iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** X **Month of publication:** October 2017

DOI: http://doi.org/10.22214/ijraset.2017.10195

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue X, October 2017- Available at www.ijraset.com

Comparative Performance Analysis of Binary Search in Sequential and Parallel Processing

Shubham Dubey[1], Dr. Kirti Mathur[2]
[1]*International Institute of Professional Studies, Devi Ahilya University, Indore India*
[2]*International Institute of Professional Studies, Devi Ahilya University, Indore India*

Abstract: ***In order to deal with huge data set alternative design with proper function is desirable. These functions may perform operations sorting, searching, updating DBMS frequently. Apart from time and space metrics, energy, pattern and size of input, exact match or approximations are also key issues to be considered. So there is versatile need to seek improvement for performance. Among state of art approaches binary search relies on divide and conquer approach explore key item at mid element of array after each iteration and accordingly moves interval to new sub range. In this paper an analysis of state of art bisection algorithm has been presented with certain parameters as effectively with some dynamic alteration in input. Further, analysis has been done with parallel processing.***
Keywords: ***Searching, Parallel Processing, Time Taken for Searching, Comparison, Order***

I. INTRODUCTION

Binary search is a searching algorithm that gives result with order of complexity O(log n) in an average case and in worst case too [1][2].The best case complexity order is equivalent to O(1) . It follows Divide and Conquer approach. The necessary and sufficient condition for this searching technique it the working array must be sorted [3]. The concept is to divide the array into two sub arrays. Initially do it with the help of the highest value (last element if ascending sorted order has been followed) and with least value i.e. first value of the array as per discussed sorting mechanism [3] Under binary search the target element is searched with comparing the middle element of the sorted array[4]. If it matches with the element which we are searching for, the index value of that matched element will be given out else it is noticed that whether this middle element is larger than the element which we are searching or smaller. If the middle element is larger than our value then the searching will be performed again in the left sub array of the main array, else we approach towards right sub array. These iterations will be continued until we get the key or either we met with the last or first element of the array [5]. This concludes that if we are searching any key element in a given array so the number of comparison will be either less than or equal to the half of the number of elements in array. Thus the comparisons taken will be less with respect to the linear search [6][7]. The thing which cumulatively affects the time taken to the complete execution is the processing speed and number of processing elements executing simultaneously [7]. The invention of single instructions multiple data approach beats vector and array processors on certain features specially dealing capacity of multiple instructions at a time. The need of time reduction is truly mandatory in contrast to modern data and performance scenario. Since the need of storage capacity is growing exponentially thus there is always a demand of performance improvement in time, space, order, iterations and recursion etc. In sequential processing the number of elements those can work together will be fixed and only one while it may be two or more than two are available for parallel processing. Parallel execution of any program can be achieved by assigning the work to more than one processor and/or by multithread based execution of program. Here in binary search there is a necessary condition it tells all the elements must be in a sorted array so the number of comparison will be equals to the number of elements available in the array at worst case. Here in this paper we are comparing the performance of binary search for threads one, four and eight threads based iterative and recursive version of searching. Remaining sections of papers are as follows, Section II discusses the related work done towards binary search and parallel processing. Section III contains the hypothesis were taken in part for implementation and results verifications. Section IV tells about the flow of execution and methodology followed, here in this section hybridization of binary searching and parallel threads processing can be seen. Section V contains results and discussion. Where various tables and charts according to the implementation were attached those help to derive conclusion. Sections VI concludes the study and analysis done in earlier sections and gives a direction of future work associated with this study.

II. RELATED WORK

Chanchal P. et.al has emphasized in research titled as "Binary Search Algorithm" mentioned that when and how binary search works for best output i.e. searching. In the paper author claimed that binary search is best suitable when we deal with tight memory space and searching key element in a sorted array (data set) [1]. Authors Concluded in binary search can be used for searching a key element in sorted array of integers having size 400 bytes. But if B-tree or RB- tree have been followed, then dependency on memory increases rapidly. Tell S.V.Sridhar in [8] has done job in comparing linear and binary search.

