ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 4 No. 2 December 2018*

Effects of Multi-core Processors on Linear and Binary Search Algorithms

[1]Abbas Muhammad Rabiu*, [2]Ahmed Baita Garko, [3]Aisha Muhammad Abdullahi
[1,2,3] Department of Computer Science,
Federal University, Dutse (FUD)
Email: mambas86@yahoo.com

Abstract
Searching is a problem that commonly arises in computer science. It is a process of checking and finding an element from the list of elements. Many search algorithms have been developed while existing ones have been improved upon to enhance their performance in terms of computational complexity, memory and other factors. In this paper the effect of multi-core processors on linear and binary search algorithms was investigated using System.nanotime() benchmark suits as the main method used; and some of java concurrency tools to measure the running times of linear and binary search algorithms on a single and multi-core machines. The results obtained shows that binary search is 1.98 times faster than linear search on a dual-core and 3.83 times faster on quad-core machine. It has thus been concluded that binary search is a better searching algorithm as compared to the linear search. In addition, increase in the number of processing cores significantly improves on the performance of both algorithms.

Keywords*: Algorithm, Binary search, Linear search, Multi-core machine, Time Complexity.*

1. INTRODUCTION
Two of the most popular search algorithms are Linear and Binary search. Linear search (also known as sequential search) searches for an element in a given array sequentially while binary search in contrast is based on divide and conquer approach (Knuth, 1997). Given the vast amount of data available, search algorithms are inevitable part of programming. Tag feedback based sorting algorithm for social search implements searching on the World Wide Web (Zhuoer *et al.,* 2011) and the backwards search algorithm by (Zongli, 2010) is fundamental to retrieval of information in the full text model. But both linear and binary search algorithms form the basis of many search applications. Linear search has the time complexity of O(n) while binary search has a complexity of O($logn$) (Knuth, 1997; Thomas, 2004). The improvement in performance gained by the use of multi-core processor depends very much on the nature of algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can run in parallel simultaneously on multi-cores; this effect is described by Amdahl's law which states that in parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speed-up that can be achieved using N number of processors is $1/((1-p)+(\frac{p}{N}))$ (Techopedia, 2018). If N tends to infinity then the maximum speedup tends to 1/(1-P). In the best case, so called embarrassingly parallel problems may realize speed up factors near the number of cores caches(s) avoiding used of much slower main system memory. Most applications, however,

*Author for Correspondence

are not accelerated so much unless programmers invest a prohibitive amount of effort in re-factoring the whole problem (Acter, 2011; Thomas, 2004). This paper will take advantage of multi-core CPU to implement the two algorithms of choice using concurrency tools provided by Java. Therefore, by measuring and comparing the running time of the two algorithms on both single and multi-core machines, the performance benefit of multi-core over single-core machine can be seen. The parallelization of a software algorithm is a significant ongoing topic of research.

1.1 STATEMENT OF THE PROBLEM

Searching is a problem that commonly arises in computer science. Many search algorithms have been developed while existing ones have been improved upon all to make them run faster. According to (Sengupta *et al*., 2007) efficiency of algorithms can be measured in terms of execution time (complexity) and amount of memory required. Computer architectures are approaching physical and technological barriers which make increasing the speed of a single core exceedingly difficult and economically infeasible. As a result, hardware architects such as AMD and Intel have begun to design microprocessors with multiple processing cores that operate independently and share the same address space (Fasiku *et al*., 2012). Therefore to reduce the running time of a given algorithm, programmers have to take advantage of the multi-core processor machines and some of the concurrency tools provided by java to gain an improved efficiency of their algorithms. Therefore, this paper analyzes these two situations using Linear and Binary search algorithms by measuring their running times on both single and multi-core processor machines to show the advantage of multi-core over single-core machines and to compare the performances of binary and linear search algorithms.

