Journal of Science Research

Energy consumption, cyclomatic and time complexities of variants of quicksort algorithm: A comparative study

Akinola, S. O.* and **Owoeye, F. O.**
Department of Computer Science
University of Ibadan
*Corresponding author: solom202@yaho.co.uk

Abstract

The increasing demand for sorting big data, software quality and proliferation of battery-powered computing devices require efficient sorting algorithms and performance metrics to measure energy consumption and cyclomatic complexity. The introduction of a faster two- pivot quicksort by Yaroslavskiy in 2009 displaced one pivot-quicksort in Java Runtime Library 7.However, the two-pivot quicksort algorithm is less efficient for sorting small data sizes. In addition to the Hoare's one-pivot and Yaroslavskiy's two-pivot quicksort algorithms, the Yaroslavskiy's two-pivot quicksort technique was modified and implemented with java programming language in this study. Randomized, unsorted data was generated into arrays ranging from small to large sizes and were subjected to Quicksort algorithm variants up to five times each. The average time spent by the algorithms to sort the data as well as their energy consumed for the sorting were recorded. The cyclomatic complexities of the algorithms were computed. The modified two-pivot was found to be time and energy efficient for all the data sizes. There was a high correlation between time and energy consumed with energy increasing linearly with time. Hoare's one pivot quicksort has the best software complexity followed by the modified Yaroslavskiy two-pivot quicksort algorithm. The modified two-pivot quicksort is recommended for sorting applications deployed on computing devices where time and energy are critical factors.

Keywords: Cyclomatic complexity; energy and power consumptions of algorithms; quicksort algorithm.

Introduction

Analysis and design of algorithms are part of the building blocks of computer science. Despite the design of new sorting algorithms, Quicksort is used in many practical implementations. It is one of the fastest comparison-based sorting algorithms and the standard sorting method in UNIX, Java and C libraries. The performance of the algorithm is dependent on the quality of selected pivot. Until recently, one pivot or key element was normally used to partition the list of data to be sorted into two regions. After partitioning the list, the values of elements in the first region are lesser or equal to the pivot while those at the second region are greater or equal to the pivot. The pivot is chosen either as the first, last, middle element or randomly picked from the list. Each of the choice of the one pivot has its associated turn on the performance of the quicksort algorithm [1].

In 2009, a two-pivot quicksort algorithm was proposed by Vladimir Yaroslavskiy [2, 3] to improve the performance of one-pivot quicksort. The introduction of the faster two-pivot quicksort displaced one pivot-quicksort in the Java Runtime Library 7. The two-pivot quicksort function is called from Java runtime library to sort both large and small array sizes. The improved execution time in two-pivot quicksort was attributed to selection of the two pivots and reduced number of comparison than one-pivot quicksort [4]. Five sampled elements were picked from the array and arranged into an ascending order. He selected the second and fourth elements of the sampled elements as pivots. The two selected pivots partitioned the array into three and recursively sorts the array by moving elements less than first pivot to the left, elements greater than the second pivot to the right while elements greater than or equal to first pivot and elements less than or equal to the second pivots were moved to the center of the array. Insertion sort was employed to sort tiny array size. Empirical results from using this approach showed an improved performance in execution time particularly for large array sizes over the one-pivot quicksort technique.

To meet the demand for maintenance and high performance in sorting big data, cyclomatic complexity and energy efficiency must be considered as a key factor in the quicksort algorithm. To the best of our knowledge, little or no work has been done to determine the performance of quicksort in terms of its energy efficiency and cyclomatic complexity. Software controls the operations of hardware and directly contributes to the energy dissipated by the computing device. Hence, selecting energy efficient algorithms minimizes the amount of energy consumption by software. Since software controls hardware, recent researches advocate the use of energy efficient algorithms and software for energy efficiency optimization [5, 6].

Battery-powered device users' preferences for energy optimization brings to fore the need to include energy consumption in evaluating the performance of quicksort algorithm. Bunse *et al* [7] presented an approach for selecting optimal sorting algorithm based on users' preference for speed or energy-saving. Considering the power consumption constraints in battery powered devices, high consumption in data centers, Bardine *et al* [8] focus on optimizing hardware for energy efficiency.

