Green Computing: The impact of Data Size and Power Consumption on Constrained Devices

Ayodele, Oluwakemi Sade, Owoeye, Folusho Olayinka & Idoko Peter Samson
Department of Computer Science
Kogi State Polytechnic
Lokoja, Kogi State, Nigeria
Correspondence Email: kemtemmy2009@gmail.com
Phone: 8069804373

ABSTRACT

In recent years, technological advancement and continuous improvement in computing devices has led to more study in the area of Green Computing with much emphasis on processing time and energy consumption. The study has also been precipitated by wide coverage of Internet of things (IoT), as well as increased global acceptability of mobile and wireless devices (smart device) which are smaller, smarter and ubiquitous. Intensive studies have been in the area of Energy efficiency of Algorithms with much concentration in improving the hardware. However, there has been little or no work in the area of studying, developing or improving the algorithm itself (energy aware algorithm). Also the few available energy models are also hardware based. The aim of this study is to determine through experiment the effect of Data size on Energy, Execution time and power consumption of four sorting algorithm. The four sorting algorithms (Quick, merge, insertion and KSU Hybrid) were implemented using three programming languages (C, Java and Python) and two algorithm implementation styles (Iterative and Recursive). Time stamp was used to capture the execution time of the sorting algorithm. Power consumed was measured using Joule meter. From the experiments, it was observed that the data size, programming language used and algorithm implementation styles affect the execution time. Use of scripting language has clear drawbacks in terms of energy consumption which should be taken into consideration. The experiments carried out show that the way software is written (algorithm implementation style) and the programming language used are both very important determinants of the energy consumption of that software. Therefore, this research work provides information on choice of sorting algorithm type and its algorithm implementation style. This is done in order to minimize energy consumption of an algorithm in the current period of explosive growth in the use of smart phones and hand-held devices which run majorly on battery life. This gives developers knowledge on energy efficiency in software leading to choosing some codes over others based on their energy efficiency.

Keywords: Energy Constraints, Green computing, Sorting Algorithm, Efficiency

CISDI Journal Reference Format
Ayodele, O.S., Owoeye, F.O. & Idoko, P.S. (2023): Green Computing: The impact of Data Size and Power Consumption on Constrained Devices. Computing, Information Systems, Development Informatics Journal. Vol 14 No 2, Pp 61-78.
Available online at https://www.isteams.net/cisdijournal. dx.doi.org/10.22624/AIMS/CISDI/V14N2P5

1. INTRODUCTION

In the recent years emphasis has been on influx of resource-constrained devices which focus mainly on developing energy and power efficient devices and the use of non-toxic materials and minimizing e-waste in such design which is more of hardware and less on the software (Christian et al, 2009). A resource-constrained device is device that has limited processing and storage capabilities, and that often runs on batteries. **Energy efficiency** is using technology that requires less energy to perform the same function.

The goal of efficient energy use is to reduce the amount of energy required to provide products and services (Javier, 2018). Therefore, understanding energy usage/ consumption of an algorithm has become a major issue in determining and improving the energy efficiency of any algorithm. Furthermore, many improvements have been introduced in sorting algorithms during the last decade. Sorting is the process of arranging the elements in some ordered sequence which can either be in ascending, descending or lexicographic order (Deepthi, 2018). To facilitate some other operations such as searching, merging and normalization sorting is often required (Waqas, 2016). It is estimated that more than 25% of all computing time is spent on sorting the keys and some installations spending more than 50% of their computing time in sorting files (Essays, UK.,2013).

As a matter of fact much research has been done on the topic of sorting & searching (Deepthi, 2018, Essays, UK., 2013) but there is no single sorting technique which can be considered the best among the rest. (Essays, UK. , 2013). Bubble sort, selection sort and exchange sort are applicable for small input size, insertion sort for medium input size whereas quick sort, merge sort and heap sort are applicable for an application expecting large to huge data size (Author, 2013) All of the above sorting algorithms are comparison based algorithms and hence cannot be faster than $O(n\log_2 n)$, where O and n have their usual meanings (Kazim, 2017; Owoeye, 2016). For an application, a sorting algorithm is selected according to its computational complexity; ease of implementation and most interestingly recently on its energy efficiency.

There is no any single sorting technique which may be called the best among the rest. Bubble sort, insertion sort, selection sort and exchange sort are applicable for input data of small to medium size whereas quick sort, merge sort and heap sort are applicable for an application expecting large to huge data size (Deepthi et al,2018). Owoeye (2016) improved on the dual pivot Quick sort algorithm, compared the existing Quick sort with modified Quick Sort based on power consumption, energy efficiency, data size and cyclomatic complexity. The empirical study showed that data sizes have effect on power consumption.

Computing devices such as laptops, smartphones, tablets, or other mobile devices, energy consumption is the top priority because they are run on battery, with limited lifespan, as their source of power (kor, 2015). Moreso, most of the energy consumed by theses computing devices is converted into heat, resulting in wear and reduced reliability of hardware components. Also protecting the environment by saving energy and thus reducing carbondioxide emissions is one of today's hottest and most challenging topics (Deepthi, 2018, Bunse C, 2015). From a technological point of view, the realization still falls behind expectations.

Moreso, an efficient sorting technique is important to optimize the design of other algorithms that would need sorted key items to work properly and efficiently. There is no any single sorting technique which may be called the best among the rest. Bubble sort, insertion sort, selection sort and exchange sort are applicable for input data of small to medium size whereas quick sort, merge sort and heap sort are applicable for an application expecting large to huge data size. These sorting algorithms are comparison based and hence can be no faster than O (n log n). There are a few algorithms claiming to run in linear time but for specialized case of input data. So, there is an urgent need of a new sorting algorithm which may be implemented for all input data and it may also beat the lower bound (O (n log n)) of the problem of sorting and also be energy efficient. This work is an effort in that direction.

Proliferation of computing devices which provides quick access to information and fast decision taken is speedily taking over all of human endeavours. Studies in the literature have demonstrated that these computing devices have limited memory and Central processing Unit (CPU) capabilities to process large programs (Niklas et al, 2017) which often accounts for the high time complexity incurred by programs that run on these devices.