2. RELATED WORK

A modification to the traditional binary search algorithm was proposed by (Ankit and Chadha, 2004) in which it checks the presence of the input element with the middle element of the given set of elements at each iteration. Modified binary search algorithm optimizes the worst case of the binary search algorithm by comparing the input element with the first & last element of the data set along with the middle element and also checks that the input number belongs to the range of numbers present in the given data set at each iteration thereby reducing the time taken by the worst cases of binary search algorithm. The result is that the modified binary search improves the execution time vastly over traditional binary search. The algorithm is comparatively more efficient as it eliminates unnecessary comparisons at the preliminary stage itself. However, this algorithm can even be extended to include the String domain. An analysis of both linear and binary search algorithms is presented in (Vimal and Kumbharana, 2015), which shows to some extent the applicability and execution efficiency of the algorithms. It also analyzes few data structures to implement these algorithms. At last based on the linear search and binary search algorithms, one algorithm is designed to function on linked linear list. The result is that the binary search is more efficient searching technique than linear search but insertion of an element is not efficient as it requires arranged elements in specific order. Furthermore, it is also possible to apply binary search on linked list by making necessary modifications to original binary search algorithm. It is concluded that the selection of searching algorithm can be based on the data structure on which it is applied and which operations are required more. Therefore, balance is required between search and maintenance of a data structure. Additionally, presented by Kumbharana and Vimal (2015) is an investigation of the effect of parameters n and p of a Binomial distribution input on the number of comparisons in linear search and binary search. Using factorial experiment, it is observed that both the main effect n and p and the interaction effects n*p are highly significant for linear but comparatively less

significant for binary search. The result clearly suggests that apart from the size of the input, the parameters of the input distribution need also be taken into account to explain the behavior of certain search algorithms. However, this paper compares the execution time of searching for an integer number among the list of a given integer numbers using linear and binary search algorithms on single and multi-core machine. System.nanotime()benchmark suit and some concurrency tools provided by Java were used to measure the running time. The emergence of this paper has established the fact that sorting time of the input before a binary search is conducted is never considered. Whereas, if sorting time is added to a binary search time, then execution time of the binary search is not faster than that of the linear search technique.

3. METHODOLOGY

To benchmark the algorithms in this paper, the main method used is the practical measurement of run time. In Java there are currently two built-in functions which let the user retrieve time: System.currentTimeMillis() and System.nanoTime(). System.currentTimeMillis() is based on computer system's clocks which has some weaknesses. The system clock is not perfect, it may drift off sometimes and occasionally needs to be corrected. Because of this weakness associated with CurrentTimeMillis, System.nanoTime() is the preferred method used to measure time in this paper. System.nanoTime() gives more precise results on Windows Operating System but it is actually slower than System.currentTimeMillis(). When this function is called it returns a number in nanoseconds from a fixed but arbitrarily chosen point. This time is then converted to milliseconds and compared for convenience. Two algorithms namely: Linear and Binary searching algorithms are analyzed as follows:

3.1 BINARY SEARCHALGORITHM

Binary search, also known as half-interval search or logarithmic search is a search algorithm that finds the position of a target value within a sorted array. It compares the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful. Binary search runs in at worst logarithmic time, making O(logn) comparisons in the worst case, O(1) in the best case, where n is the number of elements in the array and log is the binary logarithm; and using only constant space. Although specialized data structures designed for fast searching such as hash tables can be searched more efficiently, binary search applies to a wider range of search problems. Binary searches require the collection to be sorted. The brief algorithm for Binary sorting algorithm is given below.

3.1.1 BINARY SEARCH PSEUDO-CODE

```
X = number to be searched, a[] - elements array , 'n' total number of
elements
1. low=0 , high= n-1
2. while(low<high)
3. mid=(low+high)/2
4. if(a[low]>X OR a[high]<X)
5. return -1
6. end if
7. if(a[low]==X)
8. return low
9. else if(a[high]==X)
10. return high
11. else
12. if(a[mid]==X)
13. return mid
14. else if(a[mid]>X)
15. high=mid-1
16. low++
17. else if(a[mid]<X)
18. low=mid+1
19. high - -
20. end if
21. end if
22. end while
23. return -1
```

3.1.2 TERMS USED:

Search() – function call to check whether the given element is present or not.
Low – the lowest index in the array.
High- highest index in the array.
Middle – middle element's index in the array.
X–element to be searched.