Recent research efforts were concentrated on analyzing factors that contribute to the time efficiency of two-pivot quicksort. Wild and Nebel [9] attributed less execution time to reduced number of comparisons. They obtained $1.9n\log n + O(n)$ as the asymptotic number of comparisons for two-pivot quicksort while the classic one-pivot has an asymptotic comparison of $2n\log n + O(n)$. Similar study carried out by Kushagra *et al* [4] corroborates the results stated in [9]. Improving two-pivot for time efficiency when sorting small array size was not considered.

Multiple pivots quicksort algorithm has been previously examined by Salman and Sultanulah [10]. They conducted an empirical study to evaluate the performance of multiple pivot quicksort. The result of the empirical study was compared to quicksort, merge sort and heap sort. The outcome reveals that multiple pivot quicksort outperforms Hoare's one pivot quicksort, merge sort and heap sort. However, the study evaluated only the execution time of the algorithms, leaving out energy and software complexity of the algorithms.

In recent times, emphasis has been laid on the need for high software quality. Cyclomatic complexity indicates bug density in a program, low cyclomatic density is proportional to low bug occurrence [11]. The metric measure understandability and maintainability of software. Alsultanny [12] measured cyclomatic complexity of binary search algorithm implemented in C++, Java and Visual Basic. The study revealed varying complexities between the three programming languages. The author attributed the varying complexities to language flexibility, ease of understanding and syntax. Comparing quicksort algorithm variants using cyclomatic complexity will give insight into the ease of understanding and testing of its implementation.

In the present study, the Yaroslavskiy's two-pivot quicksort was modified by changing the way pivots were selected. The two pivots for quicksort were selected by picking the middle element and a random unique location. An experiment-proof approach was employed to determine if there is an improvement in execution time for sorting small and large array sizes by changing the pivots location. Time, energy consumption and cyclomatic complexity of the quicksort algorithm's variants (one-pivot, Yaroslavskiy's two-pivot and the modified two-pivot quicksort algorithms) were measured and compared.

Materials and methods

This study modifies Yaroslavskiy's two-pivot quicksort algorithm by changing how the two pivots for the algorithm are selected. The original Yaroslavskiy's two-pivot quicksort algorithm and the modified algorithm are presented next.

The original Yaroslavskiy's [2] dual pivot quicksort algorithm

In Yaroslavskiy's two-pivot quicksort algorithm, 'Less', 'great' and 'k' were pointers used to transverse the array. Pointers less and *k* scan an array or list from left to right while pointer great scans from right to left. Pivots **p** and **q** partition the array into three regions. Elements of the array less than pivot **p** are swapped to the leftmost side of the array, elements greater than pivot **q** are swapped to the rightmost side while elements greater or equal to pivot **p** and less or equal to pivot **q** are moved in between pivot one and pivot two. Hence, the array is partitioned into three regions and the partitions are recursively sorted using the same process. Insertion sort was used to sort array size less than or equal to seventeen. The cost of sorting tiny array sizes less than or equal to 17 using two-pivot quicksort is high, hence, insertion sort was used. The algorithm for the Yarslavskiy's two-pivot quicksort partitioning process is presented in Figure 1. The major difference between Yaroslavskiy's dual quicksort and modified two-pivot quicksort is the position of pivots used in partitioning the array. Increased number of pivots results in more pointers traversing the array and increased number of partitions. Quicksort algorithm whether one pivot or two pivots has two things in common, which are array partitioning and recursion.

```
Yaroslavskiy Partition (A, left, right)
less ← left + 1
great ← right – 1
k ← less
while k ≤ great do
        if A[k] < p then
            exchange A[k] ↔ A[less]
            less = less + 1
        End
        else
            if A[k] > q then
                while A[great] > q do
                     great ← great - 1
            End
        if k < great then
            if A[great] < p then
                exchange A[great] ↔ A[k]
                exchange A[k] ↔ A[less]
                less ← less + 1;
             End
        else
                exchange A[k] ↔ A[great]
            End
        great ← great -1
End

        k ← k + 1
exchange A[left] ↔ A[less - 1]
exchange A[right] ↔ A[great + 1]
```

Figure 1. Yaroslavskiy partitioning algorithm [1].