Researchers had concluded that the order of binary search is logarithmic which need less computation and fewer complexes with respect linear which has order of complexity as O(n). Authors discussed the range searching problem by referencing to Knuth "No really nice data structures seem to exist" for searching in range. Research titled as Odd Even based Binary Search, mentions that Karthick S. et.al has tried to improve the performance of the searching technique [9]. When searching a key element. Binary search basically uses whole set of data for searching but here according to authors no need to do compare with all data values. Just filter them by even and odd parameter. Scenario emphasizes, if there are m total elements on which m are odd in array then for searching any even value by binary search the complexity will be log(n) while in proposed algorithm it'll reduced by O(log m) . So the complexity will be O(log(n – m)) [9]. In this algorithm worst case performance of this algorithm will be close to average case or best case as of binary search. Thus it reduces the number of comparisons, resources needed and complexities as well. Zingtao Duon has co-related the parallel processing with the image intelligent systems in. Image processing was always a very interesting domain. Image processing includes pattern recognition, matching, generation, plotting etc [10].

III.METHODOLOGY

A. Traditional Method of Binary Search

Universal iterative procedure for the binary search is as following –

Let in target of n elements array S with records/elements $S_0 \ldots S_{n-1}$, sorted in such a way so that $S_0 \leq \ldots \leq S_{n-1}$, and Key value K, the following routine uses binary search to find the index of K in S. If t_n is the number of threads allotted to accomplish the task. According to traditional approach -

1) Give sorted array S.
2) Fix *Left* to 0 and *Right* to $n - 1$.
3) If *Left* > *Right*, the search finishes as unsuccessful.
4) Set m (the position of the middle element) to the floor (the largest previous integer) of $(L + R)/2$.
5) If $S_m < K$, set Left to $m + 1$ and go to step 2.
6) If $S_m > K$, set Right to $m - 1$ and go to step 2.
7) Now $S_m = K$, the search is done; return m.

These iterations keep track of the search boundaries via two variables. Some implementations may place the comparison for equality at the end of the algorithm, resulting in a faster comparison loop but costing one more iteration on average.

B. Proposed method for Parallel Binary Search

1) Give sorted array S.
2) Allocate the threads Let t_n is the number of threads.
3) Fix *Left* to 0 and *Right* to $n - 1$.
4) If *Left* > *Right*, the search finishes as unsuccessful.
5) Set m (the position of the middle element) to the floor (the largest previous integer) of $(L + R)/2$.
6) If $S_m < K$, set Left to $m + 1$ and go to step 2.
7) If $S_m > K$, set Right to $m - 1$ and go to step 2.
8) Now $S_m = K$, the search is done; return m.

The flow chart for sequential binary and Parallel search is discussed here in Figure 1and 2.