3.1.3 PERFORMANCE ANALYSIS OF BINARY SEARCH ALGORITHM:

Table 3.1 and Fig. 3.1show the results obtained when searching for an element using binary search algorithm for a given number of array size ranging from 40 to1400 elements. Binary search exhibits best performance on quad-core as its running time continuously decreases when compared with the results obtained on both single and dual-core machines.

Table 3.1: Running time of Binary search on single-core, dual-core and quad-core machines

Array size	Running time of Binary Search Single-core(ms)	Running time of Binary Search Dual-core(ms)	Running time of Binary Search Quad-core(ms)
50	0.0190	0.0090	0.0045
100	0.0280	0.0130	0.0066
200	0.0330	0.0160	0.0081
400	0.0440	0.0220	0.0012
800	0.0640	0.0310	0.0155
1200	0.0830	0.0410	0.0206
1400	0.1280	0.0640	0.0321

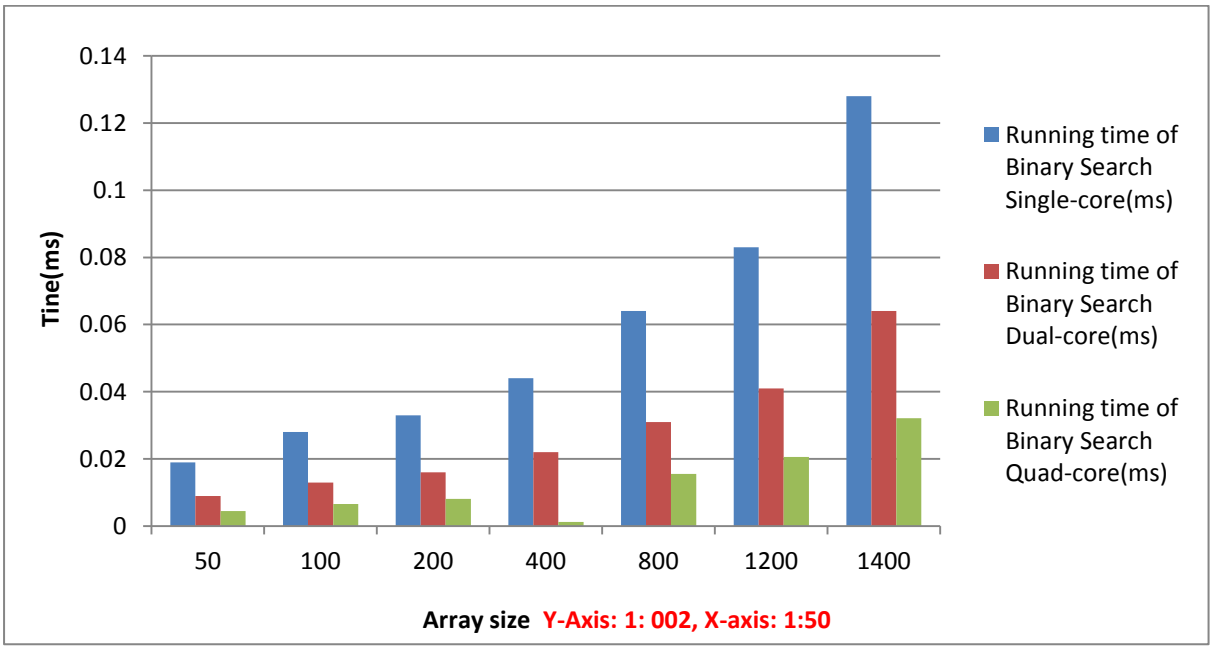

Fig. 3.1: Running time of Binary search on single-core, dual-core and quad-core machines

3.2 LINEAR SEARCH ALGORITHM

Linear search also known as sequential search is a method for finding a particular value in a list that consists of checking every one of its elements one at a time and in sequence, until the desired one is found. Linear search is the simplest search algorithm. For a list with n items, the best case is when the value is equal to the first element of the list, in which case only one comparison is needed. The worst case is when the value is not in the list (or occurs only once at the end of the list), in which case n comparisons are needed. The worst case performance scenario for a linear search is that it has to loop through the entire collection, either because the item is the last one, or because the item is not found. In other words, if you have N items in your collection, the worst case scenario to find an item is N iterations. In Big O Notation it is O(N). The speed of search grows linearly with the number of items within your collection. Linear search does not require the collection to be sorted. A brief algorithm for linear sort is given below.