The modified two pivot-quicksort algorithm

Figure 2 depicts the proposed algorithm for selecting two pivots for Yaroslavisiky's dual pivot-quicksort algorithm. The first pivot position (*f1*), which is selected at the middle of the array, is deterministic. This is done to ensure even partitioning of the array into three regions.The second pivot position (*f2*) was randomly picked from the arrayto provide equal probability to array elements irrespective of input distribution and reduce the possibility of forcing the algorithm to run in quadratic time, i.e., $O(n^2)$. Data in the two positions were compared; the greater is swapped to the right while the lesser is swapped to left. A[left] and A[right] were assigned to pivot one **p** and pivot two **q** respectively.

```
Modified Yaroslavskiy Q sort (A, left, right)
     length ← right - left + 1
     if length ≤ 17 then //Use insertion sort
        do for i ← left + 1 to right do
           key ← A[i]
           j ← i
           while j > 0 and A[j-1] > A[j]
              Exchange A[ j] ↔ A[j – 1]
              j ← j - 1
     return A[j + 1]
  else
     f1 ← length[A]/2
     f2 ← left + random(right – left + 1)
     if f1 == f2
           f2 ← left + random(right – left + 1)

     if A[f1] < A[f2] then
        exchange A[f1] ↔ A[left]
        exchange A[f2] ↔ A[right]
     else
        exchange A[f1] ↔ A[right]
        exchange A[f2] ↔ A[left]
     p ← A[left]
     q ← A[right]
     Yaroslavsky_Partition (A, left, right)

     Modified YaroslavskiyQsort (A, left, less - 2)
     Modified YaroslavskiyQsort (A, great + 2, right)
     Modified YaroslavskiyQsort (A, less, great)
```

Figure 2. Modified Yaroslavskiy's two pivot-quicksort.

Quicksort time, cyclomatic complexity and energy cost model

The quality of quicksort algorithm with one and two pivots was analyzed based on cyclomatic complexity and energy cost. The three quicksort algorithms were implemented using Java programming language. The cyclomatic complexity of the three quicksort algorithms was computed. The three algorithms were partitioned based on the number of pivots selected. The number of pointers for traversing the elements is one number higher than the number of pivots.

Figure 3 illustrates the model for the comparison between quicksort algorithms. Arrays of integers were randomly generated using random function in Java. The randomly generated integers were fed into the Hoare, Yaroslavskiy and proposed modified two pivots quicksort algorithms. Each of the quicksort algorithms sorts the generated integers into ascending order. Sorting times and power consumptions were captured from the algorithms while sorting. The cyclomatic number was computed directly from the code and it is independent of time or vector size of the integers.

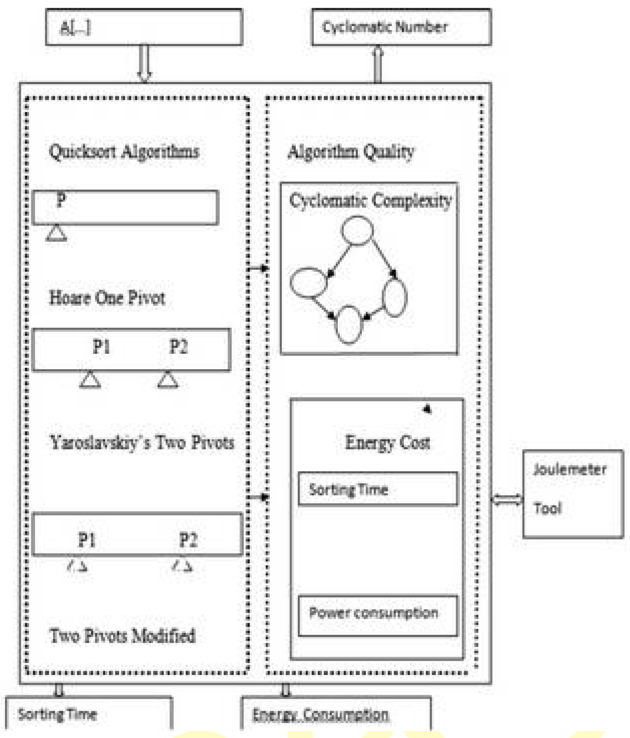

Figure 3. Quicksort time, cyclomatic complexity and energy cost model.