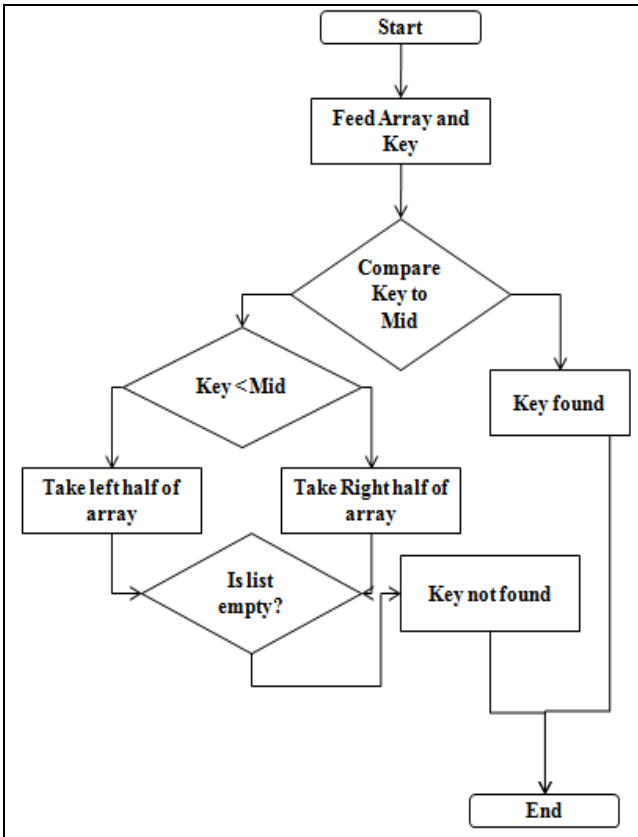


Figure 1 Flow of binary search in Sequential processing

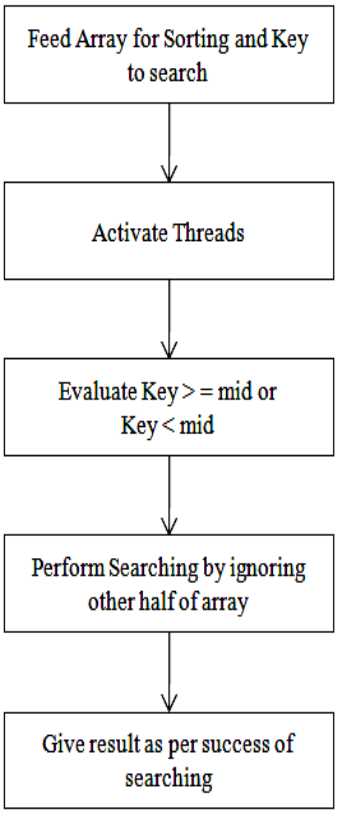


Figure 2 Flow of binary search in parallel processing

IV. RESULT AND DISCUSSION

Here in this section we have discussed about the results and tables achieved after the implementation of traditional/sequential processing based approach and by parallel processing also.

A) Sequential Processing Based Results and Performance Evaluation

1) Sequential Processor based binary searching recursive version number of elements and time taken for search is discussed here in Table1.

Table-1: Time taken by Sequential search in recursive version

Array size (Number of elements)	Time taken for searching elements in ms		
	First Element	*Mid Element*	*Last Element*
100	1.35	0.604	1.106
500	1.28	0.604	1.327
1000	2.06	0.604	1.332
1500	2.342	1.617	1.384
2000	2.347	1.69	2.29
2500	2.549	2	2.789
Average Time	1.988	1.1865	1.704667

Here in Figure -3 the data from Table -2 has been plotted which tells about the comparative time taken by sequential processing of recursive binary search for First, Middle and Last element searching.

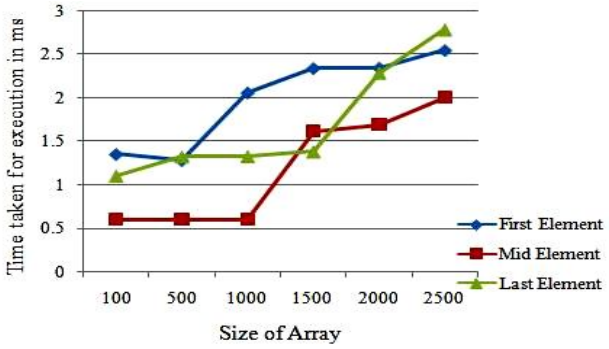


Figure 1. Time taken Vs No of elements Sequential recursive version plot

2) Sequential Processor based binary searching recursive version number of elements and time taken for search in (ms) discussed here in table 2.

Table 2 represents the time taken in recursive code of searching. Few values are atomic and have very less difference with iterative versions but some are really countable.

Table-2: Time taken by Sequential processors for searching in recursive version

Size of Array	Time taken for searching elements in ms (mili second)		
	First Element	Mid Element	Last Element
100	1.5	0.612	1.37
500	1.527	1.38	1.327
1000	1.527	1.381	1.332
1500	1.529	1.407	1.384
2000	1.619	1.379	2.29
2500	1.632	1.415	1.49
Average Time	1.555667	1.262333	1.532167

Here in Figure 4 the data from Table.1 has been plotted which tells about the comparative time taken by sequential processing of iterative binary search for First, Middle and Last element searching.

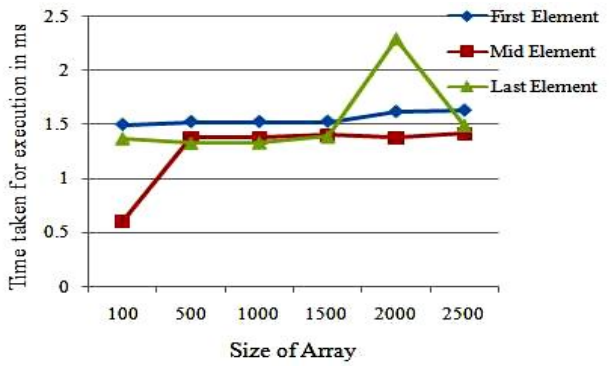


Figure 2. Time taken Vs No of elements Sequential iterative version plot

B) Parallel Four Threads Based Searching Results and Performance Evaluation

1) Thread-four based binary searching iterative version number of elements and time taken for search in (ms) has been discussed here in table-3 As a tendency and desire is to be concerned there is always a prediction that performance will improve with respect to sequential processing of binary search since number of threads are working here are greater than

