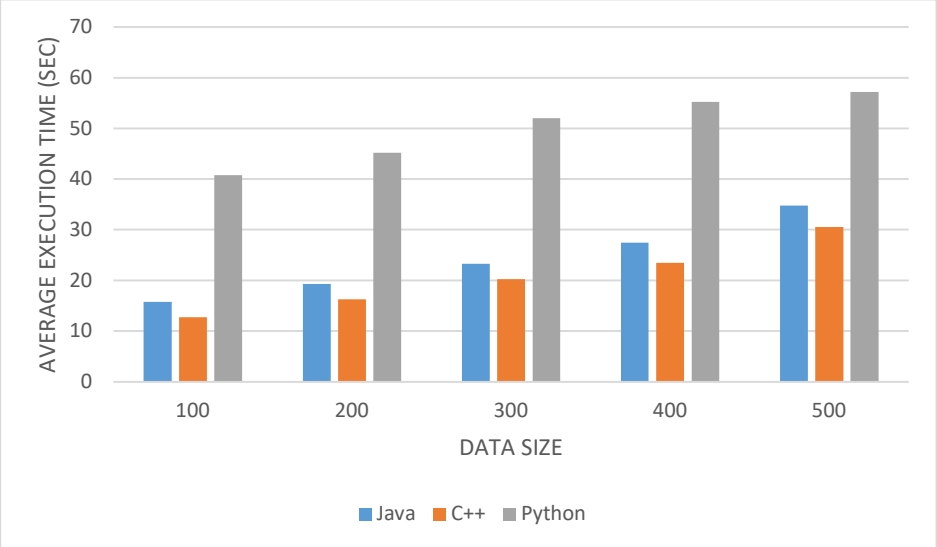

Figure 1: Comparison of Average Execution Time of Binary Search Algorithm Implementation in C++, Java and Python

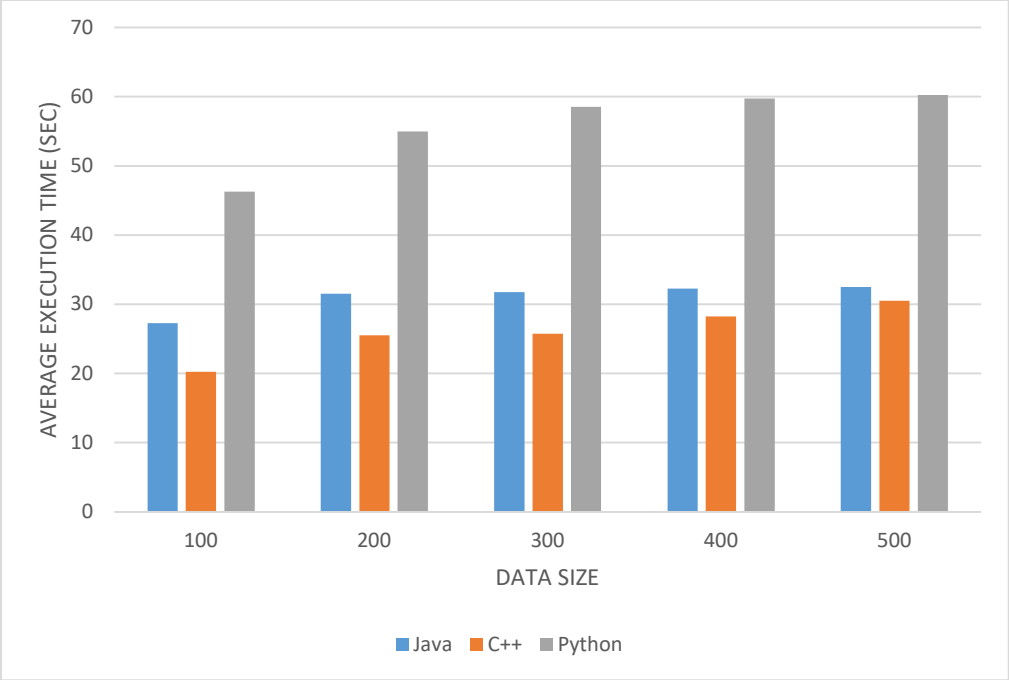

Figure 2: Comparison of Average Execution Time of Jump Search Algorithm Implementations in C++, Java and Python

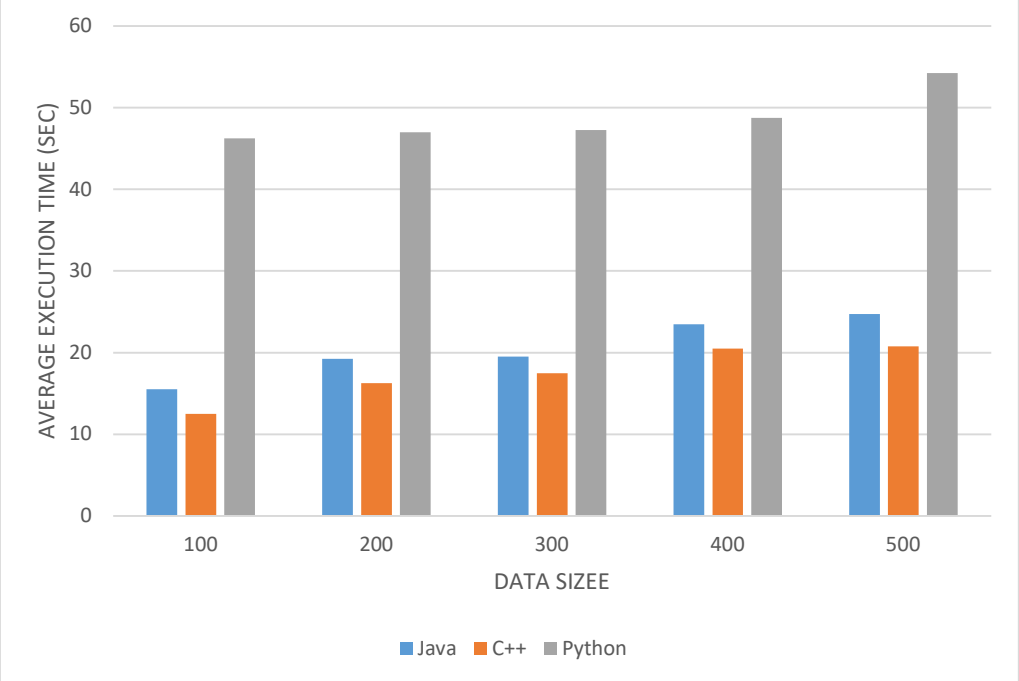

Figure 3: Comparison of Average Execution Time of Interpolation Search Algorithm Implementations in C++, Java and Python

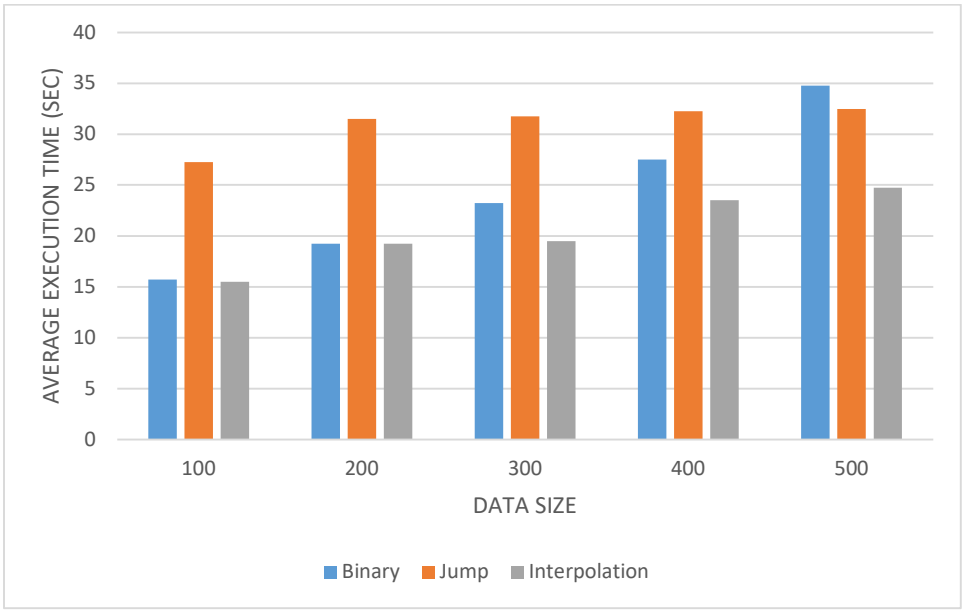

Fig. 4: Comparison of Average Execution Time of Binary, Jump, and Interpolation Search Algorithms Implementations in JAVA.

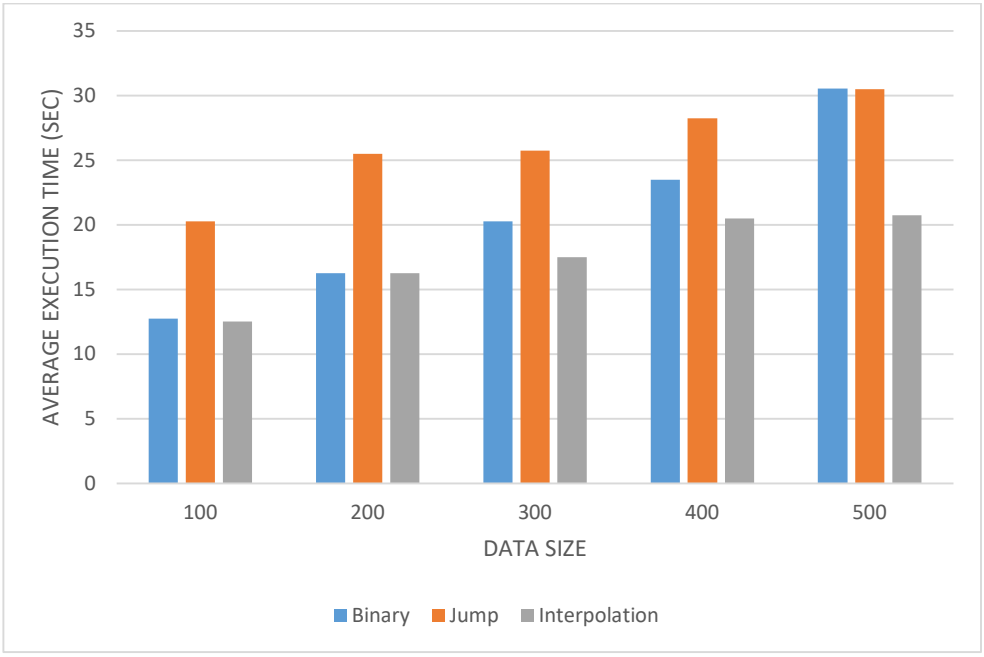

Fig. 5: Comparison of Average Execution Time of Binary, Jump, and Interpolation Search Algorithms Implementations in C++.

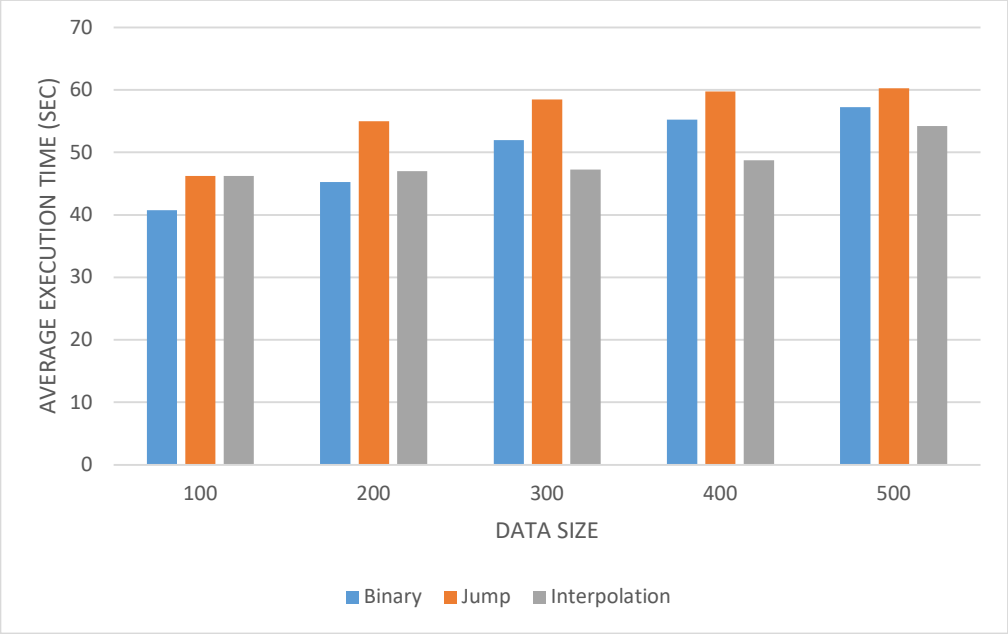

Figure 6: Comparison of Average Execution Time of Binary, Jump and Interpolation Search Algorithms Implementations in Python.

5. CONCLUSIONS

The work focused on execution time of some selected search algorithm implemented in C++, Java and Python programming language.
This study revealed that software execution time is influenced by some factors such as data size and programming language used.
It can be noted that the same searching algorithm has different execution tine with varying data sizes when implemented in different programming language.
The higher the data size the higher the time taken for searching.
Therefore, this gives developers knowledge on time efficiency in software leading to choosing codes over others based on their time performance.

6. REFERENCES

[1] Balogun G. B.," A Comparative Analysis of the Efficiencies of Binary and Linear Search Algorithms," African Journal of Computing & ICT, 13(1), 39 – 51, 2020.

[2] Balogun, G. B., "A Modified Linear Search Algorithm, African Journal of Computing & ICT", 12 (2) 43 54. 2019

[3] Elkahlout, A., and Maghari, Y. A., "A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting", International Research Journal of Engineering and Technology (IRJET) 4(1) 1 -10. 2017.

[4] Deepthi T; and Birunda A. M. J., "Time and Energy Efficiency: A Comparative Study of Sorting Algorithms Implemented in C", International Conference on Advancements in Computing Technologies (IJFRCSCE) 4(2)25–27. 2018.

[5] Ayodele O. S., and Oluwade B., " A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms using Three Programming Languages I: Execution Time Analysis", African Journal of Management Information System, 1(1) 1-18. 2019.

[6] Parimal Mridha, Binoy Kumar Datta., "An Algorithm for Analysis the Time Complexity for Iterated Local Search (ILS)", Journal of Research in Applied Mathematics, 7(6), 52 – 54. 2021.

[7] Isola E. O., Ogundoyin I. K., Akanbi C. O. and Adebayo O. Y., "The correlation among cognitive complexity metrics in Algorithm Analysis", Uniosun Journal of Engineering and Environmental Sciences 3(1), 2021.

[8] Liu, J. P., Yu, C. U., and Tsang, P. W., "Enhanced direct binary search algorithm for binary computer generated Fresnel holograms", Applied optics, 58(14). 2019.

[9] Kehinde A. Sotonwa , Monsurat O. Balogun, Esther O. Isola, Stephen O. Olabiyisi, Elijah O. Omidiora and Christopher A. Oyeleye, "Object Oriented

Programming Languages For Search Algorithms In Software Complexity Metrics" , International Research Journal of Computer Science (IRJCS), 6(4), 90 – 101. 2019.

[10] Zia, A. Z., "A Survey on Different Searching Algorithms" International Research Journal of Engineering and Technology, 7 (1), pp 1580–1584, 2020.