The energy of the algorithms was computed using the following metrics:

i. Quicksort time.
ii. Power consumption.

Quicksort time is the time taken by the algorithm to arrange set of unsorted data into an ascending or descending data. Java timestamps were used to capture the quicksort time in nanosecond. To ensure that the data captured were accurate, each data-set was repeated five times with the three quicksort algorithms and the means were computed.

The power consumed by the algorithms was captured using Joulemeter. Joulemeter was developed by Microsoft Research Laboratory to measure power consumed by a device or software. Measurements taken by the Joulemeter were saved to a file with .csv extension. Nonessential applications running on the computer were terminated to ensure that the measured power is primarily consumed by the sorting algorithm. To ensure that the energy consumed by random integer generation is not included, java timestamps were placed where sorting started and ended. The timestamps were used to trace the power consumed by the sorting algorithm saved in a file with .csv extension. The energy consumption was computed using the equation proposed by Johann *et al* [13]:

$$Energy = Power\ consumption * Quicksort\ time$$

Data-set

The data-set for the algorithms were randomly generated using *rand()* function in Java. The sizes of the dataset range from 50 to 1,000,000 elements. The dataset was purposively categorized into three to understudy the algorithm performance for a range of array size in case there are features peculiar to an array range. The first category ranges from fifty (50) to eight hundred and fifty (850) data size, second category ranges from one thousand (1,000) to one hundred and sixty thousand (160,000) data size while the third category ranges from two hundred thousand (200,000) to one million (1,000,000), representing low, medium and large data sizes. To study the performance of the three algorithms, each algorithm was used to sort same data size and type of randomly generated number. The three algorithms were implemented using Java programming language. Hoare's one pivot, Yaroslavskiy's two pivots and the proposed modified two pivots algorithms were tested on a computer with an Intel Pentium processor with a speed of 1.7 GHz, GB of RAM and a disk size of 500 GB.

Results

(a) Running time

The average running times obtained for each of the three Quicksort algorithms on the three data size categories (small, medium and large data sizes) are shown in Tables 1 to 3 while Figures 4 to 6 are charts of the results.

Table 1. Average running time output of Hoare, Yaroslavskiy and Modified Yaroslavskiy for small array size range 50-850.

Vector size (N)	Quicksort algorithm average running time (μsec)		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
50	1,517	2,039	1,196
250	2,657	3,197	2,172
450	3,543	4,023	3,292
650	4,288	4,790	3,926
850	5,532	5,857	4,997

Table 2. Average running time output of Hoare, Yaroslavskiy and Modified Yaroslavskiy for medium array size range 1,000-160,000.

Array Size (N)	Quicksort Algorithm Average Running Time (μsec)		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
1,000	5,787	6,081	5,063
40,000	35,292	33,598	30,111
80,000	69,634	58,189	54,081
120,000	108,396	98,523	92,153
160,000	175,260	158,436	152,897

Table 3. Average running time output of Hoare, Yaroslavskiy and Modified Yaroslavskiy for large array size with range 200,000-1,000,000.

Array Size (N)	Quicksort Algorithm Average Running Time (μsec)		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
200,000	230,807	204,408	157,139
400,000	319,757	277,945	202,427
600,000	447,544	399,851	308,840
800,000	581,587	474,932	459,273
1,000,000	695,399	547,309	530,810

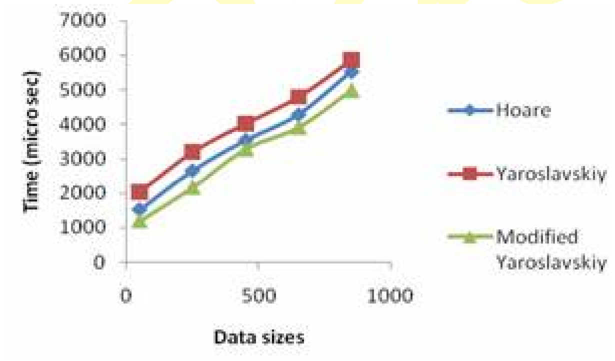

Figure 4. Average running time for Hoare, Yaroslavskiy and Modified Yaroslavskiy for small array size range 50-850.