sequential. After the table Figure-5 plots the data from Table-3, which tells about the comparative time taken by thread four based iterative binary search for First, Middle and Last element searching.

Table-3: Time taken in Iterative version of Thread-4 based binary search

Size of Array	Time taken for searching elements in ms		
	First Element	*Mid Element*	*Last Element*
100	1.11	1.1	1
500	1.15	1.2	1.27
1000	1.13	1.24	1.285
1500	1.34	0.6	1.3
2000	1.47	2	2.1
2500	1.56	1.3	2.22
Average Time	1.293333	1.24	1.529167

Figure 3. Thread 4 iterative version plot time taken Vs no of elements

2) Thread-four based binary searching recursive version number of elements and time taken for search in (ms) is discussed in Table -4

Table-4: Time taken in Recursive version of Thread-4 based binary search

Size of Array	Time taken for searching elements in ms		
	First Element	Mid Element	Last Element
100	0.9	0.58	0.85
500	1.13	0.6	2.3
1000	1.88	0.61	1.27
1500	1.357	0.63	1.69
2000	1.25	6.4	1.38
2500	1.32	6.5	1.66
Average Time	1.306167	2.553333	1.525

Here in Figure-6 the data from Table-4 has been plotted which tells about the comparative time taken by thread four based recursive binary search for First, Middle and Last element searching.

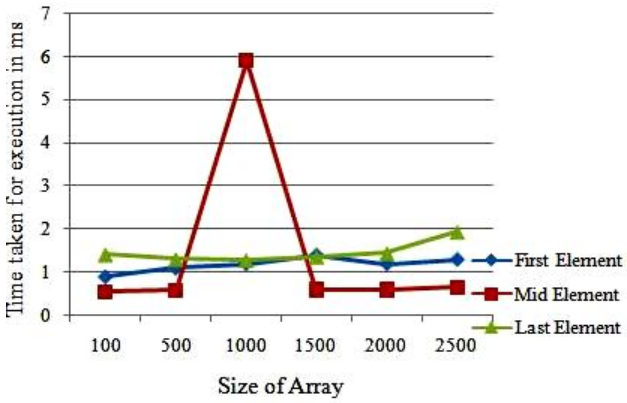


Figure 4. Thread 4 recursive version plot time taken Vs no of elements

3) Parallel Eight Threads Based Results and Performance Evaluation

4) Table-5 tells about Thread-8 based binary searching iterative version number of elements and time taken for search in (ms)

Thread eight means, all the threads are executing simultaneously for finding the solution. Numbers of thread are expected to improved performance.

Table-5: Time taken in eight threads based iterative version of binary search

Size of Array	Time taken for searching elements in ms		
	First Element	Mid Element	Last Element
100	0.7	2.61	1.25
500	1.2	1.24	1.23
1000	1.32	1.14	1.26
1500	1.333	1.39	1.35
2000	1.34	1.26	1.25
2500	1.35	1.27	1.3
Average Time	1.207167	1.485	1.273333

Here in Figure-7 the data from Table-5 has been plotted which tells about the comparative time taken by thread eight based iterative binary search for First, Middle and Last element searching. Here

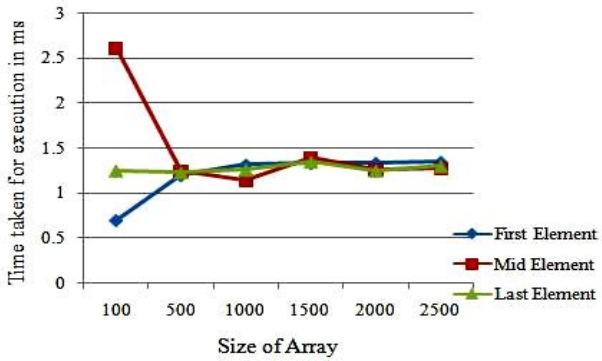


Figure 5. Thread 8 based recursive version plot time taken Vs Number of elements

5) Table for Thread-8 based binary searching recursive version number of elements and time taken for search in (ms) is attached below .Description of plotting the data of table 6 is given in figure 8.