3.2.1 LINEAR SEARCH PSEUDOCODE

ai= number to be searched, a[] - elements array ,

'n' total number of elements, b= last element

```
1: Linear search (a1,…,an: int a, b);
2:  I = 1;
3: while (ai !b and b<n);
4: i = i+1;
5: if (ai = b) return I;
6: else return zero.
```

How it works

1. Given an array elements A[i] to A[n].
2. If the first element in the array is the desired element then return its index.
3. If the desired element is not the last element of the array but is less than the last element then continue searching
4. If the desired element is the last element then return its index.
5. Else the element is not in the array.

3.2.2 PERFORMANCE ANALYSIS OF LINEAR SEARCH ALGORITHM:

Table 3.2 and Fig. 3.2show the results obtained when searching for an element linearly in a given number of array size ranging from 40 to1400 elements. Linear search performs far better on quad-core machine thereby recording shortest running time when compared with the result obtained on single and quad-core machine.

Table 3.2: Running time of Binary search on a single, dual and quad-core machine

Array size	Running time of Linear Search Single-core(ms)	Running time of Linear Search Dual-core(ms)	Running time of Linear Search Quad-core(ms)
50	0.03	0.01	0.01
100	0.04	0.02	0.01
200	0.07	0.03	0.02
400	0.09	0.05	0.02
800	0.12	0.06	0.03
1200	0.13	0.06	0.03
1400	0.23	0.11	0.06

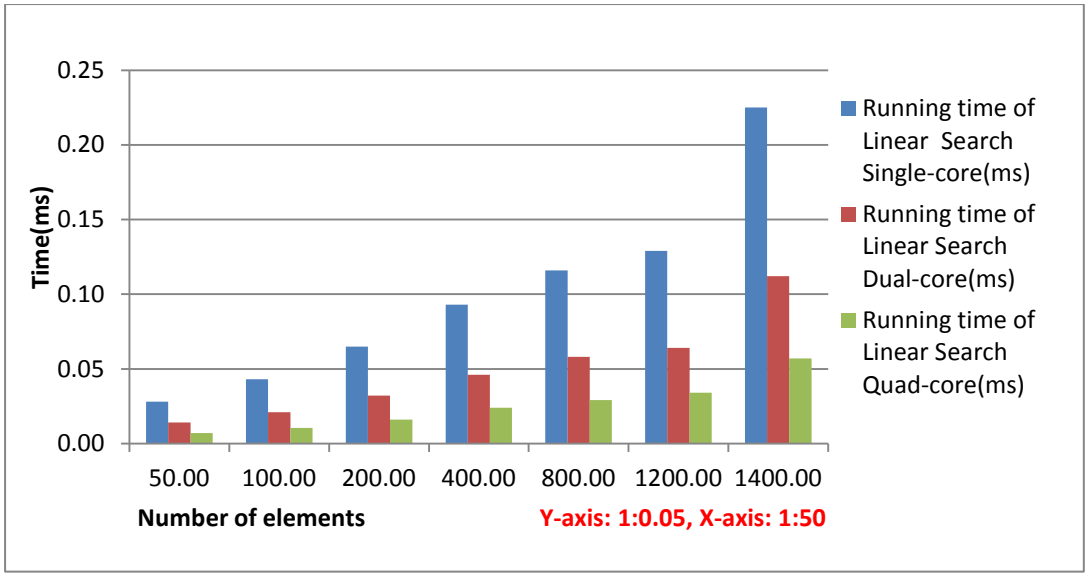

Fig.3.2: Running time of Binary search on a single-core, dual-core and quad-core machines

3.3 HARDWARE AND SOFTWARE SPECIFICATIONS

For the development and benchmarking of linear and binary algorithms used in this paper the hardware specifications shown below in Table 3.1, Table 3.2and Table 3.3 were used.