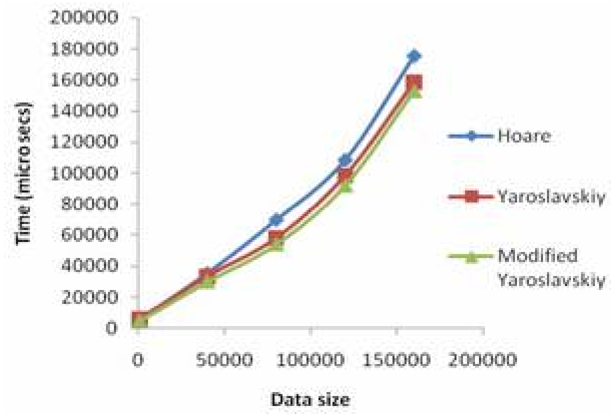

Figure 5. Average running time for Hoare, Yaroslavskiy and modified Yaroslavskiy for medium array size range 1,000-160,000.

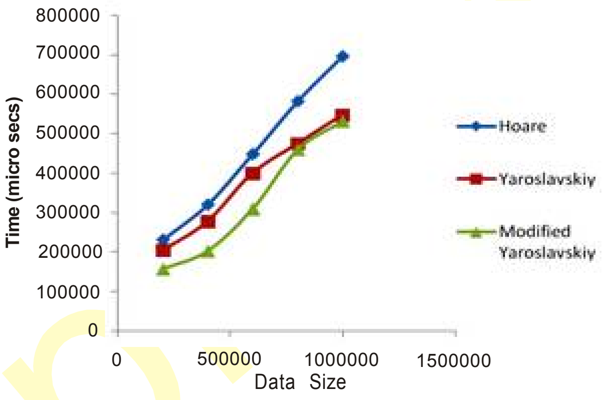

Figure 6. Average running time for Hoare, Yaroslavskiy and modified Yaroslavskiy for large array size range 200,000-1,000,000.

Observations from Figures 4 to 6 show that when the data size was small (50-850, Figure 4), the Hoare's single pivot partition was better than the original Yaroslavskiy's two-pivot algorithm in term of their running time complexities. However, as the data-size increases in Figures 5 and 6, the original Yaroslavskiy's algorithm's time complexity was better. In any case, the time complexity of the proposed modified Yaroslavskiy's algorithm was the best for small and large array data sizes. This means the proposed modified Yaroslavskiy's two-pivot quicksort algorithm takes least time to sort the data, compared to the other algorithms.

(b) Energy consumption

The average energy expended by the three Quicksort algorithms on the three data size categories (small, medium and large data sizes) are shown in Tables 4 to 6 while Figures 7 to 9 are charts of the results.

Table 4. Average energy consumption of Hoare, Yaroslavskiy and Modified Yaroslavskiy for small array sizes ranging from 50-850.

Vector Size (N)	Quicksort Algorithm Average Energy Consumption		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
50	0.009	0.012	0.007
250	0.017	0.021	0.015
450	0.028	0.033	0.027
650	0.036	0.042	0.035
850	0.051	0.058	0.050

Table 5. Average energy consumption of Hoare, Yaroslavisky and Modified Yaroslaviky for medium array sizes ranging from 1,000-160,000.

Array Size (N)	Quicksort algorithm average energy consumption		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
1,000	0.057	0.062	0.053
40,000	0.644	0.619	0.563
80,000	1.327	1.123	1.058
120,000	2.211	2.031	1.917
160,000	3.768	3.454	3.364

Table 6. Average energy consumption of Hoare, Yaroslavisky and Modified Yaroslaviky for large array sizes ranging from 200,000 to 1,000,000.