Table-6 : Time taken in eight threads based recursive version of binary search

Size of Array	Time taken for searching elements in ms		
	First Element	*Mid Element*	*Last Element*
100	0.9	0.56	1.4
500	1.1	0.59	1.29
1000	1.2	0.59	1.27
1500	1.4	0.6	1.35
2000	1.2	6.1	1.43
2500	1.3	0.65	1.93
Average Time	1.18334	1.485	1.445

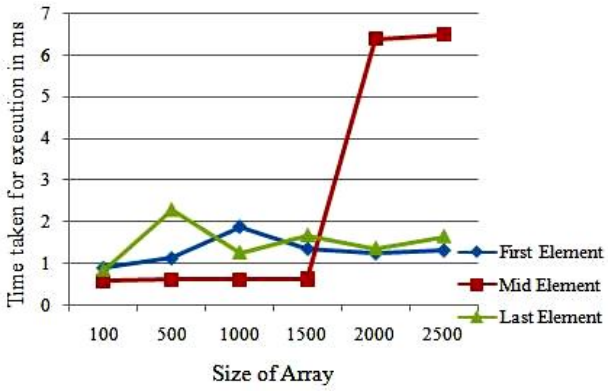


Figure 6. Thread 8 iterative version plot time taken Vs number of elements

Here in Figure-8 the data from Table-6 has been plotted which tells about the comparative time taken by thread eight based iterative binary search for First, Middle and Last element searching. in graph an ambiguous behaviour of mid element searching can be seen it's due to employing some unnecessary threads. According to model eight threads have been allocated for doing the task but element meet after first comparison thus thread allocation become overheads here.

6) Comparative analysis based results

7) Analysis of iterative binary searching for all (one, four, eight threads based) conditions

Comparative time taken(in ms) on an average for searching first, last and mid element, for each 1-4-8 threads based binary searching Iterative Version has been discussed here in Table-7

Table-7 : one, four and eight thread based iterative version Comparative performance of binary search.

Number of Threads	Time taken for searching elements in ms		
	First Element	Mid Element	Last Element
Thread=1	1.555667	1.262333	1.532167
Thread=4	1.293333	1.24	1.529167
Thread=8	1.202072	1.485	1.273333

Here in Figure-9 the data from Table-7 has been plotted which tells about the comparative time taken by thread one, four and eight based iterative binary search for First, Middle and Last element searching.

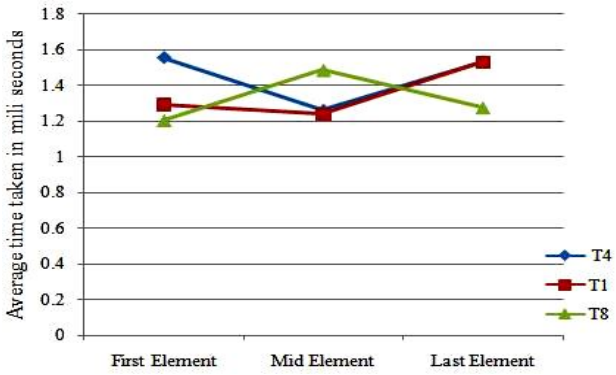


Figure 7. Thread1, 4 and 8 iterative version plot time taken Vs number of elements

8) Analysis of recursive binary searching for all (one, four, eight threads based) conditions

Comparative times taken (in ms) on an average for searching first, last and mid element for each One, four and eight thread based binary searching recursive Version is discussed here in Table-8

Table-8: One, four and eight thread based recursive version Comparative performance of Binary Search.