Table 3.1: Single-Core Processor Specification

CPU	Mobile AMD Sempron™ Single Core Processor 3200+1.60GHz
RAM	1.50 GB
OS	Windows 7 64-bit

Table 3.2: Dual-Core-Core Processor Specification

CPU	Intel(R) Pentium® Dual Core CPU T2390 @ 1.86GHz, 1.87GHz
RAM	1.00 GB
OS	Windows 7 64-bit

Table 3.3: Quad-Core-Core Processor Specification

CPU	Intel Core i7 920 @ 3.5GHz Quad-Core
RAM	GB DDR3 @ 1333MHz
OS	Windows 7 64-bit / Linux Mint 13 64-bit

4. PERFORMANCE COMPARISM OF LINEAR AND BINARY SEARCH ON ALL THE THREE MACHINES:

In Table 4.1 and the graph shown in Fig. 4.1 below, Y-axis represent the execution time taken by the two algorithms on both single and multi-core machines and the X-axis represents the number of input elements in the array. The input size sorted ranges from 50 to 1400 elements. Each array size is sorted on a single core, dual core and quad-core machines. And for each input size; the time taken to search for an element is recorded and the results were tabulated as seen in the respective tables below.

Table 4.1: Running Time of Linear and Binary Search Algorithms on Single, Dual and Quad-core Machines

Array size	Running time of Linear Search Single-core(ms)	Running time of Binary Search Single-core(ms)	Running time of Linear Search Dual-core(ms)	Running time of Binary Search Dual-core(ms)	Running time of Linear Search Quad-core(ms)	Running time of Binary Search Quad-core(ms)
50	0.0280	0.0190	0.0140	0.0090	0.0071	0.0045
100	0.0430	0.0280	0.0210	0.0130	0.0105	0.0066
200	0.0650	0.0330	0.0320	0.0160	0.0160	0.0081
400	0.0930	0.0440	0.0460	0.0220	0.0240	0.0012
800	0.1160	0.0640	0.0580	0.0310	0.0292	0.0155
1200	0.1290	0.0830	0.0640	0.0410	0.0340	0.0206
1400	0.2250	0.1280	0.1120	0.0640	0.0570	0.0321

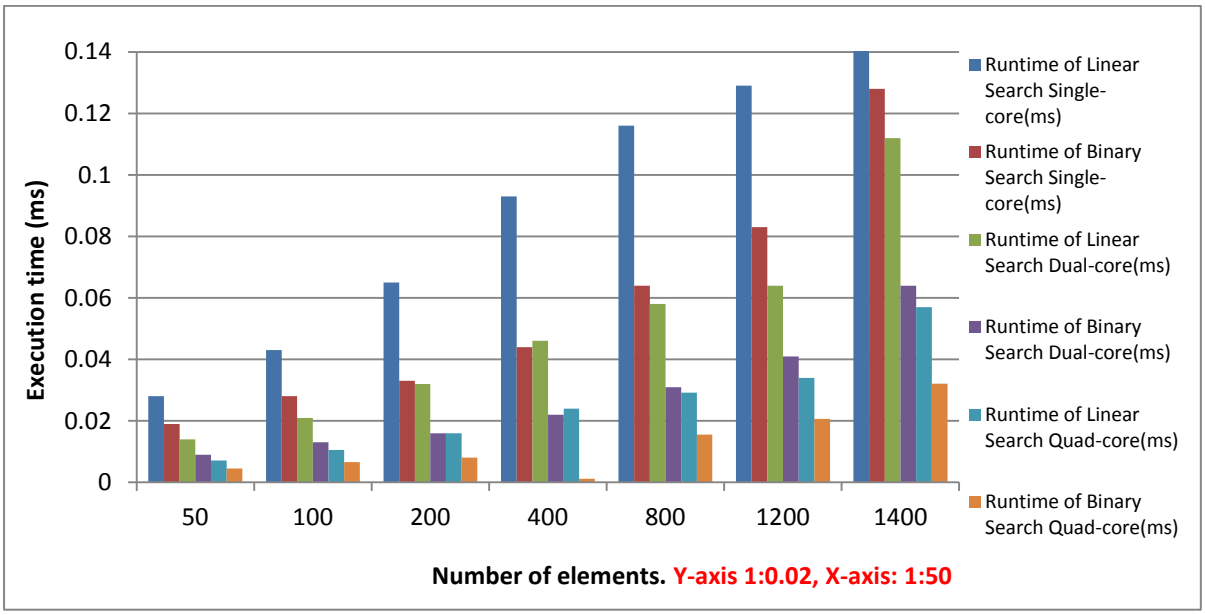

Figure 4.1: Performance comparison of linear and binary search on single, dual, and quad-core machines.

5. DISCUSSION OF RESULTS

From Table 4.1 and Fig. 4.1 above, comparing linear and binary search performances, it is clear that linear search recorded the worst running time throughout the searching processes on single-core, dual-core and quad-core machines while binary search has the best performances. When the array size is small, binary search exhibits better performance than linear search. Similarly, binary search algorithm performs better when searching larger number of elements on all the three machines. The result also shows that running time of binary search recorded on the Dual-core machine is 1.98 times faster than linear search when compared with the results obtained on the Single-core machine. Similarly, running time of binary search algorithm obtained on Quad-core machine is 3.83 times faster than linear search algorithms when compared with the results obtained on a Single core machine and 1.78 faster when compared with the results on a Dual-core machine. Therefore, binary search emerged as the best searching algorithms in this paper. However, for a binary search to be performed the array elements must be completely sorted. Therefore the sorting time of binary search is not considered in this paper. It has been established in this paper that when

the sorting time of binary search is added to the binary search time, the binary search is not faster than linear search. The results also show that machines with more number of processing core have less running time (more efficient) with careful implementation of both the two algorithms.

6. CONCLUSION

This paper discusses two comparison based searching algorithms. It analyses the performance of these algorithms for the same number of elements on a single, dual, and quad-core machines. Binary search on a dual-core machine was found to be 199% faster and almost 400% faster on a quad-core machine when compared with the running time obtained on a single-core machine. It has been concluded however, that binary search is the best searching algorithm without inclusion of its sorting time. And it has also been concluded that machines with more number of processing core(s) has less running time as compared to those with lower number of cores in their processors and hence, have better efficiency. Other popular algorithms such as merge and quick sorts have the potential to show improved performance by using the same approaches. These and other algorithms deserve further research.

REFERENCES

Ankit, R. Chadha (2014). Modified Binary Search Algorithm. *International Journal of Applied Information Systems (IJAIS),* [online], *7*(13), pp.1-4. Available at: http://www.vogella.com/ [Accessed 2nd Feb. 2015].

Fasiku, A., Olawale, B. and Jinadu, T. (2012). A Review of Architectures - Intel Single Core, Intel Dual Core and AMD Dual Core Processors and the Benefits. *International Journal of Engineering and Technology* 2(5), pp.1-9.

Knuth, D. (1997). *The art of computer programming, sorting and searching*.3rd ed.[ebook]. New York: Addison Wasley, pp.395-409

Sengupta, E., Robert S. and Kelvin, W. (2003). *Algorithms in Java.* [online], 3rd Ed. [pdf] New York: Addison Wasley. Available at: http://www.quora.com [accessed 1st Feb. 2015]. pp. 60-70

Techopedia (2017). Amdahl's law. [online],Available at: https://www.techopedia.com/definition/17035/amdahls-law [accessed 29th Oct., 2018).

Thomas, H., Cormen, C., Leiserson, E. and Ronald L. (2004). *Introduction to Searching Algorithms*.2nd Ed. Prentice-Hall: New Delhi, pp.120-144.

Vimal, P. and Kumbharana, C. (2015a). 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), p.1-4.

Vimal, P. and Kumbharana, (2015b).Comparing Linear Search and Binary Search Algorithms. *International Journal of Computer Application,* [online], *121*(3), P.1-16.Available at: http://www.research.ijcaonline.org/comparing / [Accessed 20th Mar. 2016].

Zongli, J., (2010). A tag feedback based sorting Algorithms for Social Search. In: *International conference on System and Informatics (ICSAI2012),* 3(2-4), p.12-32.

Zhuoer, L., Chenghong Z. and Yunfa H., (2011). Backwards Search Algorithm of Double-Sorted Inter-relevant Successive Trees. *Fifth International Conference on Fuzzy Systems and Knowledge Discovery,* [online],*3(23),* 2-12. Available at: http://researchgate.net [Accessed 20 Apr. 2016].