Vector Size (*N*)	Quicksort algorithm average energy consumption		
	Hoare	Yaroslavskiy	Modified Yaroslavskiy
200,000	5.310	4.800	3.720
400,000	7.700	6.810	5.040
600,000	11.270	10.230	8.300
800,000	15.120	12.580	12.340
1,000,000	18.840	15.080	14.750

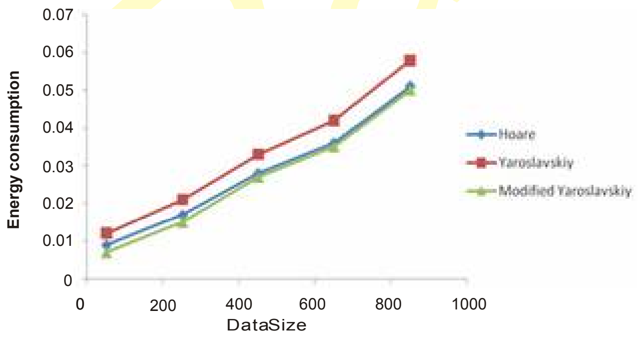

Figure 7. Average energy consumption of Hoare, Yaroslavskiy and Modified Yaroslavskiy for small array sizes ranging from 50-850.

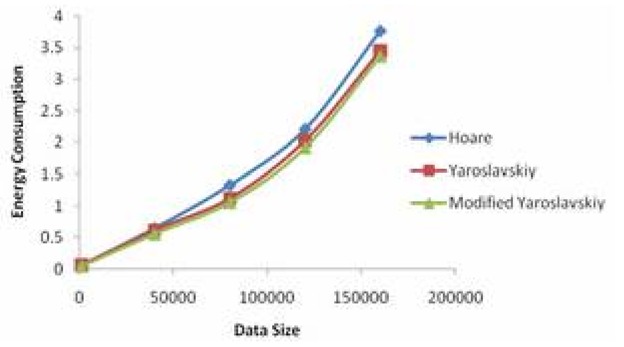

Figure 8. Average energy consumption of Hoare, Yaroslavisky and Modified Yaroslaviky for medium array sizes ranging from 1,000-160,000.

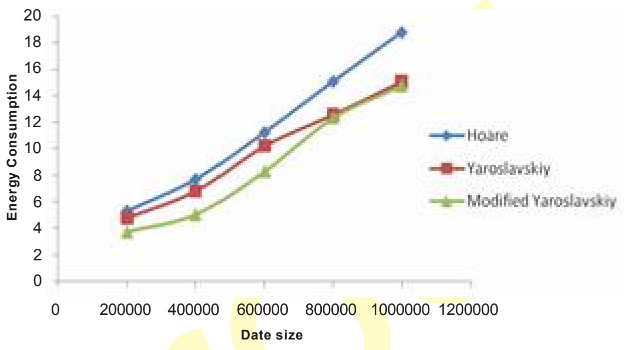

Figure 9. Average energy consumption of Hoare, Yaroslavisky and Modified Yaroslaviky for large array sizes ranging from 200,000 to 1,000,000.

Observation from the energy consumption results of the three Quicksort algorithm's variants shows that the modified Yaroslavskiy two-pivot quicksort expended the least energy with increased vector size for the different data size categories.

(c) Cyclomatic complexity

Figure 10 shows the cyclomatic number of Hoare, Yaroslavskiy and the modified Yaroslavskiy Quicksort algorithms.

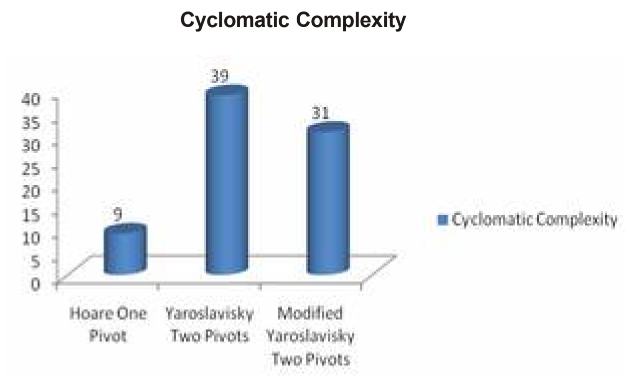

Figure 10. Cyclomatic complexity for Hoare, Yaroslavskiy and Modified Yaroslavskiy.

Observation from Figure 10 shows that the Hoare's one-pivot algorithm has the best (least) cyclomatic complexity followed by the modified Yaroslavisky's two-pivot algorithm. This means that the program code for the Hoare's one-pivot algorithm is less complex, i.e. less prone to bugs and easily understandable.

(d) Correlation analysis

The correlation between the time required to sort arrays of varying sizes and the energy performed was computed. Figure 11 shows the scattered plot graph of energy versus time for the small data size with the variants of the Quicksort Algorithms.

The correlation between the time and energy was high based on the value of coefficient determination, R^2 obtained for the different data sizes and the closeness of the data points to the regression line ($R^2 =$ 0.975, 0.998 and 0.994, respectively for small, medium and large data sizes). The scattered plot shows a linear trend between energy and time. This implied that the time taken alone by Hoare, Yaroslavskiy and modified Yaroslavskiy is sufficient to predict the energy consumed by the quicksort algorithm variants. Hence, the value of energy increases with increase in the value of time. The strength of association between energy and time is very high and the correlation coefficient is very highly significant with $p<0.001$.

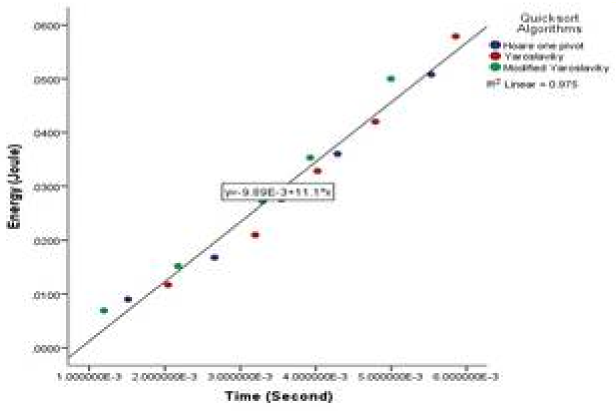

Figure 11. Energy versus time correlation graph for array sizes ranging from small data size.

Discussion of results

The empirical analysis of Hoare's one pivot, Yaroslavskiy and the modified Yaroslavskiy showed that the modified Yaroslavskiy two-pivot quicksort has a better time efficiency for sorting small, medium and large array of numbers. Though the three variants of Quicksort has a best case time complexity of *nlogn*, but their practical running time differs. From the results obtained in this study, the modified Yaroslavskiy algorithm outperforms the original Yaroslavskiy and Hoare Quicksort Algorithms in terms of time for the entire small and large sized array. The result is in line with Kushagra *et al* [4] work, who assert that quicksort algorithm has best case time complexity of *nlogn* but the value of *n* is responsible for the difference in running time.

The energy consumption rate of the three Quicksort algorithm's variants shows that the modified Yaroslavskiy two-pivot quicksort expended the least energy with increased vector size for the different data size categories. This implied that the fastest algorithm (the modified two-pivot Quicksort algorithm) is high energy efficient. This is attributed to less time taken by the algorithm to sort elements in an array for data sizes considered in this work. In the work of Hagar, *et al* [14], it was concluded that the fastest algorithm is energy efficient primarily due to time of execution rather than differences in power consumption levels.

The correlation between energy and time shows a strong influence of time on energy expended by the algorithms. This is in line with Muhammed, Luca and Marco [15], who concluded that algorithm with the lowest execution time expends the least energy. The implication is reducing the execution time of an algorithm increases the energy efficiency.

Hoare's one pivot quicksort is software complexity efficient while Yaroslavskiy and modified Yaroslavskiy require more time to implement due to high cyclomatic complexity. Nurminen [16] pointed out that algorithms with high cyclomatic complexity required more execution paths, harder to understand and maintain.

Conclusion

It was demonstrated in this study that the modified Yaroslavskiy two-pivot is most time and energy efficient compared to Hoare and Yaroslavskiy for small, medium and large data sizes. Yaroslavskiy and the modified version require more human energy to maintain, test and understand compared to Hoare one pivot Quicksort. Time is the key factor in energy consumption of an algorithm as the correlation between energy and time is high.

In conclusion, the proliferation of battery-powered devices, sorting of big data and continuous maintenance of software require algorithms that are most suitable for the application and targeted platforms. Results of this work shows that the number and position of pivot play important roles in determining the performance of Quicksort algorithm. This will help programmers and system designers in selecting appropriate Quicksort algorithm by considering time, energy and cyclomatic complexity.

References

[1] Cormen, T. H., Leiserson, C. E., Ronald L. and Rivest, C. S. 2001. *Introduction to Algorithms, Second Edition,* MIT Press.

[2] Yaroslavskiy V., 2009. Dual-Pivot Quicksort, http://codeblab.com/wpcontent/uploads/2009/09/DualPivot Quicksort. pdf downloaded on 5 August, 2015.

[3] Sedgewick R., 1975. Quicksort, PhD Dissertation, Stanford University, Stanford Computer Science Report, pp. 75- 492.

[4] Kushagra, S., Alejandro L., Aurick, Q. and Munro, J. I. 2014. Multi-Pivot Quicksort: Theory and Experiments, *In Proceeding of the 16th Meeting on Algorithms Engineeing and Experiments*, pp. 47-60.

[5] Kaushik, R., Mark, C. J., 1997. Software design for low power, in low power design in deep submicron electronics. *Kluwer Nato Advanced Science Institutes Series*, Vol. 337, pp. 433-460.

[6] Larsoon, P. 2011. Energy-efficient software guidelines. *Intel Software Solution Group Technology Report.*

[7] Bunse, C., Hopfner, H.., Roychoudhury, S. and Mansour, E. 2011. Energy efficient data sorting using standard sorting algorithms. *Software and Data Technologies, Communications in Computer and Information Science*, 2011, Vol. 50, pp. 247-260.

[8] Bardine, P., Foglia, G., Gabrielli, and Prete, C. A. 2007. Analysis of static and dynamic energy consumption in nuca caches: Initial results. In *Proceedings of the Workshop on Memory performance: Dealing with Applications, Systems and Architecture.* ACM, 2007, pp. 105-112.

[9] Wild, S. and Nebel, M. E. 2012. Average case analysis of Java 7's dual pivot quicksort. In Epstein, L., Ferragina, P. (eds.) ESA 2012. *LNCS,* Springer, Heidelberg, Vol. 7501, pp. 825-836.

[10] Salman, F. S. and Sultanullah. 2011. Multiple pivot sort algorithms: an empirical study. *International Journal of Electrical and Computer Science*, *Vol. 11, No. 3,* pp. 253-261.

[11] Vladimir, V.R. 2011. Methodologies and tools for the software quality assurance course. *Journal of Computer Science in Colleges, Vol. 26, No. 6,* pp. 86-92.

[12] Alsultanny, Y. 2009. Using McCabe method to compare the complexity of object oriented languages. *International Journal of Computer Science and Network Security*, *Vol. 9, No. 3,* pp. 320-327.

[13] Johann, T., Dick, M., Naumann, S. and Kern, E. 2012. How to measure energy-efficiency of software: Metrics and measurement results, *Green and Sustainable Software (GREENS) International Workshop*, *Vol. 1, No.1,* pp. 51-54.

[14] Hager, C. T. R., Midkiff, S. F., Jung-Min, P. and Martin, T. L. 2005. Performance and energy efficiency of block ciphers in personal digital assistants. *The 3rd IEEE International Conference on Pervasive Computing and Communications*, pp. 127-136.

[15] Muhammed, R., Luca, A. and Marco, T. 2015. Energy consumption analysis of image encoding and decoding algorithms. *4th International Workshop on Green and Sustainable Software*, pp. 16-21.

[16] Nurminen, J. K. 2003. Using software complexity measures to analyse algorithms: An experiment with the shortest-paths algorithms. *Journal of Computers and Operations Research*, *Vol. 30,* pp. 1121-1134.

Journal of Science Research
Volume 15, 2016, ISSN 1119 7333
Citation: Akinola, S. O. and Owoeye, F. O.
Energy consumption, cyclomatic and time complexities of variants of quicksort algorithm: A comparative study

Textflow Limited
Ibadan, Nigeria