Number of Threads	Time taken for searching elements in ms		
	First Element	Mid Element	Last Element
Thread=1	1.988	1.1865	1.704667
Thread=4	1.30617	2.55333	1.525
Thread=8	1.18333	1.485	1.445

Here in Figure-10 the data from Table-8 has been plotted which tells about the comparative time taken by thread one, four and eight based recursive binary search for First, Middle and Last element searching

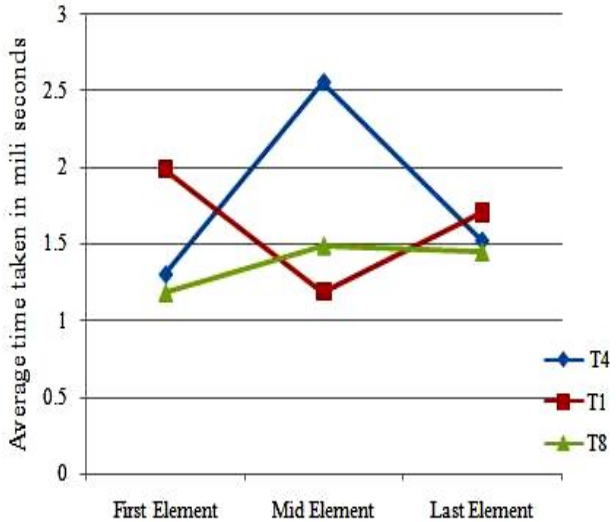


Figure 8. Thread1, 4 and 8 Recursive version plot time taken Vs no of elements

V. CONCLUSION

Our objective was to discuss issues and performance analysis of binary search under sequential and parallel processing .It is clear from the result section that for recursive code in long run, for large set of data, eight thread based searching takes,

minimum time for first and/or last element searching. While For Iterative code in long run for large set of data, four thread based and sequential searching both perform similarly, while eight threads based searching takes minimum time for first and/or last element searching. If our value ranges at mid value searching in iterative version 8 threaded searching performs worst while in recursive version 4 threaded searching gives poorest performance. So for large set of data the best suited algorithm for searching either first or last element will be eight threads based searching. The premier issues towards binary search were dependencies among instructions and thread communications those can be mentioned as

If there is variable number of elements in an array then we can't judge how many cores of processors needed to employee to gain optimize computation Vs communication ratios (in case of stream of data).

Although research enables us to direct or to make decision about, selection of thread and version of binary search for optimized results; but still there is need to invent size of grain and grain packing.

Since performance improvement is all time favourable area of enhancement thus this study gives a direction for future research in order to minimize the time taken for searching.

REFERENCES

[1] A.Oommen ,C. Pal, "Binary Search Algorithm", IJIRT 100392 Publisher,India, Volume 1,Issue 5 ,2014.

[2] V.P.Parmar,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" IJCA,USA, Volume 121, IssueNo.3, July 2015.

[3] B.Hon, Y.Lu, "Research on optimization and parallelization of Optimal Binary Search Tree using Dynamic Programming", International Conference on Electronic & Mechanical Engineering and Information Technology ,China, EMEIT-2012

[4] K. Zamanifar, M. Koorangi, "Designing Optimal Binary Search Tree Using Parallel Genetic Algorithms", Publisher (IJCSNS) International Journal of Computer Science and Network Security, Korea, Vol.7 Issue 1, January ,pp.138, 2007

[5] P.Suri, V. Prasad,"Binary Search Tree Balancing Methods: A Critical Study",IJCSNS International Journal of Computer Science and Network Security, Korea,Vol.7,Issue.8,pp. 237-244, 2007.

[6] A. R. Chadha, R. Misal,T. Mokashi, "Modified Binary Search Algorithm",International Journal of Applied Information Systems (IJAIS),Foundation of Computer Science FCS, New York, USA,Vol.7, Issue. 2, pp. 37-41, 2014

[7] C.Marttnez, S. Roura, "Randomized Binary Search Trees", Journal of the ACM, Vol. 45, Issue. 2, pp. 288 –323,1998.