As a result, there is a need to develop energy efficiency algorithms to manage the complexity of large data sizes on these computing devices since energy efficiency is an essential design criterion for them**. However, past research studies are in the area of hardware implementations which are prone to external influences with little or nothing in the area of Application/Algorithm, they are usually treated as black box. All experimentations from this study are therefore going to be software based.**

Resource-Constrained devices have become an integral part of our life and provide dozens of useful services to their users. However, usability of these devices is hindered by battery lifetime. Moreso, most of the energy consumed by these systems are converted into heat, resulting in wear and reduced reliability of hardware components. Also protecting the environment by saving energy and thus reducing carbondioxide emissions is one of today's hottest and most challenging topics .From a technological point of view, the realization still falls behind expectations. Therefore, energy conservation and efficiency by applications that run on computing devices is still a big challenge. To this end, developing an energy aware application that can effectively and efficiently be executed remains an open problem.

The aim of this research is to develop a modified Insertion Sort Algorithm that is energy –aware Embarking on this study will help to realize energy-efficient computation (Sorting) of large data sizes. An energy efficient sorting mechanism is important to optimizing the design of other algorithms that require sorted data items in order to build an energy-aware and for it to work correctly (Reusable component). The result of this study will provide an insight into the choice of the appropriate sorting algorithm for system implementation based their requirements. Although a number of researches have been conducted on the development of energy efficient algorithm, in this research, sorting algorithm for large data sizes were considered. Also researches have been conducted on the development of energy efficient algorithms for Wireless Sensor Network (WSN), Java Virtual Machine (Guangyu, 2006), Large scale distributed systems, Wireless sensor networks (Jacob et al, 2018) and so on. An energy aware modified insertion sort algorithm was developed from quick and insertion sorting algorithms. However, in this research, energy-efficiency of large data sizes is considered and implemented using conventional insertion, quick, merge and modified insertion Sorting algorithms.

2. LITERATURE REVIEW

A number of useful Algorithmic related energy efficient model and work have been reported in literature. This section gives a few of such reported and related work. (Ali et. al., 2016) studied the working of five sorting algorithms (bubble, selection, insertion, merge, quick sorts) and compared them on the basis of computational complexities, usage of memory and other computer resourcses, stability, number of swaps and number of comparisons. Quick sort was said to be the faster sorting algorithm while bubble sort is the slowest algorithm. Nothing was discussed on energy efficiency as parameter for comparing these sorting algorithms.

(Totla, 2016) tried to improve upon execution time of the Bubble Sort algorithm by implementing the algorithm using a new algorithm. An extensive analysis has been done by author on the new algorithm and the algorithm has been compared with the traditional methods of —Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort. Were implemented it in C language on GCC complier on Linux operating system implemented in C++ programming languages and tested for the random sequence input of length 10000, 20000, 30000. Observations have been obtained on comparing this new approach with the existing approaches of All Sorts. All algorithms were tested on random data of various ranges from small to large. It has been observed that the new approach has given efficient results in terms of execution time. Through the experimental observations that the new algorithm given in the paper is better than Selection Sort, Insertion Sort, Merge Sort, and Bubble Sort except Quick Sort for larger inputs. However, the data size used is not large enough to give a better comparison of these sorting algorithms.

Deepthi et al (2018) conducted experiments to study how different sorting algorithms have an impact on energy consumption using C language implementation. It was discovered that both time and energy have an impact on the efficiency of these sorting algorithms with quick sort, merge sort and shell sort found to be in the same range of time and energy consumption, followed by insertion and selection sort which is far better than Bubble sort. However, the implementation is only in C language with a non-varying small data size of 10,000. The effect of sorting large data sizes was not considered, varying the algorithm implementation style was also not put into consideration and no modification of algorithm was carried out. Energy models not developed.

(Carroll & Heiser, 2010) performed a detailed analysis of energy consumption of a smartphone, based on measurements of a physical device. How the different components of the device contribute to overall power consumption was analysed. A model of the energy consumption for different usage scenarios was developed, and showed how these translate into overall energy consumption and battery life under a number of usage patterns. The measurement of energy consumption was on smartform as hardware but not on the installed software Adel Noureddine (2014) presented energy models, approaches and tools that can be used to estimate accurately the energy consumption of software at the code level and the application level. JALEN and JALEN UNIT for estimating how much energy each portion of the codes consumes which helped to provide energy information, draws a model of energy consumption evolution of software based on the value of input parameters. However, this study is not RCD centered and with no application to sorting algorithm.

3. METHODOLOGY

3.1 Research Approach

The research approach in this study comprises of three developmental phases which are

i. Experimental acquisition
ii. Designing energy-efficient insertion sort algorithm
iii. Implement energy-efficient sorting algorithms including the modified Insertion sort, Quick Sort, Merge Sort and the conventional Insertion sort algorithms.

3.2 Experimental Data Acquisition

Source: The data were randomly generated by importing random function from the language(s) libraries.

Size and Format: In the algorithm execution, five different integer elements were sorted (100,000; 200,000; 300,000; 400, 000 and 500, 000) which were randomly generated.
To reduce measurement error, each data size been considered was executed five times, the average captured and recorded for use.

3.3 Measurement tools

Joulemeter: Joulemeter is a modeling tools that over the years have been used for measuring the power consumption by servers, virtual machines, desktops, laptops, and software applications running on a computer. The estimates for the power usage of individual components (CPU/Monitor/disk/"base") are provided while running the application. The report consists of power consumption by CPU, monitor, hard disk, and base (base power is the power used by PC even when it is idle). All the numbers are in watts. The total power consumption is also displayed.

TimeStamp: A **timestamp** is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second. A timestamp is the time at which an event is recorded by a computer, not the time of the event itself. In many cases, the difference may

be inconsequential: the time at which an event is recorded by a timestamp (e.g., entered into a log file) should be close to the time of the event.

This data is usually presented in a consistent format, allowing for easy comparison of two different records and tracking progress over time; the practice of recording timestamps in a consistent manner along with the actual data is called **timestamping**

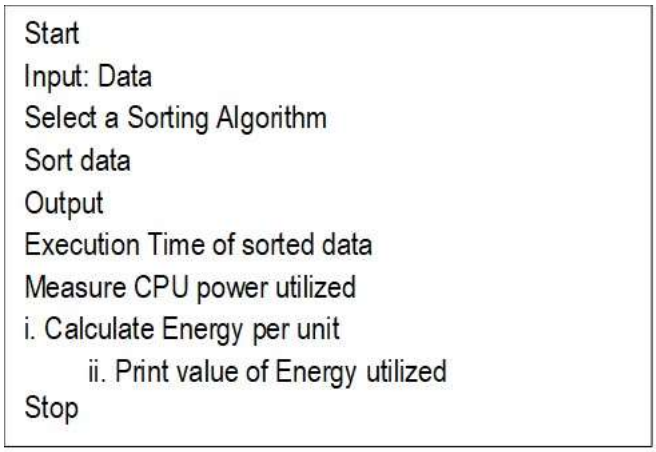

Figure 3.3 : Program flow of the Sorting

3.4 Algorithm

3.4.1 Algorithm Implementation Languages

The three sorting algorithms were run using three different programming languages. This is to:

i. Determine how energy is consumed when changing the implementation language.
ii. Give a clue as to why energy consumption varies between languages.
iii. Determine complexities of the sorting algorithms when changing the implementation language and datasize.

Merge, Quick, Conventional Insertion and the modified sorting algorithms were implemented using Java, C and Python Programming Languages. Merge and Quick sort were implemented using both recursive and iterative programming styles. Insertion was implemented using only iterative programming structure.

The choices of programming languages were categorized based on:

- Virtual machine based languages: Java (compiled and interpreted language)
- Native Languages: C (Compiled language)
- Scripting Languages: Python (Interpreted language)

The Experiments were conducted on the same computer configuration, and running Window 7. The sorting algorithms were run on a laptop with the following configuration, HP 630 Notebook PC, 4GB RAM, Intel(R) Core(TM) i3 @2.53 processor.

3.4.2 Performance Evaluation Metrics

CPU power utilization
This is a measure of Central Processing Unit (CPU) power utilized by Sorting Algorithm.
Merge sort, Insertion sort and Quick sort algorithms were implemented using C, Java and Python. Dev C++ and Net beans Integrated Development Environments were used for C and Java programming languages respectively. Python 3.6.0 was used in the implementation of python codes. T

he sorting algorithms were implemented using recursive and iterative styles. Each of the algorithms was implemented in three programming languages, Merge sort and Quick sort algorithms were implemented using the iterative and recursive version for each programming languages. Data sizes ranges from one hundred thousand (100,000) to five hundred thousand (500,000) at an interval of one hundred thousand. The integers for each data size were randomly generated using the rand() function. The sorting algorithms were placed in functions and classes and called in the main function and class.

Immediately, input data size and click ok to start sorting. The power measurement was stop immediately sorting is completed. The execution time was displayed in millisecond. The start time and end time was used to trace power consumption for the sorting. The experiment is repeated five times for each algorithm using one data size and one programming structure (iterative or recursive). The average of the five experiments was recorded as the value for the data size.

Execution Time
The Execution Time T is the time that it takes for an algorithm to execute.
Execution time for sorting was measured using system clock imported from programming langauge's libraries.

```
Start_Time      Invoke _System_clock
        Call_sortingAlgorithm_class/method
        End_Time  <-  Invoke_System_Clock
        Execution_Time = End_Time -  Start_ Time
```

Figure 3.4: The algorithm for execution time

System. cuurentTimeMillis(), time(Null) and datatime , timeit.default_timer() were time functions used for Java, C and Python Programming Languages respectively. Execution Time was derived by subtracting Start time from end time. Measured times were in seconds.

Algorithm
Conventional Insertion Sort
Insertion sort is an iterative sorting algorithm. The main idea of this is that at each iteration, insertion sort removes an element, find its ordered position in the sorted array of the previous elements and inserts it there. The algorithm can be written as follows:

```
function insertionSortR(array A, int n)
   if n>1
     insertionSortR(A,n-1)
     x = A[n]
     j = n-1
     while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j-1
     end while
     A[j+1] = x
   end if
 end function
```

Figure 3.5 Algorithm of the Conventional Insertion Sort

Quick Sort Algorithm

Quick sort is a divide and conquer algorithm. It first divides a large array into two sub-arrays with respect to a pivot element, where all elements of one sub-array is not more than the pivot element, and those of the other are not less than that.

Then it does the same for the two sub-arrays and continue to do so until a stage is reached where all sub-arrays are of size 1. Since all these sub-arrays are now sorted trivially, merging these will result in completion of the sorting process. The algorithm to sort the pth to rth of the array A is as follows.

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p – 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo - 1
    for j := lo to hi - 1 do
        if A[j] ≤ pivot then
            i := i + 1
            swap A[i] with A[j]
    swap A[i+1] with A[hi]
    return i + 1
```

Figure 3.6 Algorithm of Quick Sort

Merge Sort

Step 1 – if it is only one element in the list it is already sorted, return.

Step 2 – divide the list recursively into two halves until it can no more be divided.

Step 3 – merge the smaller lists into new list in sorted order.

Figure 3.7 Algorithm for Merge Sort

Modified Insertion Sort

Quick sort is very popular since it is the fastest known general sorting algorithm in practice which provides best run-time in average cases while Insertion sort, on the other hand, works very well when the array is partially sorted and also when the array size is not too large. Using the partition approach of the quicksort algorithm the sorting procedure is commenced until we get to a sun arrays whose size is less than a given cut-off-size which distinguishes between the small and large arrays. At the end of this process, we have an array constituting of sub-arrays of sizes less than or equal to the cut-off size that are not sorted themselves but as a whole they are sorted.

```
QUICKSORT (A, p, r)
If p<r
      q= PARTITION (A, p ,r)
      QUICKSORT(A, p, q)
      QUICKSORT(A, q + 1, r)
PARTITION (A, p, r)
x = A[p]
i = p – 1
j = r + 1
while TRUE
   repeat
      j =  j -1
   Until A[j] <= x
   repeat
      i = i +1
   Until A[i] >= x
   If (i< j) exchange A[i] with A[j]
   else return j
```

Figure 3.8: Improved Quicksort Using First Element As Pivot

```
MODINSERTSORT (A, p, r, k)
If (p < r)
      If (r - p  + 1 > k)
         q = PARTITION (A, p, r)
          MODINSERTSORT (A, p, q, k)
          MODINSERTSORT (A,q + 1, r, k)
INSERTIONSORT(A)
```

Figure 3.9: Modified Insertion Sort

The next procedure was that insertion sort was performed over the entire array to get a sorted result. In the process of running the insertion, binary search is used for comparison which from literature will lead to a time complexity of 0(nlogn) as compared to $0(n^2)$ in the worst case. In this research, we will try to combine these two algorithms (Quick sort and insertion sort) in other to take advantage of the strength of these two sorting techniques (the speed of quick sort and also the benefit of effectiveness of insertion sort) using binary search for comparison. Afterwards, modified insertion sort (ModInsertSort) algorithm (combination of insertion and quick), which is optimum in the sense of minimum average run-time since reducing the execution time has a great effect on the total energy consumption of any computing device.

Algorithm Energy $Ea = P \text{ X } T$
where $P =$ CPU Power utilization
and $T =$ the execution time.

3.5 Software Setup

Merge sort, Insertion sort and Quick sort algorithms were implemented using C, Java and Python. Dev C++ and Net beans Integrated Development Environments were used for C and Java programming languages respectively. Python 3.6.0 was used in the implementation. The sorting algorithms were implemented using recursive and iterative styles. Each of the algorithms was implemented in three programming languages, Merge sort and Quick sort algorithms were implemented using the iterative and recursive version for each programming languages. Data sizes ranges from one hundred thousand (100,000) to five hundred thousand (500,000) at an interval of one hundred thousand.

The integers for each data size were randomly generated using the rand() function. The sorting algorithms were placed in functions and classes and called in the main function and class. Time stamp was placed directly above the called function containing the sorting algorithm and another time stamp was placed directly below the called sorting function. This is to ensure that the execution time captured is solely for the sorting algorithm. Execution time was derived by subtracting the start time from end time. Power consumed was measured using Joule meter.

To carry out the experiment, launch Joule meter and the IDE (Dev C++ or Net beans). Run the program, a request to input data size was displayed. Click on browse on Joule meter and specify file name to store power consumed by sorting algorithm per millisecond and recorded. Click on the start button on the Joule meter to record power measurement. Immediately, input data size and click ok to start sorting. The power measurement was stop immediately sorting is completed. The execution time was displayed in millisecond.

The start time and end time was used to trace power consumption for the sorting. The experiment is repeated five times for each algorithm using one data size and one programming structure (iterative or recursive). The average of the five experiments was recorded as the value for the data size.

4. DATA PRESENTATION AND DISCUSSION OF RESULTS

4.1 Discussion of Results on the Impact of Data Size on Energy Consumption (Energy versus Data Size)

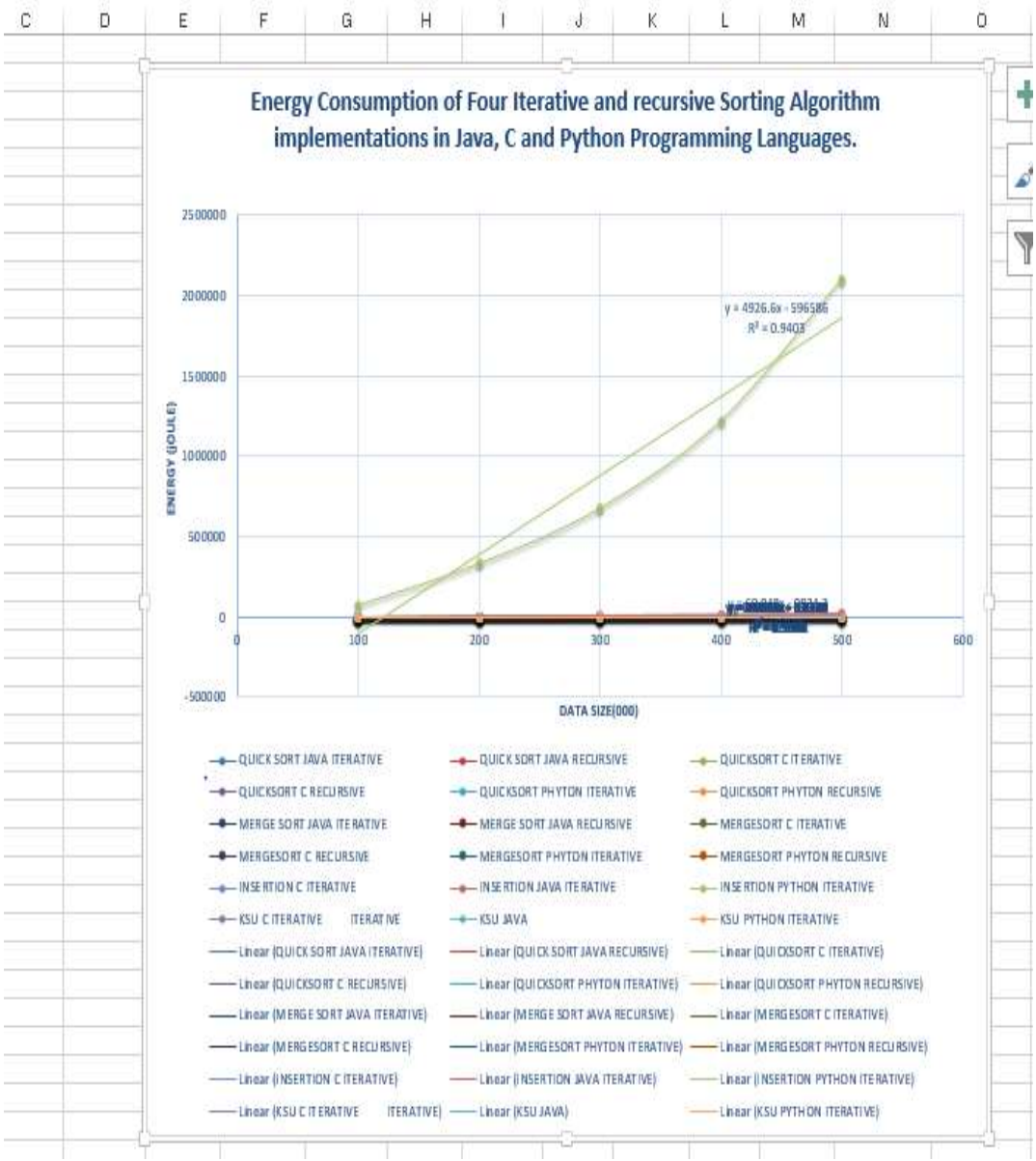

Figure 4.1: Energy Consumption of Four Iterative and recursive Sorting Algorithm implementations in Java, C and Python Programming Languages.

Therefore, from these observations,
There is increase in Energy usage as data size increases in the four sorting algorithms considered including the KSU Hybrid sorting algorithm that was developed.

1. There is an increase in Energy usage as data size increases in the three programming languages used
2. Also, there is an increase in energy consumption in the two algorithm implementation styles used as the data size increases.
3. Generally, from these observations, the table and figures indicate decrease in energy efficiency (increased energy consumption) with increase in data size.
4. This implies that both datasize and energy have strong impact on the efficiency of the algorithm.

4.2 Discussion of Result of Impact of Data Size on Execution Time (Execution Time versus Data Size)

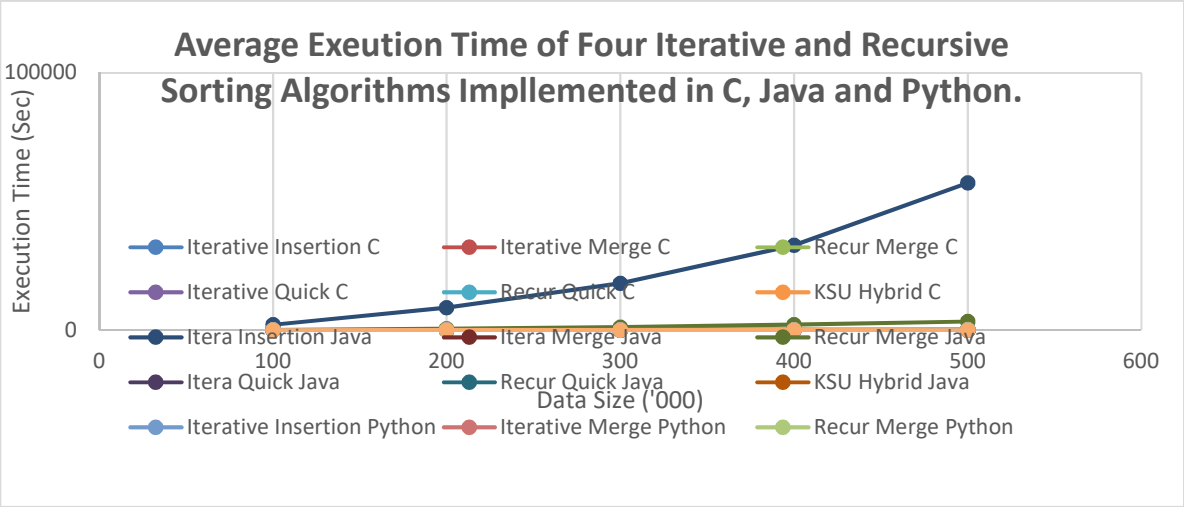

Figure 4.2: Average Execution Time Comparison of FourIterative and Recursive sorting Algorithms implemented in C, Java and Python Programming languages

Checking the trend in the shape of the graph, there is a continuous linear flow of the graphs in these figures which implies that as the data size increases the execution time also increases.
Therefore,

1. The Average Execution Time increases with increase in data size
2. Irrespective of the Algorithm implementation style used, statement in (1) above is upheld.
3. Irrespective of the Programming language used, statement in (1) above is upheld.
4. Average Execution time of recursive implementations of all the sorting Algorithms and in all programming languages used are higher than its equivalent iterative implementations.
5. The execution time for the KSU hybrid sort is lower in almost all cases than other sorting algorithm used and in all programming languages used.
6. From the figure presented both the execution time and energy have an impact on the efficiency of the algorithms.

4.3 Discussion of Result of Impact of Data Size on Power Usage (Power versus Data Size)

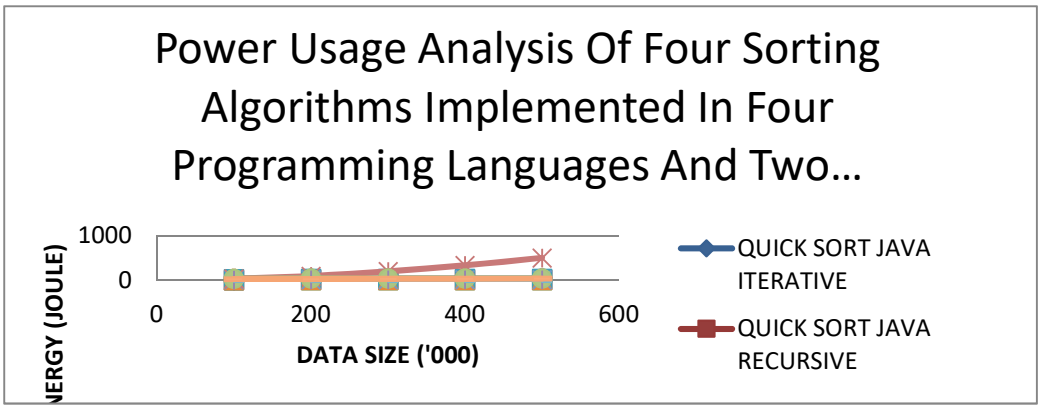

Fig. 4.3: Power Usage Analysis

The Power consumption over various levels of data sizes was plotted. The power consumption of four different sorting algorithms, their relative implementation in three distinct languages and two algorithm implementation styles were analysed. The power consumption for iterative merge and quick sort is relatively stable with increasing data size. However, insertion sort consumes high quantity of power with increase in data size. From the graphs, insertion sort is not suitable for large data size. The KSU hybrid sort has the lowest power consumption. Power consumed by Merge sort and Quick sort implemented in C fluctuates with increasing data size while insertion sort is relatively stable with increasing data size though with high power consumption. From the graph, Quick sort is more power friendly.

Power consumed by iterative insertion, merge and quick sorts implemented in python is high compared to other programming languages. Merge sort consumed less power compared to quick and insertion sort. Insertion sort had the highest power consumption. Power consumption for recursive merge and quick sorts implemented in Java, recursive quick sort consumed lesser power compared to recursive merge sort. Hence, recursive merge sort is more power efficient. Power consumption for recursive merge and quick sort implemented in C, quick sort is unstable with smaller data size but consumed lesser power compared with merge sort. Recursive merge sort implemented in python consumed very high power compared to recursive quick.

The result shows that recursive merge implemented in python is not suitable for platforms where energy is highly prioritized. Insertion sort implemented in Java consumed a noticeable high power compared to iterative merge sort, recursive merge, iterative quick and recursive quick sorts. Insertion sort implemented with java is not suitable for power critical devices. Data size, algorithm, programming language and algorithm implementation style all affect the power consumption of an Algorithm.In all programming languages used and implementation styles, KSU hybrid sort has the lowest power consumption.

4.4 Data Presentation and Discussion of Result on Impact of Programming Languages

The following were observed:

1. The Energy consumed by same algorithm varies from one programming language to the other.
2. As data size increases, the energy consumption increasing in the three programming languages.
3. The Energy consumed by the scripting language (Python) is significantly higher than the virtual based language and also to the native code language.

4. It was noted the high energy consumption of the scripting language which can be attributed to the fact that there is the need to interpret then execute the algorithm. This additional step clearly has a higher value in terms of energy consumption.
5. Programming languages are diverse in terms of design and specifications. The experiment on programming languages with this diversity gives us hints as to why energy consumption varies from one programming language to another.

4.5 Discussion of Result on Impact of Algorithm implementation Style (Iterative and Recursive)

4.5.1 Impact of Algorithm implementation Style (Iterative and Recursive)

Observations:

From figure 4.6, the Insertion sorting algorithm using the Iterative algorithm implementation style implemented in python programming language has the highest energy consumption. As the data size increases, the energy consumption also increases. It should be noted as earlier on stated that insertion sort algorithm was only implemented using the iterative algorithm implantation style. In all other sorting algorithms used implemented in the three programming languages, the energy consumption of the recursive version is higher than the iterative. Checking the trend in the shape of the graph,

The following were observed:

- The algorithm under review determines the energy efficiency of a particular algorithm implementation style.
- Generally, iterative implementation of sorting algorithms are more energy efficient than equivalent recursive implementation.
- The recursive insertion algorithm implemented in Python has the highest energy consumption. This implies that this sorting algorithm implemented in this language is not energy efficient.

Relationship between Energy Usage, Data Size, Algorithm Implementation Style and Programming language used.

It is worth noting that Sorting Algorithms have looping structures. If the loops iterate n times, then the time for all n iterations is T1. N where T1 indicates number of times the computation takes place in one loop iteration. The value of T1 is dependent on programming language used, compiler or interpreter used in translating source code to machine code, programming structure, algorithm implementation style and other factors. While the T1 has negligible effect on computing the asymptotic complexity, the impact is evident in actual implementation of the algorithms.

The impact of T1 on the execution time for each algorithm implemented accounts for differences in behavior. Same algorithm implemented with different programming languages and algorithm implementation styles have slight differences in execution time. Obviously, T1 is not negligible but key factors to be considered in algorithm implementation. However, it is interesting to see the differences in execution time between iterative and recursive variants of the same algorithms. These lead to the assumption that the use of recursion might improve performance, but will definitively increase energy consumption. An Iterative algorithm will be faster than the Recursive algorithm because of overheads like calling functions and registering stacks repeatedly. Many times the recursive algorithms are not efficient as they take more space and time. The results show that the iterative version of the Sorting Algorithms used are more energy efficient than the recursive version. In conclusion we can say that energy consumption is strongly related to execution time.

From all the Figures presented it can be seen that Data size, Algorithm implementation Style and programming language are factors that impact the energy consumption of Algorithm. It can be noted that the same sorting algorithm has different energy consumption with varying data sizes, when implemented in different programming language or/and when implemented in different algorithm implementation styles. The higher the data size, the higher the power consumption, the higher the time taken for sorting and invariably the higher the energy consumption. Also how software is written and the programming language used are important factor for energy efficiency. Varying the parameters values (Data size, algorithm implementation style, programming language) impacts the energy consumption with different evolution patterns based on which parameters are varied.

5. CONCLUSION AND RECOMMENDATION

Energy management plays a major role in determining the Sorting algorithm to be used during the course of building a system. Energy management has become an important issue in computing environments. Therefore, measuring the energy consumption of software is the first step in order to produce an energy efficient code to complement other hardware and system-based approaches. Developers over the years made significant effort to optimize hardware component in pursuant of an energy efficient device treating the algorithm as a black box. The research work analyzed the energy, time execution and power consumption of four different sorting algorithms, the relative implementation in three distinct languages, and two implementation styles over an average of five data sizes. The Data size, Programming language, the implementation algorithm style are factors that impact the energy consumption of software. Varying the parameters values impacts the Energy consumption with different evolution patterns based on which parameters are varied.

This study therefore provides the basic information to choose a specific sorting implementation to minimize energy consumption and reduce software complexities most especially in any handheld battery driven devices. This research work is an eye opening to factors that contributed to Energy efficiency of any Algorithm.

REFERENCES

1. Aaron Carroll and Gernot Heiser (2010) “An analysis of power consumption in a Smartphone”. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC’10, pages 21–21, Berkeley, CA, USA, 2010.USENIX Association, pp 4,36. Retrieved [online] https://orinuxeo.univlille1.fr/nuxeo/site/esupversions/5dd79154-0514-469fbd60-899cf9f57035. Retrieved on 9/9/2017.
2. Abulnaja, Osama & Ikram, Muhammad & Al-Hashimi, Muhammad & Saleh, Mostafa. (2018). Analyzing Power and Energy Efficiency of Bitonic Mergesort Based on Performance Evaluation. IEEE Access. 6. 42757-42774. 10.1109/ACCESS.2018.2861571, Vol. 6, pg 42757-42774.Retrievedfrom https://www.researchgate.net/publication/326726090_Analyzing_Power_and_Energy_Efficiency_of_Bitonic_Mergesort_Based_on_Performance_Evaluation on 26/02/2019
3. Adel Noureddine (2014)”Towards a better understanding of the energy consumption of software systems”[cs.SE]. Universite des Sciences et Technologie de Lille I. 2014. [Online] https://tel.archives-ouvertes.fr/tel-00961346v2.Retrieved on 9/9/2016.
4. Aditya Dev Mishra & Deepak Garg.(2008). “Selection of Best Sorting Algorithm ".In International journal of intelligent information Processing.II (II).pp. 363-368.[Online] http://academia.edu/1976253/Selection_of_Best_Sorting_Algorithm.Retrieved on 29/11/2017.

5. Ahmad Elkahlout and Ashraf Y. A. Maghari (2017). "A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting" In International Research Journal of Engineering and Technology (IRJET) e-Volume:04 Issue:01 Jan 2017 (PDF Download Available). Available from: https://www.researchgate.net/publication/314753240_A_comparative_Study_of_Sorting_Algoriths_Comb_Cocktail_and_Counting_Sorting . Retrieved on18/11/ 2017].
6. Ali, Waqas; Islam, Tahir; Rehman, Habib Ur; Ahmed, Izaz; Khan, Muneeb; and Mahmood, Amna (2016) "Comparison of Different Sorting Algorithms", International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE), Volume 5, Issue 7, [Online] ijarcsee.org/index.php/IJARCSEE/article/view/554/526 on 13/03/2019
7. Antti P. Miettinen and Vesa Hirvisalo (2017)"Energy-efficient parallel software for mobile hand-held devices" [Online] from https://www.usenix.org/.../energy-efficient-parallel-software-mobile-hand-held-device. Retrieved on 3/7/2018s
8. Aremu D.R, O.O. Adesina, O.E. Makinde, O. Ajibola and O.O. Agbo-Ajala (2014). "A Comparative Study of Sorting Algorithms" In African Journal. of Comp & ICTs. 2016, Vol 6, No. 5. Pp 199-206. [Online] fromhttps://pdfs.semanticscholar.org/7b3d/18ae3348bb5dc6fda9b90e0cb07083731218.pdf. Retieved on 18/11/2017
9. Author S.B, Jr. (2013) "Modified Insertion Sort Algorithm: Binary Search Technique". [Online] from https://www.ukessays.com/essays/computer-science/modified-insertion-sort-algorithm-binary-search-1967.php?on 29/08/2018
10. Ayodele, Oluwakemi Sade and Oluwade, Bamidele (2019), A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms Using Three Programming Languages I: Execution Time Analysis, Afr. J. MIS, Vol 1, Issue 1, pp 1-18.
11. Bunse, C., Höpfner, H., Roychoudhury, S., Mansour, E.(2009) "Choosing the "best" Sorting Algorithm for Optimal Energy Consumption". In: Proceedings of the 4th International Conference on Software and Data Technologie (ICSOFT 2009), Setúbal, Portugal, July 26-29, vol. 2, pp. 199–206. INSTICC press, Setúbal (2009) (PDF Download Available online) from: https://www.researchgate.net/publication/220738363_Choosing_the_Best_Sorting_Algorithm_for_Optimal_Energy_Consumption Retrieved on Oct 31 2017].
12. Bunse C., Höpfner H., Roychoudhury S., Mansour E (2015). Energy Efficient Data Sorting Using Standard Sorting Algorithms. In International Conference on Software and Data Technologies ICSOFT 2015: Software and Data Technologies PP 247-260 [ONLINE] https://link.springer.com/chapter/10.1007/978-3-642-20116-5_19 retrieved 23/07/2018
13. (Call for paper on) Green Computing, In Software Engineering Aspects of Green Computing In the 28th Annual ACM Symposium on Applied Computing, March 18-22, 2013, Coimbra, Portugal. [online]; http://trese.ewi.utwente.ni/workshops/SEAGC. Retrieved 07/11/2016
14. Carroll, Aaron and Heiser, Gernot (2010). "An analysis of power consumption in a smartphone", In Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010.USENIX Association, pp 4,36. Retrieved [online] https://ori-nuxeo.univlille1.fr/nuxeo/site/esupversions/5dd79154-0514-469fbd60-899cf9f57035 on 13/03/2019
15. Christian Bunse, Hagen Hopfner, Essam Mansour and Suman Roychoudhury," (2009) Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments" In Mobile data Management Systems, Services and Middleware, 2009. MDM '09 Tenth International Conference on May 2009, pp 600-607. Retrieved [Online] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.543.8109&rep=rep1&type=pdf on 29/09/2017.

16. Deepthi .T, Birunda Antoinette Mary (2018) "Time and Energy Efficiency: A Comparative Study of Sorting Algorithms Implemented in C" In International Conference on Advancements in Computing Technologies (IJFRCSCE) -ICACT 2018 ISSN: 2454-4248 Volume: 4 Issue: 225–27 Available @ http://www.ijfrcsce.org. Retrieved 20/5/2018
17. Erik D. Demaine (2016), Jayson Lynch, Geronimo J. Mitano, Nirvan Tyagi " Energy Efficient Algorithms" In ITCS'16 Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science Pages 321-332 , Cambridge, Massachusetts, USA January 14 - 17, 2016. Retrieved [Online] http://dl.acm.org/citation.cfm?id=2840756 on 08/01/2017.
18. Essays, UK. (2013). Modified Insertion Sort Algorithm: Binary Search Technique. Retrieved from https://www.ukessays.com/essays/computer-science/modified-insertion-sort-algorithm-binary-search-1967.php? on 29/08/2018
19. Essays, UK. (2013). Increasing Time Efficiency of Insertion Sort. Retrieved from https://www.ukessays.com/essays/computer-science/increasing-time-efficiency-insertion-6036.php?vref=1 on 29/08/2018
20. Jacob John and Paul Rodrigues (2018) " Energy Efficiency Enhancement Mrthods for Mobile Wireless Sensor Networks: A survey" In IOSR Journal of computer Engineering (IOSR JCE), Volume 20, Issue 2, ver. 11 (Mar-Apr. 2018) PP 59-66 [Online] www.iosrjournals.org Retrieved on 3/6/2018.
21. JavadMemariani, Zuriati Ahmad Zukarnain, Azizol Abdullah and ZurinaMohd. Hanapi (2013)."A Distributed Energy-aware Clustering Algorithm for Life Time Enhancement of Wireless Sensor Network"International Conference on High-Performance Computing HiPC 2013: High Performance Computing - HiPC 2003 pp 216-216
22. Javier Mancebo, Hector Omar Arriaga, Félix García, Maria Ángeles Moraga, Ignacio García-Rodríguez de Guzmán, Coral Calero, (2018) "EET: A Device to Support the Measurement of Software Consumption", Green And Sustainable Software (GREENS) 2018 IEEE/ACM 6th International Workshop on, pp. 16-22, 2018. Retrieved from https://ieeexplore.ieee.org/document/8449823/ on 12/08/2018.
23. Jiayin Li(2012) "Energy- aware optimization for embedded systems with chip multiprocessor and phase change memory". Theses and Dissertation Electrical and Computer Engineering.7.https://uknowledge.uky.edu/ece_etds/7
24. Kazim Ali (2017) "A Comparative Study of well known Sorting Algorithms", In International Journal of Advanced Research in Computer Science.Volume 8, No 1. ISSN No. 0976-5697. [Online] www.ijarcs.info/index.php/Ijarcs/article/download/2903/2886 retrieved on 19/9/2018
25. Kor A, C. Pattinson, I. Imam, I. AlSaleemi and O. Omotosho (2015), "Applications, energy consumption, and measurement," In 2015 International Conference on Information and Digital Technologies, Zilina, 2015, pp. 161-Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7222967&isnumber=7222934 Retrieved on 12/01/2018.
26. Kyriakos Georgiou, Samuel Xavier-de-Souza and Kerstin Eder (2017)"The IoT energy challenge: A software perspective" [Online] from https://arxiv.org/pdf/1706.08817.pdf. Retrieved on 3/7/2018.
27. kyung C.M and S Yoo (2011): Energy Aware system design. Algorithms and Architectures , Springer Dordrecht Heldelberg London, New York, ISBN 978-94-007-1678-0. [Online] from DOI 10.1007/978-94-007-1679-7.Retrieved on 3/7/2018.
28. Lohiya P.B, D.G.Harkut and M.S.Ali (2014) " Implementation of Earliest Deadline First Algorithm for Wireless Sensor Network" In International Journal of Advanced Research in Computer Science, Volume 5, No. 6, July-August 2014. ISSN No. 0976-5697 [Online] from www.ijarcs.info. Retrieved on 3/7/2018

29. Marcus Hahnel (2016) Measuring Energy Consumption for Short Code Paths Using RAPL. [online]. Available: https://www.researchgate.net/.../236325391_Measuring_Energy_Consumption_for_Short_Code_Paths_Using_RAPL. Retrieved, 02/11/2016.
30. Masood Ahmad, Ataul Aziz Ikram, Ishtiaq Wahid and Abdu Salam (2013) "Efficient Sort Using Modified Binary Search-a New Way to Sort" In World Applied Sciences Journal 28 (10): 1375-1378, 2013 ISSN 1818-4952. Retrieved [PDF] from DOI: 10.5829/idosi.wasj.2013.28.10.1715 on 12/08/2018
31. MiloshStolikj, Pieter J.L. , CuijpersJohan and J. Lukkien (2012) "Energy-aware Reprogramming of Sensor Networks Using Incremental Update and Compression" [Online] from https://www.sciencedirect.com/science/article/pii/S1877050912003833. Retrieved on 3/7/2018
32. Mohammed Rashid, L.Ardito, M.Torchiano,(2015) "Energy Consumption Analysis of Algorithm Implementations", In 2015 proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). [online]. Available: http://doi.ieee/computer society.org/10.1109/esem.2015.7321198. Retrieved 04/11/2017
33. Niklas, Karvonen; Lara Lorna, Jimenez; Miguel, Gomez;Joakim Nilsson; Kikhia Basel Salah; Josef Hall berg (2017) Classifier Optimized for Resource-constrained Pervasive Systems and Energy-efficiencyIn International Journal of Computational Intelligence Systems. 2017, 10 -1 1272-1279.10.2991/ijcis.10.1.86 [Online] Retrieved on 3/7/2018
34. Owoeye, F.O(2016). An empirical comparative study of Time, Energy and Cyclomatic complexities of a modified two-pivot quichsort algorithm.Thesis (Msc).University of Ibadan.
35. Oyedepo, S.O (2012) "Energy Efficiency and Conservation Measures: Tools for sustainable Energy Development in Nigeria" In International Journal of Energy Engineering IJEE volume 2, Issue 3 Aug 2012. Pg 86-98 www.ij-ee.org ©World Academic Publishing ISSN 2225-6571 (Pdf) retrieved on 25/01/2018
36. Palem K.V. (2013) Energy Aware Algorithm Design via Probabilistic Computing: From Algorithms and Models to Moore's Law and Novel (Semiconductor) Devices. In: Pinkston T.M., Prasanna V.K. (eds) High Performance Computing - HiPC 2003. HiPC 2003. Lecture Notes in Computer Science, vol 2913. Springer, Berlin, Heidelberg [online] DOI https://doi.org/10.1007/978-3-540-24596-4_23
37. Ray A(2018) "Hybrid Insertion Sort: Runtime Comparison". [Online] https://www.isid.ac.in/~deepayan/ICP2017/projects/Anirban_Ray/report.html.Retrieved on 30/4/2018.
38. Sonis Prasad, Shubham Jaiswal, N.S.V. Shet and P. Sarwesh (2016)"Energy Aware Routing Protocol for Resource Constrained Wireless Sensor Networks" In Proceedings of the International Conference on Informatics and Analytics Article No. 121. [Online].Retrieved from https://dl.acm.org/citation.cfm?id=2982113 on 3/7/2018.
39. Steigerwald, Chabukswar, Krishnan, and De Vega.Creating Energy-Efficient Software Working Paper. Intel Corp 2008.. [online]: http://software.intel.com/en-us/articles/creating-energy-efficient-software-part-1/ . Retrieved 11/10/2017.
40. Surabhi Patel, Moirangthem Dennis Singh, Chethan Sharma (2014) "Increasing Time Efficiency of Insertion Sort for the Worst Case Scenario" In proceedings of Patel IncreasingTE,retrieved from **https://www.semanticscholar.org/paper/Increasing-Time-Efficiency-of-Insertion-Sort-for-Patel-singh/**19b98726fd50ee4ddbb4ee2e1f59bea1301e3d16?tab=references on 21/01/2018
41. Susanne Albers (2010) "Energy Efficient Algorithms" In Communications of the ACM, May 2010, Vol. 53 No. 5, Pages 86-96 [Online] from https://cacm.acm.org/magazines/2010/5/87271-energy-efficient-algorithms/fulltext. Retrieved on 3/7/2018
42. Totla, Jyoti (2016), "Review on Execution Time of Sorting Algorithms - A Comparative Study", International Journal of Computer Science and Mobile Computing, Vol. 5, Issue.11, pp. 158-166.

43. Tiwari V, S. Malik, and A. Wolfe (1994) "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization,"IEEE Transactions on Very Large Scale Integration (VLSI) Systems,Volume 2, Number 4, December 1994. [Online]: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.4890&rep=rep1&type=pdf. Retrieved 16/10/2017.
44. Vi Ngoc-Nha T and Phuong Ha " ICE (2016): A General and validated Energy Complexity Model for Multithreaded Algorithm" In the Conference proceeding The 22nd IEEE International Conference on Parallel and Distributed Systems (ICPADS '16) DOI: 10.1109/ICPADS.2016.0138. Retrieved 11/10/2107.
45. Waqas Ali, Tahir Islam, Habib Ur Rehman, Izaz Ahmed, Muneeb Khan, Amna Mahmood (2016) "Comparison of Different Sorting Algorithms" In International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE) volume 5, Issue 7, ISSN: 2277-9043. Pdf [Online] ijarcsee.org/index.php/IJARCSEE/article/view/554/526. Retrieved on 19/09/2018
46. Yokoyama T, G Zeng, H Tomiyama (2009) "Analyzing and optimizing energy efficiency of algorithms on DVS systems A first step towards algorithmic energy minimization" In. Asis and South Pacific Design Automation Conference, 2009. ASP-DAC [Online]. Available: **DOI:** 10.1109/ASPDAC.2009.4796566. Retrieved on 20/10/ 2017]