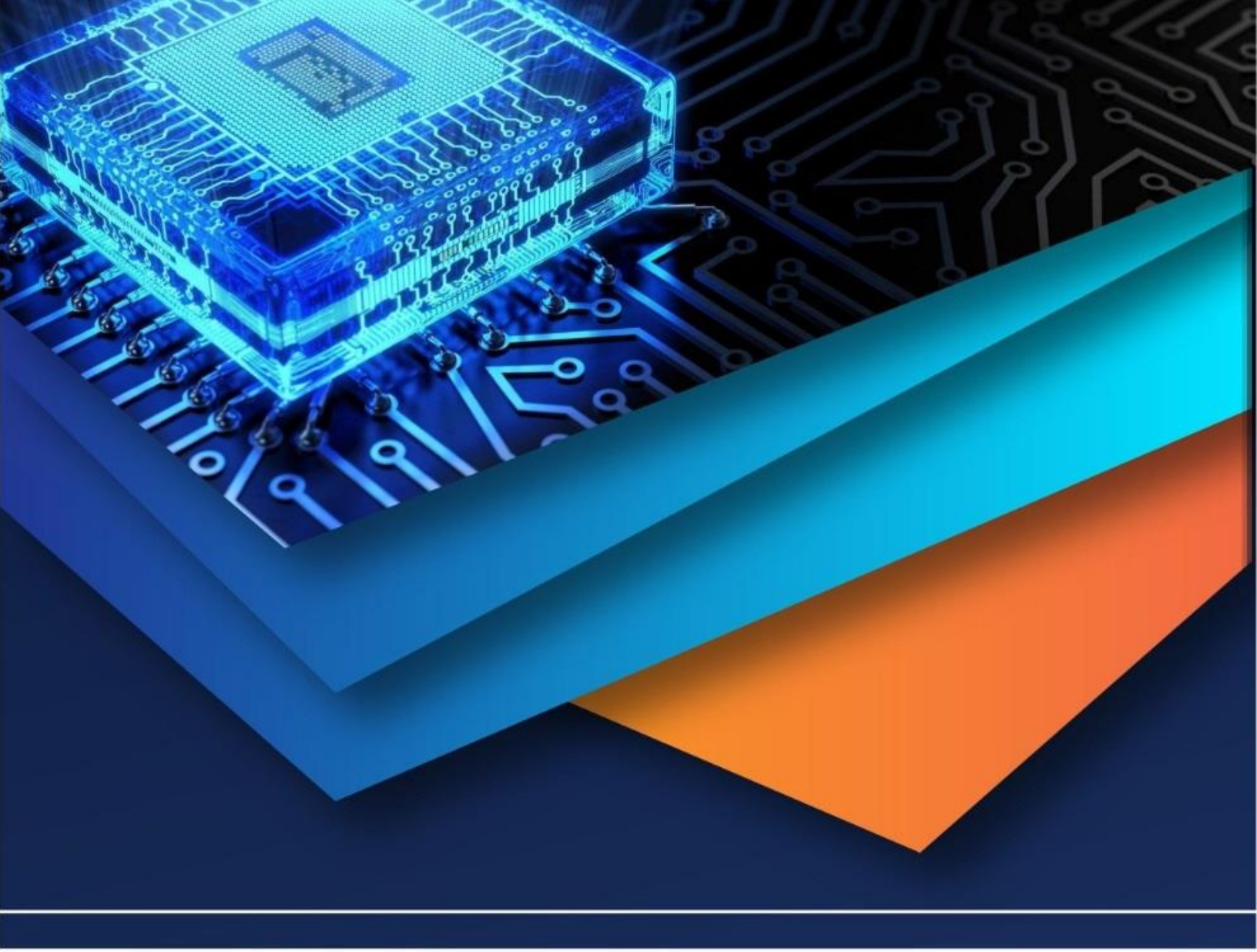
[8] S.V.Sridhar, P.R. Rao, E.PriyadarshinI, N.V.Krishna,"Comparison of Search Data Structures and Their Performance Evaluation with Detailed Analysis" International Journal of Scientific & Engineering Research, Volume 4, Issue 4, pp.1437-1446, 2013.

[9] S.Karthik, "Odd Even Based Binary Search", International Journal of Computer Engineering & Technology (IJCET),Volume 7, Issue 5, pp. 40–55,2016.

[10] Z. Duon,T.Lei, h. Fan "Parallel Computing system for image Intelligent Processing ", Asia Network of Scientific Information, Information Technology Journal, China,2012.

[11] C.Kathuria,G. Datta,V.Kaul,"Context Indexing in search Engine Using Binary Search",International Journal on Computer Science and Engineering (IJCSE),Vol. 5, Issue.6,pp. 514-521,2013.

[12] A.M.Berman,"Data Structure via C++",Oxford University Press Publication, New York,USA, pp.108-113,1997.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY